

І. Б. Туркін, А. Г. Чухрай, О. В. Лучшева, Т. Г. Дегтярьова

ДИСКРЕТНА МАТЕМАТИКА З ПРИКЛАДАМИ НА PYTHON

Навчальний посібник

Київ
Видавництво Ліра-К
2026

УДК 519.1:004.43(075.8)
Т 87

*Рекомендовано до друку Вченою радою
Національного аерокосмічного університету
«Харківський авіаційний інститут»
(протокол № 5 від 24 грудня 2025 р.)*

Рецензенти:

Волк Максим Олександрович, доктор технічних наук, професор, професор кафедри електронних обчислювальних машин Харківського національного університету радіоелектроніки

Романенков Юрій Олександрович, доктор технічних наук, професор, професор кафедри електронних обчислювальних машин, професор кафедри економічної кібернетики та управління економічною безпекою, проректор з наукової роботи Харківського національного університету радіоелектроніки

Туркін, І. Б.

Т 87 Дискретна математика з прикладами на Python : навч. посіб. / І. Б. Туркін, А. Г. Чухрай, О. В. Лучшева, Т. Г. Дегтярьова. – Київ : Видавництво Ліра-К, 2026. – 232с.

ISBN 978-617-8883-07-2

Навчальний посібник присвячено систематичному викладу основ дискретної математики з акцентом на їх практичну реалізацію в галузі комп'ютерних наук. Розглянуто фундаментальні розділи: теорію множин, відношення та функції, алгебру логіки, булеві функції, числення висловлень та предикатів, а також елементи комбінаторики. Особливістю видання є інтеграція теоретичного матеріалу з прикладами програмної реалізації мовою Python. Наведено алгоритми виконання операцій над множинами, побудови таблиць істинності, синтезу поліномів Жегалкіна, роботи з логічними стеками та розв'язання комбінаторних задач.

Призначено для студентів технічних спеціальностей закладів вищої освіти, аспірантів та викладачів, які цікавляться математичними основами інформаційних технологій.

УДК 519.1:004.43(075.8)

ISBN 978-617-8883-07-2

© Туркін І. Б., Чухрай А. Г.,
Лучшева О. В., Дегтярьова Т. Г., 2026
© Видавництво Ліра-К, 2026

ВСТУП

Сучасна інформатика та комп'ютерні науки розвиваються надзвичайно швидкими темпами, охоплюючи дедалі ширші сфери людської діяльності – від інженерії та наукових досліджень до бізнесу, освіти та повсякденного життя. У фундаменті цих дисциплін лежить **дискретна математика** – розділ математики, що вивчає структури з дискретною природою, тобто такі, які складаються з окремих, чітко розмежованих елементів. Саме дискретна математика забезпечує теоретичний базис для моделювання, аналізу й оптимізації алгоритмів, побудови комп'ютерних програм, систем штучного інтелекту, баз даних і мереж.

Вивчення дискретної математики є невід'ємною складовою підготовки фахівців у галузях комп'ютерних наук, програмної інженерії, кібербезпеки та суміжних спеціальностей. Вона формує апарат мислення, що дає змогу оперувати абстрактними структурами та моделювати практичні задачі у вигляді формальних моделей. Без розуміння понять множин, відношень, функцій, логічних висловлень чи булевих функцій неможливо створювати ефективні алгоритми або проєктувати складні інформаційні системи.

Особливу роль відіграє поєднання теоретичного матеріалу з практичними прикладами. Адже абстрактні поняття дискретної математики значно легше засвоювати, коли вони підкріплені реалізаціями у вигляді програмного коду. У пропонованому посібнику для цього обрано мову Python – одну з найпопулярніших і водночас доступних мов програмування, яка активно використовується як у наукових дослідженнях, так і у промислових застосуваннях. Python дозволяє просто реалізовувати алгоритми, візуалізувати структури даних і проводити експерименти, що робить його зручним інструментом для вивчення дискретної математики.

Метою цього посібника є подати систематичний виклад основних розділів комп'ютерної дискретної математики, поєднавши теоретичний матеріал з практичними завданнями та додатками на Python. Структура книги відображає логіку вивчення дисципліни:

- у першому розділі розглядаються основи теорії множин, відношень і функцій як базових об'єктів дискретної математики;
- другий і третій розділи присвячені алгебрі висловлень та булевим функціям, які є основою для побудови логічних схем, оптимізації виразів та синтезу цифрових пристроїв;
- четвертий і п'ятий розділи знайомлять із численням висловлень та логікою предикатів, що дозволяє формалізувати знання й будувати обґрунтовані логічні висновки;
- шостий розділ знайомить із комбінаторикою – важливим розділом для аналізу складності алгоритмів, побудови ймовірнісних моделей та оцінки можливих конфігурацій у системах;
- сьомий розділ містить додаткові положення.

Розділи містять не лише теоретичні виклади, а й **прикладі розв'язання задач із застосуванням Python**. Це дозволяє студентам та слухачам:

- навчитися реалізовувати математичні структури у вигляді програмного коду;
- розвивати алгоритмічне мислення через практику;
- перевіряти правильність розв'язань шляхом запуску програмних прикладів;
- поступово переходити від простих конструкцій до більш складних алгоритмів.

Посібник призначений для студентів закладів вищої освіти, які навчаються за спеціальностями комп'ютерного профілю, а також може бути корисним учителям інформатики, молодим науковцям та всім, хто прагне глибше зрозуміти математичні основи програмування.

Вивчення матеріалу з цього посібника не лише розширить теоретичні знання, а й допоможе здобути **практичні навички**, необхідні для роботи з алгоритмами, структурами даних та логічними моделями. У результаті засвоєння курсу читач отримає цілісне уявлення про фундаментальні поняття дискретної математики та навчиться застосовувати їх у програмуванні на Python.

Таким чином, пропонована праця є спробою поєднання класичного підходу до викладу дискретної математики з інноваційною орієнтацією на програмну реалізацію та сучасні інструменти.

1. ОСНОВИ ТЕОРІЇ МНОЖИН. ВІДНОШЕННЯ І ФУНКЦІЇ

1.1. Поняття множини. Способи задання множин

Під **множиною** розуміють будь-який набір певних, відмінних один від одного об'єктів нашої інтуїції чи інтелекту, що розглядаються як єдине ціле. Згідно з цим визначенням у теорії множин вивчають не окремі об'єкти, а їх сукупності як певні утворення.

Об'єкти, що утворюють множину, називаються її **елементами**. Усі елементи множини розрізняються між собою, немає однакових елементів у одній множині. Про множину говорять, що вона містить ці елементи.

Якщо m – елемент множини M , це позначається символом $\in$: $m \in M$. Якщо m не належить множині M : $m \notin M$.

Вважатимемо, що множина A є **підмножиною** множини B , якщо кожен елемент множини A є елементом множини B , тобто належить B :

$$A \subseteq B \Leftrightarrow \forall m \in A \Rightarrow m \in B.$$

Тоді множина B – **надмножина** множини A .

Якщо при цьому може виконуватися умова, що $A \neq B$ або A – *порожня множина* (не має жодного елемента), то A називають *невласною підмножиною* множини B . Нестроге додання припускає рівність двох нескінченних множин.

Порожня множина і сама множина A є невластими підмножинами нескінченної множини A , інші підмножини є власними.

Порожня множина є підмножиною будь-якої множини.

Множини A, B дорівнюють одна одній ($A = B$), якщо вони складаються з однакових елементів. Якщо $A \subseteq B$ і $B \subseteq A$, то $A = B$.

Універсум U (універсальну множину) можна розглядати як надмножину всіх множин. Універсум – поняття відносне. Якщо йдеться про множину чисел в арифметиці, нескінченну множину R дійсних чисел розглядають як U .

Множину можна задати декількома способами, а саме:

1) переліченням елементів у фігурних дужках:

$$A = \{1, 2, 3\},$$

$$B = \{a, b, c, \&, *\};$$

2) з використанням характеристичної властивості:

$$A = \{a \mid a \text{ має властивість } Q\}, \text{ чи } M = \{x \mid P(x)\};$$

$$M611п = \{x \mid x - \text{студент групи 611п}\},$$

$$A = \{x \mid x = 2k, k \in N\},$$

$$M = \{x \mid \sin x = 1\};$$

3) за допомогою опису процедури, що породжує множину, чи операції над множинами (будуть визначені далі):

$$X = (A \cap B) \cup C;$$

4) графічно за допомогою *діаграм Ейлера-Венна*.

На діаграмах Ейлера-Венна універсум позначають прямокутником, усі інші множини – кругами або еліпсами.

Наприклад, якщо $B \subset A$, то це можна проілюструвати так (рис. 1.1):

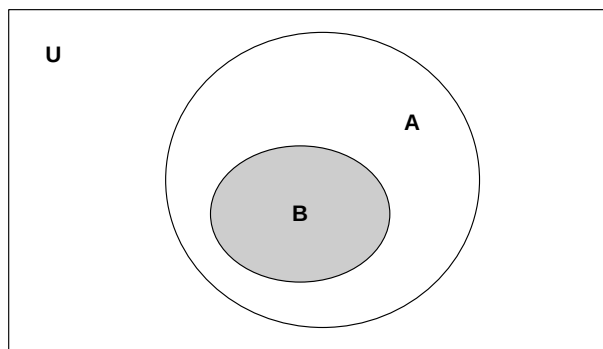


Рис. 1.1. B – підмножина A ; A – надмножина B

Для множин, що часто використовуються в математиці, є спеціальні позначення:

$\emptyset$ – *порожня* множина;

Z – множина *цілих* чисел, $Z = \{\dots, -2, -1, 0, 1, 2, \dots\}$;

R – множина *дійсних* чисел;

N – множина *натуральних* чисел, $N = \{1, 2, \dots\}$;

N_0 – множина натуральних чисел із числом 0, $N_0 = \{0, 1, 2, \dots\}$.

Множина може бути *скінченною*, тобто мати певну кількість об'єктів, *нескінченною* або тільки *зліченою*, якщо її об'єктам відповідають числа натурального ряду.

Потужність множини (або її **кардинальне число**) – кількість елементів скінченної множини. Потужність множини M позначається $|M|$ або $\text{card } M$. Порожня множина $\emptyset$ не містить жодного елемента, та її потужність дорівнює нулю.

Ще одне первинне поняття – **кортеж** – це впорядкований набір елементів, який називається також *вектором*, або *ланцюжком*. Елементи кортежу називають його **компонентами**; їхню кількість – **довжиною (розмірністю)** кортежу.

Компоненти кортежу можуть повторюватися. Їх прийнято нумерувати й записувати у круглих дужках. Наприклад, $X = (1, 2, 3, 4, 5)$ – це кортеж, довжина якого дорівнює п'яти.

Кортежі довжиною 2 називають *двійками (парами)*, довжиною 3 – *трійками*.

Два кортежі **дорівнюють один одному**, якщо вони мають однакову довжину та рівні відповідні елементи.

Приклад 1.1

Дано множину $A = \{1, 2, 3\}$. $|A| = 3$.

Приклад 1.2

Дано множину $A = \{1, 2, \{3\}, \{4\}\}$, які з наведених тверджень правильні:

- 1) $2 \in A$ – правильне, оскільки у множині A є елемент 2;
- 2) $\{1, 2\} \subset A$ – правильне, оскільки у множині A є елементи 1, 2, тобто $1 \in A, 2 \in A$;
- 3) $\{3\} \in A$ – правильне, оскільки у множині A є елемент $\{3\}$;
- 4) $4 \in A$ – неправильне, оскільки у множині A немає елемента 4, тобто $4 \notin A$;
- 5) $\{4\} \in A$ – правильне, оскільки у множині A є елемент $\{4\}$;
- 6) $\{4\} \subset A$ – неправильне, оскільки у множині A немає елемента 4, тобто $\{4\} \not\subset A$?

1.2. Операції над множинами

Вважатимемо далі, що всі розглядувані нами множини – це підмножини якогось універсуму. Визначення всіх операцій над множинами проілюструємо діаграмами Ейлера.

1. **Об'єднанням** двох множин A і B (рис. 1.2) називають множину $A \cup B = \{x | (x \in A) \text{ або } (x \in B)\}$.

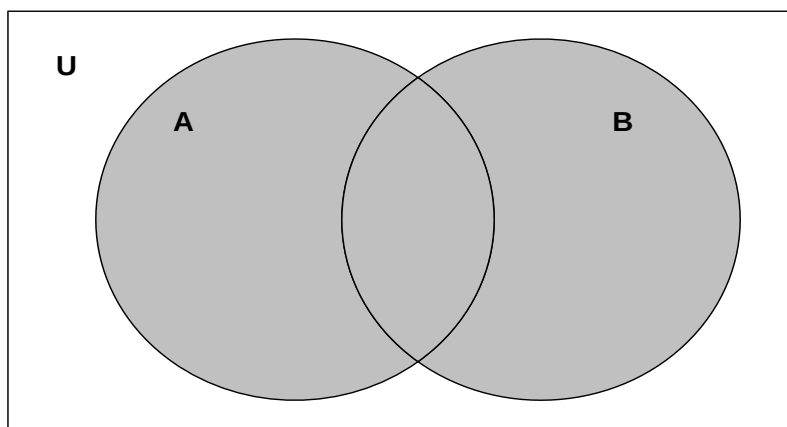


Рис. 1.2. Об'єднання множин A і B

2. **Перетином** двох множин A і B (рис. 1.3) називають множину $A \cap B = \{x | (x \in A) \text{ і } (x \in B)\}$.

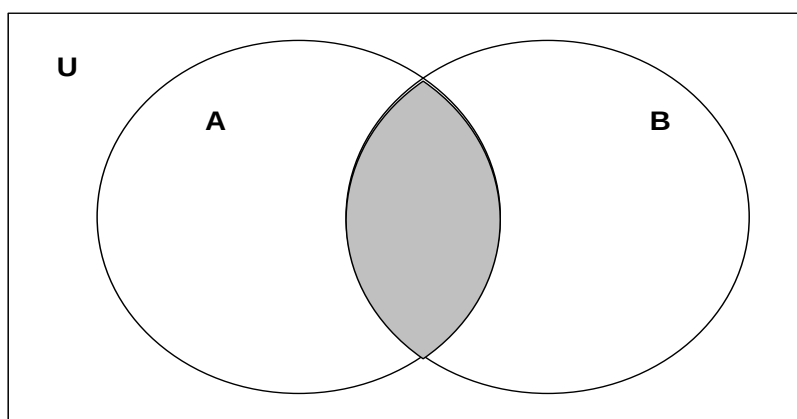


Рис. 1.3. Перетин множин A і B

3. **Різницею** двох множин A і B (рис. 1.4) називають множину $A \setminus B = \{x | (x \in A) \text{ і } (x \notin B)\}$.

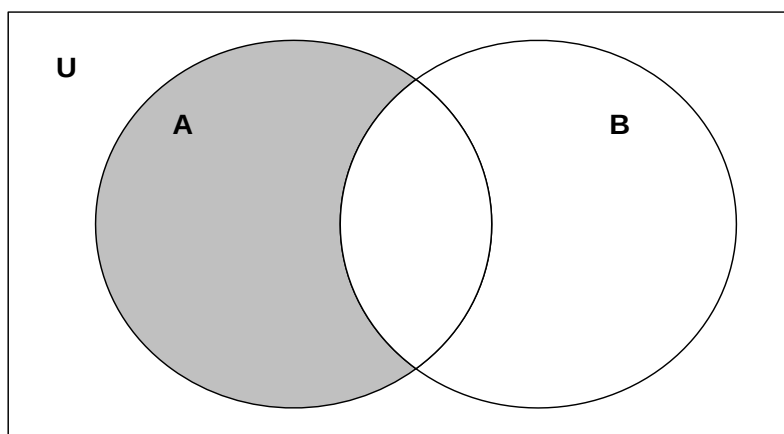


Рис. 1.4. Різниця множин A і B

4. **Доповненням** множини A до множини B (рис. 1.5) називають множину $\overline{A}_B = \{x | (x \notin A) \text{ і } (x \in B)\}$.

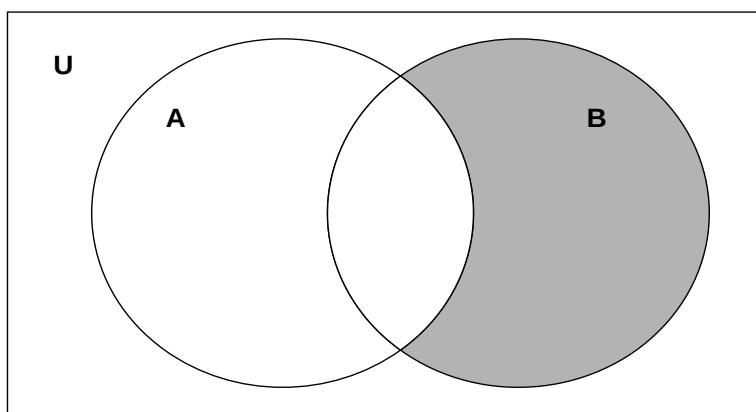


Рис. 1.5. Доповнення множини A до множини B

Доповненням множини A до універсуму U називають різницю

$$\overline{A} = U \setminus A.$$

5. **Симетричною різницею** двох множин A і B (рис. 1.6) називають множину $A \Delta B = A \setminus B \cup B \setminus A$.

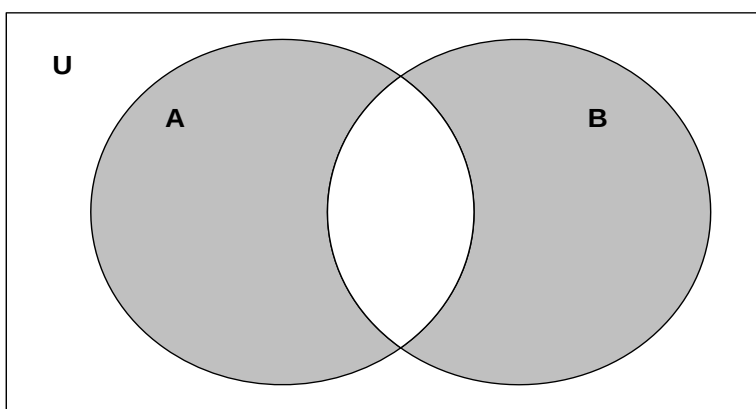


Рис. 1.6. Симетрична різниця множин A і B

6. **Прямим (декартовим) добутком** множин A і B називають множину всіх пар (a,b) , таких, що $a \in A, b \in B$: $A \times B = \{(a,b) | a \in A, b \in B\}$.

Добуток $A \times A = A^2$ називається **декартовим квадратом** множини A , а добуток $\underbrace{A \times A \times \dots \times A}_n = A^n$ – n -ю декартовою степінню множини A .

Приклад 1.3

Нехай $A = \{x, y\}$, $B = \{1, 2, 3\}$. Знайти декартові добутки та степені $A \times B$, $A^2 = A \times A$, $B^2 = B \times B$:

$$A \times B = \{(x, 1), (y, 1), (x, 2), (y, 2), (x, 3), (y, 3)\}.$$

Цей розв'язок можна проілюструвати графічно (рис. 1.7).

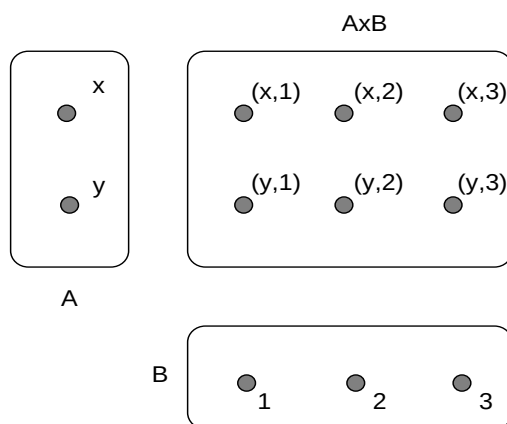


Рис. 1.7. Декартів добуток множин A і B

Аналогічно:

$$A^2 = \{(x, x), (y, y)\},$$

$$B^2 = \{(1, 1), (2, 2), (3, 3)\}.$$

Приклад 1.4

Дано множини $A = \{1, 2\}$, $B = \{3, 4, 5\}$. Знайти їх декартів добуток:
 $A \times B = \{(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5)\}.$

Приклад 1.5

$A = \{a, b, c, d, e, f, g, h\}$, $B = \{1, 2, 3, 4, 5, 6, 7, 8\}$. Що являє собою множина $A \times B$?

$A \times B$ – множина, що містить позначення всіх 64 клітинок шахової дошки.

1.3. Алгебра множин

Алгебра множини A є сукупністю носія N і сигнатури S :

$$A = \langle N, S \rangle,$$

де сигнатура S – набір операцій, що діють за певними правилами.

Множини, що позначаються літерами $A, B, C, \dots$, у сукупності з операціями об'єднання ($\cup$), перетину ($\cap$) і доповнення ($\overline{}$), створюють алгебру множин Кантора $A_K = \langle B(U), \cup, \cap, \overline{} \rangle$.

Розглянемо основні закони та тотожності алгебри множин.

1. Закони комутативності:

$$A \cup B = B \cup A,$$

$$A \cap B = B \cap A.$$

2. Закони асоціативності:

$$(A \cup B) \cup C = A \cup (B \cup C),$$

$$(A \cap B) \cap C = A \cap (B \cap C).$$

3. Закони дистрибутивності (розподільні закони):

$$(A \cup B) \cap C = (A \cap C) \cup (B \cap C),$$

$$(A \cap B) \cup C = (A \cup C) \cap (B \cup C).$$

4. Закон ідемпотентності (повторення або тавтології):

$$A \cup A = A,$$

$$A \cap A = A.$$

5. Операції з константами (порожньою множиною чи універсумом):

$$A \cup U = U, A \cap U = A,$$

$$A \cup \emptyset = A, A \cap \emptyset = \emptyset.$$

6. Закон виключеного третього:

$$A \cup \bar{A} = U.$$

7. Закон протиріччя:

$$A \cap \bar{A} = \emptyset.$$

8. Інволюція (подвійне заперечення):

$$\overline{\bar{A}} = A.$$

9. Закони де Моргана:

$$\overline{A \cup B} = \bar{A} \cap \bar{B},$$

$$\overline{A \cap B} = \bar{A} \cup \bar{B}.$$

10. Правило поглинання:

$$(A \cup B) \cap A = A,$$

$$(A \cap B) \cup A = A.$$

11. Правило склеювання:

$$(A \cup B) \cap (A \cup \bar{B}) = A,$$

$$(A \cap B) \cup (A \cap \bar{B}) = A.$$

12. Закони Порєцького:

$$(A \cup B) \cap (A \cup \bar{B}) = A,$$

$$(A \cap B) \cup (A \cap \bar{B}) = A.$$

Для операцій об'єднання і перетину множин справедливі такі формули обчислення потужності:

$$|A \cup B| = |A| + |B| - |A \cap B|,$$

$$|A \cap B| = |A| + |B| - |A \cup B|.$$

Приклад 1.6

Нехай N – нескінченна множина натуральних чисел, N_0 – нескінченна множина натуральних чисел і нуля. Знайти $N \cup N_0$, $N \cap N_0$, $N \setminus N_0$, $N_0 \setminus N$, $N_0 \Delta N$.

$$N \cup N_0 = N_0, N \cap N_0 = N, N_0 \setminus N = \{0\}, N_0 \Delta N = \{0\}.$$

Приклад 1.7

Довести тотожність $A \setminus (A \setminus B) = A \cap B$.

$A \setminus B = A \cap \bar{B} \Rightarrow A \setminus (A \setminus B) = A \cap \overline{(A \cap \bar{B})}$; за законами де Моргана, інволюції й Порєцького маємо $A \cap \overline{(A \cap \bar{B})} = A \cap (\bar{A} \cup B) = A \cap B$.

Приклад 1.8

Спростити вираз $\overline{(\bar{A} \cup \bar{B}) \cap C} \cup \overline{(A \cup B \cup C)}$ за допомогою законів алгебри множин.

Скористаємося законами де Моргана, подвійного заперечення, поглинання:

$$\begin{aligned} \overline{(\bar{A} \cup \bar{B}) \cap C} \cup \overline{(A \cup B \cup C)} &= (\overline{\bar{A} \cup \bar{B}}) \cup \bar{C} \cup \bar{A} \cup \bar{B} \cup \bar{C} = \\ &= A \cup B \cup \bar{C} \cup \bar{A} \cap \bar{B} \cap \bar{C} = A \cup B \cup \bar{C}. \end{aligned}$$

Приклад 1.9

Пояснити справедливість операцій з константами на діаграмах Ейлера-Венна.

На рис. 1.8 зображено діаграми Ейлера-Венна, що ілюструють операції з універсумом U .

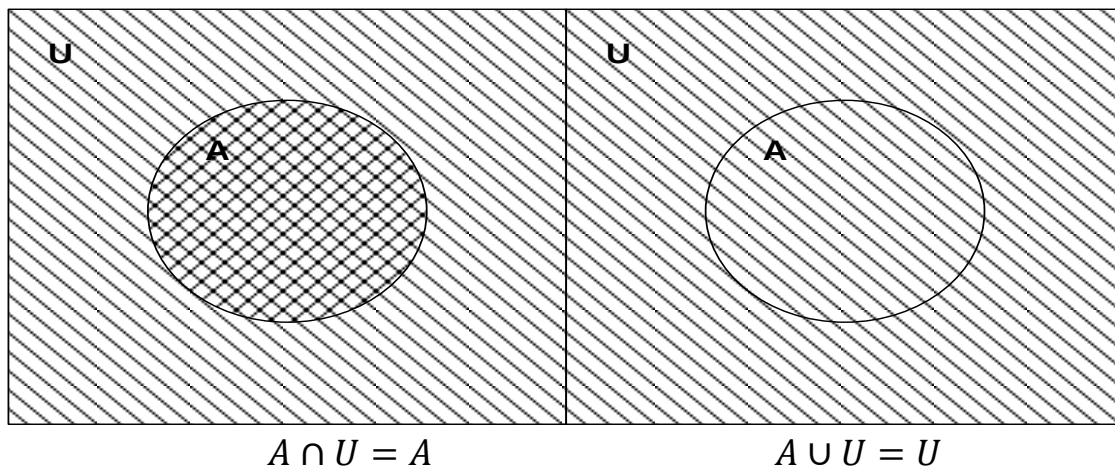


Рис. 1.8. Ілюстрація щодо операцій з константами

Приклад 1.10

Проілюструвати за допомогою діаграм Ейлера-Венна закони дистрибутивності.

На рис. 1.9 і 1.10 зображено діаграми Ейлера-Венна для законів дистрибутивності.

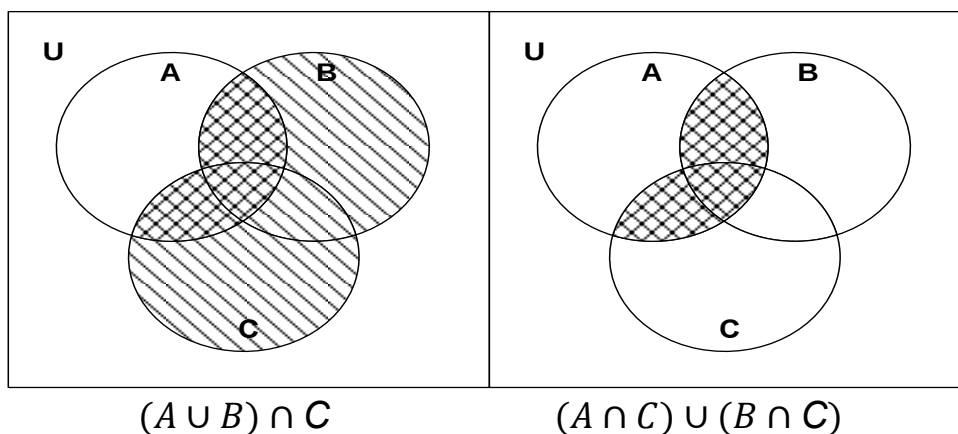


Рис. 1.9. Діаграма Ейлера-Венна закону дистрибутивності для операції перетину $\cap$

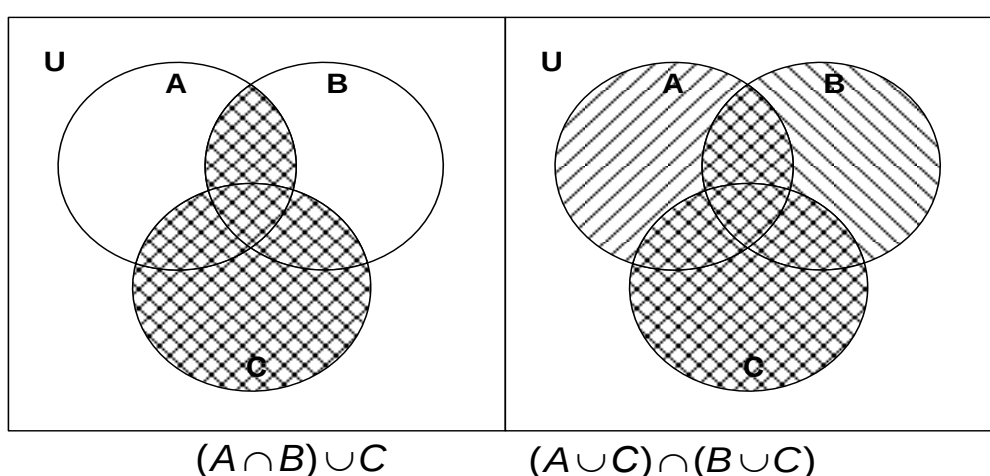


Рис. 1.10. Діаграма Ейлера-Венна закону дистрибутивності

1.4. Відношення

За визначення приймають, що **бінарним (двійковим) відношенням** називається будь-яка множина пар (двійок). Якщо R – відношення, то говорять, що предмети x і y **зв'язані відношенням R** за умови, що пари (x, y) є елементами R , і позначають xRy .

Під **n -місцевим** ($n = 1, 2, \dots$) **відношенням** на довільній множині M розуміють будь-яку підмножину довільної множини $M^n = M \times M \times \dots \times M$.

Приклад 1.11

Розглянемо бінарне відношення « $<$ » (менше) на нескінченній множині натуральних чисел N , яке складається з усіх таких пар (x, y) , що $x < y$. Пари $(1, 2)$, $(2, 3)$, $(4, 6)$ належать цьому відношенню, а пари $(5, 2)$ не належать.

Нехай φ – бінарне відношення на множині M .

Областю визначення D_φ (або $dom(\varphi)$) **відношення φ** називається множина всіх таких a , що $(a, b) \in \varphi$ хоча б при одному b , тобто в символічному записі:

$$D_\varphi = \{a \mid \exists b((a, b) \in \varphi)\}.$$

Областю значень R_φ (або $Im(\varphi)$) відношення φ називається множина усіх таких b , що $(a, b) \in \varphi$ хоча б при одному a , тобто:

$$R_\varphi = \{b \mid \exists a((a, b) \in \varphi)\}.$$

Нескінченна множина $M_\varphi = D_\varphi \cup R_\varphi$ називається **областю задання відношення φ** .

Приклад 1.12

Доведіть усі впорядковані пари, що належать бінарним відношенням на множинах $A = \{1, 3, 5, 7\}$, $B = \{2, 4, 6\}$:

$$U = \{(x, y) \mid x + y = 9\} \text{ та } V = \{(x, y) \mid x < y\};$$

$$U = \{(3, 6), (5, 4), (7, 2)\};$$

$$V = \{(1, 2), (1, 4), (1, 6), (3, 4), (3, 6), (5, 6)\}.$$

Бінарне відношення φ називається:

- 1) **рефлексивним**, якщо $(a, a) \in \varphi$ для будь-якого $a \in M_\varphi$;
- 2) **симетричним**, якщо з $(a, b) \in \varphi$ випливає $(b, a) \in \varphi$;
- 3) **транзитивним**, якщо з $(a, b) \in \varphi$ і $(b, c) \in \varphi$ випливає $(a, c) \in \varphi$.

Приклад 1.13

Відношення « $\leq$ » на множині натуральних чисел N є рефлексивним і транзитивним, але не симетричним, а відношення « $<$ » на цій самій множині тільки транзитивним.

Бінарне відношення, що є одночасно рефлексивним, симетричним і транзитивним, називається **відношенням еквівалентності**.

Приклади відношень еквівалентності:

- 1) відношення подібності на множині трикутників у площині;
- 2) відношення паралельності між прямими на площині.

Приклад 1.14

Як можна пояснити рефлексивність, симетричність або транзитивність відношень:

- а) « x ділить y » на множині натуральних чисел N ;
- б) « $x \neq y$ » на множині цілих чисел;
- в) « x – кількість років, що прожиті людиною за її вік»?

Розглянемо ці відношення.

1. Оскільки x завжди ділиться сам на себе, це відношення – рефлексивне. Воно не є симетричним, оскільки, наприклад, якщо 2 ділить 6, то 6 не ділить 2 націло.

Перевіримо транзитивність. Нехай x ділить y , а y , у свою чергу, ділить z . Тоді $y = mx$, для деякого натурального m , а $z = ny$, для натурального n . Із цього виходить, що $z = ny = (nm)x$, тобто x ділить z . Відношення є транзитивним.

2. Оскільки x завжди дорівнює x , умова $x \neq x$ не виконується і відношення не є рефлексивним. Воно є симетричним, тому що $x \neq y$ тоді й тільки тоді, коли $y \neq x$.

Відношення не є транзитивним, оскільки, наприклад, з $2 \neq 3$ і $3 \neq 2$ не випливає, що $2 \neq 2$. Відношення $x \neq y$ вочевидь є рефлексивним і симетричним.

3. Якщо взяти три людини x, y, z , у яких «кількість років x збігається з кількістю років y », а «кількість років y теж збігається з кількістю років z », то всі вони будуть одного віку. Отже, це відношення є відношенням еквівалентності.

1.5. Функціональні відношення

Відношення R називається **функціональним відношенням**, або **функцією**, якщо для будь-якого x у R містяться рівно одна пара (x, y) з першим елементом x .

Формально це означає: $(x, y_1) \in R \wedge (x, y_2) \in R \Rightarrow y_1 = y_2$, що називають умовою уніформності за другою координатою.

Іншими словами, для будь-якого x з області визначення D_f відношення f існує єдиний елемент y , такий, що $(x, y) \in f$.

Елемент y називають **значенням** функції f на **аргументі** x і позначають через $f(x)$, тобто $y = f(x)$, $y \in R_f$ – область значень f .

Приклад 1.15

Якою множиною можна задати функцію $f(x) = x^2 + 1$, $x \in N$?

Цілочислова функція $f(x) = x^2 + 1$ є множиною, що складається з усіх пар $(a, a^2 + 1)$, де $a \in N$.

Приклад 1.16

Визначити, які з наведених відношень між множинами $A = \{a, b, c\}$ і $B = \{1, 2, 3\}$ є функціями:

$$f = \{(a, 1), (a, 2), (b, 3), (c, 2)\};$$

$$g = \{(a, 1), (b, 2), (c, 1)\};$$

$$h = \{(a, 1), (c, 2), (b, 3)\}.$$

Відношення f не є функцією, оскільки елементу $a \in A$ відповідають два елементи $1, 2 \in B$.

Відношення g – функція.

Відношення h також є функцією.

Функції інакше ще називаються **відображеннями**. Відображення f позначають $f: X \rightarrow Y$, що означає:

1) відображення нескінченної множини X на нескінченну множину Y (X на Y), якщо $X = D_f$ і $Y = R_f$ чи **сюр'єкція** X на Y ;

2) відображення X у Y , якщо $X = D_f$ і $R_f \subseteq Y$;

3) відображення з X на Y , якщо $D_f \subseteq X$ і $R_f = Y$;

4) відображення з X у Y , якщо $D_f \subseteq X$ і $R_f \subseteq Y$.

Відношення $S^{-1} = \{(x, y) | (y, x) \in S\}$ називається **оберненим відношенням** до відношення S .

Відношення, обернене до функції, не завжди є функцією.

Якщо f^{-1} є функція, то функція f називається **бієкцією**, або **оберненою**.

Функція f як бінарне відношення на M називається:

1) **ін'єкцією**, або **взаємно однозначною**, якщо з $f(a) = f(b)$ виходить, що $a = b$;

2) **константною**, якщо $D_f = M$ та існує $b \in M$, що для будь-якого $x \in M$ $f(x) = b$;

3) **тотожною**, якщо для кожного $x \in M$ $f(x) = x$;

4) **відображенням M у M** , якщо $D_f = R_f = M$.

Приклад 1.17

Визначити, до яких різновидів належать наведені функції:

а) функція $f(x) = x^4$, де x – довільне ціле число, є відображенням, але не ін'єктивним, оскільки $f(1) = f(-1)$;

б) функція $f(x) = 2x + 1$, де x – довільне ціле число, є ін'єктивною і відображенням одночасно;

в) функція $f(x) = 5$, де x – довільне ціле число, є константною.

Приклад 1.18

На множині прямих, що є на площині, розглянемо відношення паралельності прямих. Чи буде воно відношенням еквівалентності на цій множині?

Нехай a, b, c – прямі на площині.

$a || a \Rightarrow p$ – рефлексивне;

$a || b \Rightarrow b || a \Rightarrow p$ – симетричне;

$a || b$ і $b || c \Rightarrow a || c \Rightarrow p$ – транзитивне.

Звідси виходить, що p – відношення еквівалентності.

1.6. Вправи до розділу 1

1. Які з наведених нижче теоретико-множинних виразів є правильними і чому:

а) $x \in \{2, a, x\}$;

в) $3 \in \{1, \{2, 3\}, 4\}$;

б) $x \in \{1, \cos x\}$;

г) $\{x, y\} \in \{a, \{x, y\}, b\}$?

2. Чи рівні між собою множини A і B , якщо ні, то чому? У якому відношенні вони знаходяться між собою у випадку їх нерівності:

а) $A = \{2, 5, 4\}$, $B = \{5, 4, 2\}$;

в) $A = \{1, \{2, 5\}, 6\}$, $B = \{1, \{5, 2\}, 6\}$;

б) $A = \{1, 2, 4, 2\}$, $B = \{1, 2, 4\}$;

г) $A = \{1, \{2, 5\}, 6\}$, $B = \{1, 5, 2, 6\}$?

3. Що можна сказати про справедливість цих виразів (відповідь пояснити):

- а) $\{a, b\} \in \{\{a, b, c\}, \{a, c\}, a, b\}$;
- б) $\{a, b\} \notin \{\{a, b, c\}, \{a, c\}, a, b\}$?

4. Навести приклад таких множин A , B і C , що $A \subset B$, але $A \not\subset C$.

5. Описати словесно кожен з наведених нижче нескінченних множин:

- а) $\{x \in \mathbb{Z} \mid x \text{ поділяється на } 2 \text{ і на } 3\}$;
- б) $\{x \mid x \in A \text{ і } x \in B\}$;
- в) $\{x \mid x \in A \text{ або } x \in B\}$;
- г) $\{x \in \mathbb{Z}^+ \mid \text{для деякого цілого } y, x = 2y\}$;
- д) $\{x \in \mathbb{Z} \mid \text{для деяких цілих } y, x = 3y\}$;
- е) $\{x^2 \mid x - \text{просте число}\}$;
- ж) $\{a/b \in \mathbb{Q} \mid a < b, a, b \in \mathbb{Z}\}$;
- з) $\{(x, y) \mid x^2 + y^2 = 1, x, y \in \mathbb{R}\}$;
- і) $\{(x, y) \mid y = 2x \text{ і } x, y \in \mathbb{R}\}$.

6. Чи дорівнюють одна одній множини A і B у наведених прикладах (а – д)? Якщо ні, то пояснити, чому:

- а) $A = \{1, 3, 8\}, B = \{8, 1, 3\}$;
- б) $A = \{5, 8, 9, 10\}, B = \{10, 9, 8\}$;
- в) $A = \{1, 2, 3\}, B = \{3, 2, 4\}$;
- г) $A = \{9, \{6, 7\}, 6\}, B = \{9, \{7, 6\}, 6\}$;
- д) $A = \{9, \{6, 7\}, 6\}, B = \{9, 6, 7, 6\}$.

7. Чи зв'язані множини A і B відношенням включення (якщо так, то вказати, яке з них є підмножиною іншого):

- а) $A = \{a, b, d\}, B = \{a, b, c, d\}$; в) $A = \{c, d, e\}, B = \{c, a\}$;
- б) $A = \{a, c, d, e\}, B = \{a, c, e\}$; г) $A = \{a, b, \{c\}, d\}, B = \{a, b, c, d\}$?

8. Приймаючи множину натуральних додатних чисел від 10 до 35 за універсум, записати такі його підмножини: A – парних чисел, B – непарних чисел, C – квадратів чисел, D – простих чисел. У яких відношеннях знаходяться ці підмножини?

9. Описати словами кожен з цих множин:

- а) $\{x \mid x - \text{точка площини, що знаходиться на відстані } r \text{ від початку координат}\}$;
- б) $\{x \mid x^2 - 4x + 3 = 0\}$;
- в) $\{x \mid x \in A \text{ і } x \in B\}$; A – нескінченна множина космічних об'єктів; B – нескінченна множина планет.

10. Чи правильно, що:

- а) $\{1,2\} \in \{\{1,2,3\}, \{1,3\}, 1,2\}$; б) $\{1,2\} \subseteq \{\{1,2,3\}, \{1,3\}, 1,2\}$?

11. Чи існують такі непорожні множини A , B і C , що:

- а) $A \cap B = \emptyset$, $A \cap C = \emptyset$, $(A \cap B) \setminus C = \emptyset$;
 б) $(A \cap B) \subseteq (A \cap C)$ і $C \subseteq B$;
 в) $(A \cap B) \subseteq (A \cap C)$, $(A \cap C) \subseteq (C \cap A)$, $A \cap B \neq \emptyset$;
 г) $A \cap B \neq \emptyset$, $B \cap C \neq \emptyset$, $A \Delta C \neq \emptyset$;
 д) $A \subseteq C/B$ і $(A \cap B \cap C) \subseteq A$?

12. Використовуючи операції над множинами, записати аналітично зафарбовані штриховкою області на діаграмах Ейлера-Венна (рис. 1.11).

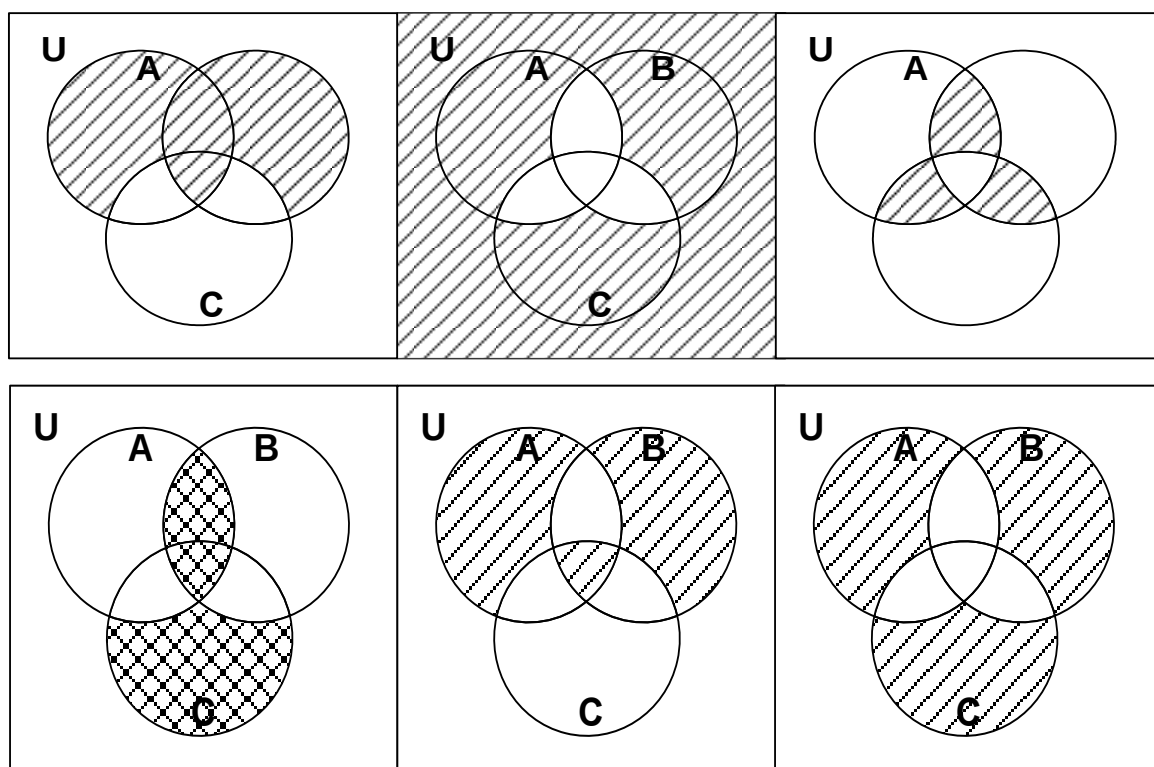


Рис. 1.11. Діаграми Ейлера-Венна

13. Довести або спростувати за допомогою діаграм Ейлера-Венна, що:

- а) $(A \setminus B) \setminus C = (A \setminus C) \setminus B$, де $A \setminus B$ – операція «різниця множин A і B »;
 б) $(A \setminus B) \cup C = (A \cup C) \setminus (B \cup C)$;
 в) $((A \setminus B) = C) \sim (B \cup C = A)$.

14. Довести за допомогою діаграм Ейлера-Венна, що:

- а) $A \setminus (A \setminus B) = A \cap B$;
 б) $A \cup B = A \cup (B \setminus A)$;
 в) $(A \cap B) \cup (B \setminus A) \cup (C \setminus B) = B \cup C$;
 г) $A \setminus B = A \setminus (A \cap B)$;
 д) $(A \cap B) \cap (C \cap D) = (A \cup C) \cap (B \cup C) \cap (A \cup D) \cap (B \cup D)$;

$$\text{е) } A \setminus (B \setminus C) = (A \setminus B) \cup (A \cap C);$$

$$\text{ж) } (A \cap \bar{B} \cap C) \cup (A \cap B \cap \bar{C}) = [A \setminus (B \cap C)] \setminus [A \setminus (B \cup C)];$$

$$\text{з) } (A \cap B) \cup (A \cap \bar{B}) = (A \cup B) \cap (A \cup \bar{B});$$

$$\text{и) } (A \cup B) \setminus C = (A \setminus C) \cup (C \setminus B);$$

к) $A \cup B = (A \triangle B) \cup (A \cap B)$, де $A \triangle B$ – операція «симетрична різниця множин A і B »;

$$\text{л) } (A \setminus B) \triangle B = \bar{A} \cap \bar{B};$$

$$\text{м) } A \cup B = (A \setminus B) \triangle B;$$

$$\text{н) } A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C);$$

$$\text{о) } A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C);$$

$$\text{п) } A \setminus (A \cap B) = A \setminus B;$$

$$\text{р) } A \setminus B = A \triangle (A \cap B).$$

15. Показати, що $(A \cup B) \cap C = A \cup (B \cap C)$, якщо і тільки якщо $A \subseteq C$. Проілюструвати результат діаграмою Ейлера-Венна.

16. Знайти всі підмножини множин:

$$\text{а) } x \mid (3 \leq x \leq 5) \text{ і } (x \in N);$$

$$\text{б) } \{x \mid x^2 = 0\};$$

$$\text{в) } \{ |, \setminus, / \} \cup \{ \setminus, / \}.$$

17. Нехай U – нескінченна множина усіх паралелограмів на площині, A_1 – нескінченна множина квадратів, A_2 – нескінченна множина прямокутників, A_3 – нескінченна множина ромбів на площині. Знайти результати таких операцій: $A_i \cup A_j$, $A_i \cap A_j$, $A_i \setminus A_j$, де $i, j \in \{1, 2, 3\}$.

18. За допомогою діаграм Ейлера-Венна дослідити питання про справедливості кожного з цих міркувань:

$$\text{а) якщо } A, B \text{ і } C \text{ – підмножини } U, A \cap B \subseteq C \text{ і } A \cup C \subseteq D, \text{ то } A \cap C = \emptyset;$$

$$\text{б) якщо } A, B \text{ і } C \text{ – підмножини } U, A \subseteq (B \cup C) \text{ і } B \subseteq (A \cup C), \text{ то } B = \emptyset.$$

19. Описати впорядковані пари, що належать відношенням на множині N натуральних чисел, для кожного з цих відношень:

$$R = \{(x, y) \mid 2x + y = 9\};$$

$$S = \{(x, y) \mid x + y < 7\};$$

$$T = \{(x, y) \mid y = x^2\}.$$

20. Нехай R – відношення на множині $\{1, 2, 3, 4\}$, що визначене такою умовою: uRv тоді й тільки тоді, якщо виконується умова $u + 2v$ – непарне число. Описати відношення R кожним зі вказаних способів:

а) як множину впорядкованих пар;

б) у графічній формі.

21. Визначити, які з наведених нижче відношень на множині людей є рефлексивними, симетричними або транзитивними:

- а) «... має тих же батьків, що і...»;
- б) «... є братом...»;
- в) «... старше або молодше, ніж...»;
- г) «... не вище, ніж...»?

22. Визначити, які з наведених нижче відношень на множині Z є рефлексивними, симетричними, а які транзитивними:

« $x + y$ – непарне число»:

- а) « $x + y$ – парне число»;
- б) « xy – непарне число»;
- в) « $x + xy$ – парне число»;
- г) « $x - xy$ – парне число».

23. Перелічити впорядковані пари, що належать відношенням, заданим на множині $\{x: x \in Z, 1 \leq x \leq 12\}$:

- а) $R = \{(x, y) \mid xy = 9\}$
- б) $S = \{(x, y) \mid 2x = 3y\}$.

24. Для кожного з відношень еквівалентності на вказаній множині A описати підмножини, на які розбивається множина A :

а) A – множина книг у бібліотеці, а R визначається умовою xRy , якщо і тільки якщо колір обкладинки x такий самий, що і колір обкладинки y ;

б) $A = Z$, R задається умовою: xRy тоді й тільки тоді, коли $x - y$ є парним числом;

в) A – множина людей, і xRy , якщо x має ту ж стать, що і y .

25. Нехай R – відношення між множинами $\{1, 2, 3\}$ і $\{1, 2, 3, 4\}$, що задане переліком пар $R = \{(1, 1), (2, 3), (2, 4), (3, 1), (3, 4)\}$.

Крім того, S – відношення між множинами $\{1, 2, 3, 4\}$ і $\{1, 2\}$, що має вигляд $S = \{(1, 1), (1, 2), (2, 1), (3, 1), (4, 2)\}$.

Обчислити R^{-1} , S^{-1} і перевірити, чи є функціями відношення R , S , R^{-1} , S^{-1} .

26. Нехай $A = \{0, 2, 4, 6\}$ і $B = \{1, 3, 5, 7\}$. Які з наведених нижче відношень між множинами A і B є функціями, визначеними на A із значеннями у B :

- а) $\{(6, 3), (2, 1), (0, 3), (4, 5)\}$;
- б) $\{(2, 3), (4, 7), (0, 1), (6, 5)\}$;
- в) $\{(2, 1), (4, 5), (6, 3)\}$;
- г) $\{(6, 1), (0, 3), (4, 1), (0, 7), (2, 5)\}$?

Визначити, до якого типу належать ці відображення.

27. Нехай R – відношення «...батько...», а S – відношення «...брат...» на множині всіх людей. Надати словами короткий опис відношення: R^{-1} , S^{-1} .

28. Нехай $A = \{1, 2, 3, 4\}$ і $B = \{a, b, c, d\}$. Які з наведених нижче відношень між множинами A і B є функціями, визначеними на A із значеннями у B :

- а) $\{(1, b), (2, a), (3, d), (4, c)\}$;
- б) $\{(2, a), (4, a), (3, c), (1, b)\}$;
- в) $\{(2, b), (4, c), (3, c), (2, a)\}$;
- г) $\{(1, a), (1, c), (2, a), (3, c), (4, d)\}$?

Визначити, до якого типу відображень вони належать.

29. Про кожну з наведених нижче функцій, чиї області визначення і значень збігаються із Z , сказати, чи є вони ін'єкціями, сюр'єкціями або бієкціями:

- а) $f(n) = 2n + 1$;
- б) $g(n) = \begin{cases} \frac{n}{2}, & \text{якщо } n - \text{парне,} \\ 2n, & \text{якщо } n - \text{непарне;} \end{cases}$
- в) $h(n) = \begin{cases} n + 1, & \text{якщо } n - \text{парне,} \\ n - 1, & \text{якщо } n - \text{непарне.} \end{cases}$

30. Зобразити графіки наведених функцій та визначити, які з них є ін'єкціями, сюр'єкціями або бієкціями:

- а) $f: Z \rightarrow Z, f(x) = x^2 + 1$;
- б) $g: N \rightarrow N, g(x) = 2^x$;
- в) $h: R \rightarrow R, h(x) = 5x - 1$;
- г) $j: R \rightarrow R, j(x) = \begin{cases} 2x - 3, & \text{якщо } x \geq 1, \\ x + 1, & \text{якщо } x < 1; \end{cases}$
- д) $k: R \rightarrow R, k(x) = x + |x|$;
- е) $l: R \rightarrow R, l(x) = 2x - |x|$.

31. Нехай $A = \{1, 3, 5, 6\}$, $B = \{1, 2, 3, 5, 7\}$ і $C = \{2, 4, 7\}$. Обчислити:

- а) $A \cup B$;
- б) $(A \cup C) \setminus B$;
- в) $A \cap B \cap C$;
- г) $(A \setminus C) \cup (B \setminus A)$;
- д) $A \triangle B$;
- е) $(B \setminus C) \cap (A \setminus B)$.

32. Чи існують множини A , B і C , для яких одночасно виконуються такі співвідношення:

$$A \neq \emptyset, A \cap B = \emptyset, A \cap C = \emptyset, A \setminus (B \cup C) = \emptyset$$

33. За допомогою діаграм Ейлера-Венна та методу логічних таблиць перевірити такі теоретико-множинні рівності:

- а) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$;
- б) $(A \cap B) \cup A = (A \cup B) \cap A = A$;
- в) $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$;
- г) $(A \cup B) \setminus C = (A \setminus C) \cup (B \setminus C)$;
- д) $A \cap (B \Delta C) = (A \cap B) \Delta (A \cap C)$;
- е) $A \Delta B = (A \cup B) \setminus (A \cap B)$.

34. Визначити множини:

- а) $\emptyset \cap \{\emptyset\}$;
- б) $\{\emptyset\} \cap \{\emptyset\}$;
- в) $\{\emptyset\} \cup \emptyset$;
- г) $\{\emptyset, \{\emptyset\}\} \setminus \emptyset$;
- д) $\{\emptyset, \{\emptyset\}\} \setminus \{\emptyset\}$;
- е) $\{\emptyset, \{\emptyset\}\} \setminus \{\{\emptyset\}\}$.

35. Нехай A і B – довільні множини. Довести, що співвідношення, розміщені в одному рядку, рівносильні між собою, тобто з істинності одного з них випливає істинність усіх інших.

- а) $A \subseteq B, B \subseteq A, A \cup B = B, A \cap B = A, A \setminus B = \emptyset$;
- б) $A \subseteq B, A \cup B = E, A \cap B = \emptyset$;
- в) $A \cap B = \emptyset, A \subseteq B, B \subseteq A$;
- г) $A \cup B = E, A \subseteq B, B \subseteq A$.

36. Перевірити (довести чи спростувати) правильність твердження:

- а) якщо $A \cup C = B \cup C$, то $A = B$;
- б) якщо $A \cap C = B \cap C$, то $A = B$;
- в) якщо $A \Delta B = A \Delta C$, то $B = C$;
- г) якщо $A \cup C = B \cup C$ і $A \cap C = B \cap C$, то $A = B$;
- д) якщо $A \setminus C = B \setminus C$ і $C \setminus A = C \setminus B$, то $A = B$;
- е) якщо $A \setminus C = B \setminus C$ і $A \cap C = B \cap C$, то $A = B$.

37. Довести, що:

- а) $A \cup B \subseteq C \Leftrightarrow A \subseteq C \text{ і } B \subseteq C$;
- б) $A \subseteq B \cap C \Leftrightarrow A \subseteq B \text{ і } A \subseteq C$;
- в) $(A \cap B) \cup C = A \cap (B \cup C) \Leftrightarrow C \subseteq A$;
- г) $A \subseteq B \cup C \Leftrightarrow A \cap B \subseteq C$;
- д) $(A \setminus B) \cup B = A \Leftrightarrow B \subseteq A$;
- е) $A \setminus B = A \Leftrightarrow A \cap B = \emptyset$.

38. Навести приклад множин A і B , які спростовують рівність $(A \cup B) \setminus B = A$. Сформулювати й довести необхідні й достатні умови для виконання цієї рівності.

39. Що можна сказати про множини A і B , якщо:

- а) $A \cup B = A \cap B$;
- б) $A \setminus B = B \setminus A$;
- в) $A \subseteq B$ і $A \subseteq B$;
- г) $A \cup B = \emptyset$;
- д) $A \setminus B = A$;
- е) $A \setminus B = B$;
- ж) $A \setminus B = \emptyset$;
- з) $(A \setminus B) \cup (B \setminus A) = A \cup B$?

40. Довести тотожність $(A \cap B) \cup A = (A \cup B) \cap A = A$.

41. Сформулювати та довести необхідні й достатні умови для множин A і B , щоб виконувалася рівність $\beta(A) \cup \beta(B) = \beta(A \cup B)$.

42. Що можна сказати про множини A і B , якщо:

- а) $A \triangle B = B$;
- б) $A \triangle B = B$;
- в) $A \triangle B = \emptyset$;
- г) $A \triangle B = E$;
- д) $(A \setminus B) \triangle (B \setminus A) = \emptyset$;
- е) $(A \cup B) \triangle A = B$?

43. Для заданих множин $A = \{1, 2\}$ і $B = \{2, 3, 4\}$ визначити:

- а) $A \times B$;
- б) $B \times A$;
- в) $B \times B$;
- г) $(B \setminus A) \times A$;
- д) $A \times B \times A$;
- е) $A \times (A \cup B)$.

44. Довести, що $(A \times B) \cup (B \times A) \subseteq (A \cup B) \times (A \cup B)$.

45. Сформулювати й довести необхідні та достатні умови виконання рівності $(A \cup B) \times (A \cup B) = (A \times B) \cup (B \times A)$.

46. Позначимо $D = \beta(M) \times \beta(M) \times \beta(M)$, де $M = \{1, 2\}$. Виписати всі такі кортежі $(A, B, C) \in D$, що:

- а) $A \cap B = C$;
- б) $A \cup B \cup C = M$;
- в) $A \cap B \neq B \cap C$.

47. На множині $M = \{1, 2, 3, 4\}$ задано відношення:

$R_1 = \{(1, 1), (1, 3), (2, 2), (2, 4), (3, 1), (3, 3), (4, 2), (4, 4)\};$

$R_2 = \{(1, 1), (2, 2), (2, 3), (2, 4), (3, 1), (3, 3), (4, 1), (4, 4)\};$

$R_3 = \{(1, 3), (1, 4), (2, 1), (1, 2), (3, 1), (3, 3), (4, 1)\};$

$R_4 = \{(1, 1), (1, 2), (1, 4), (2, 2), (2, 4), (3, 3), (3, 4), (4, 2), (4, 3), (4, 4)\};$

$R_5 = \{(1, 2), (1, 3), (2, 3), (2, 4), (4, 1), (4, 3)\}.$

Визначити, які з цих відношень:

- а) рефлексивні;
- б) антирефлексивні;
- в) симетричні;
- г) антисиметричні;
- д) транзитивні.

48. Довести, що перетин транзитивних відношень є транзитивним відношенням.

49. Довести, що будь-яка непорожня скінченна частково впорядкована множина A містить мінімальний і максимальний елементи.

50. Довести, що скінченна частково впорядкована множина має найменший (найбільший) елемент тоді й тільки тоді, коли вона містить рівно один мінімальний (максимальний) елемент. Чи це так для нескінченних частково впорядкованих множин?

1.7. Програмна реалізація вправ

1.7.1. Завдання для програмної реалізації

Завдання 1. Без використання вбудованих функцій Python реалізувати основні операції над множинами.

Завдання 2. Для довільної множини написати програму, яка обчислює булеан. Наприклад, для множини $\{4, 5, 7\}$ булеан дорівнює $\{\}, \{4\}, \{5\}, \{7\}, \{4, 5\}, \{4, 7\}, \{5, 7\}, \{4, 5, 7\}$. Усього множин $2^3 = 8$.

1.7.2. Основи Python для операцій з множинами

Щоб реалізувати операції над множинами мовою Python без використання готових інструментів, необхідна мінімальна база знань програмування. Такі знання дають змогу не просто використовувати готову функцію, а зрозуміти внутрішню логіку побудови алгоритму.

Множини у завданнях подаються у вигляді списків, які дозволяють власноруч реалізовувати перевірку належності елементів, об'єднання, перетин, різницю та симетричну різницю. Цей підхід сприяє глибшому розумінню, як працюють цикли, формуються умови та зберігаються дані.

Зрештою, такий підхід перетворює абстрактні теоретичні операції на конкретний програмний код, роблячи процес навчання максимально прикладним і зрозумілим.

Списки в Python

Список (*list*) – це базова структура даних мови програмування Python, що дозволяє зберігати впорядковану колекцію елементів. Кожний елемент списку має свій індекс, починаючи з нуля, що дає змогу отримати доступ до будь-якого елемента за його порядковим номером. Списки належать до змінюваних (*mutable*) типів даних: після створення список можна змінювати – додавати, вилучати або редагувати його елементи.

Список можна *створити* за допомогою квадратних дужок [], у яких перелічуються елементи через кому:

```
numbers = [1, 2, 3, 4, 5]          # список цілих чисел
words = ["кіт", "пес", "риба"]    # список рядків
mixed = [1, "текст", 3.14, True]   # список із різнотипними даними
empty = []                        # порожній список
```

Елементи списку нумеруються починаючи з 0. Для отримання *доступу* до певного елемента використовується його індекс у квадратних дужках:

```
numbers = [10, 20, 30, 40]
first = numbers[0]    # перший елемент - 10
third = numbers[2]    # третій елемент - 30
```

У Python підтримуються також *від'ємні індекси*, які відраховуються від кінця списку:

```
last = numbers[-1]    # останній елемент - 40
```

Оскільки список є змінюваним типом даних, значення елемента можна *змінити*, звернувшись до нього за індексом:

```
numbers = [10, 20, 30]
numbers[1] = 200      # тепер список виглядає як [10, 200, 30]
```

Щоб дізнатися, скільки елементів містить список, використовують вбудовану функцію *len()*. Вона повертає ціле число – кількість об'єктів у послідовності:

```
numbers = [10, 20, 30]
len(numbers)    # довжина списку 3
```

У Python існують два основні способи додати новий елемент у список: у кінець або у конкретну позицію. Ці методи дають змогу динамічно формувати списки залежно від задачі.

Метод *append(x)* додає новий елемент *x* у *кінець* списку, розширюючи його на один елемент.

Метод *insert(i, x)* дає змогу вставити елемент *x* у *задану позицію*, яка визначається індексом *i*. Після вставки всі наступні елементи зміщуються праворуч:

```
numbers.append(50)    # [10, 200, 30, 50]
numbers.insert(1, 15) # [10, 15, 200, 30, 50]
```

Можна легко *об'єднувати* два (або більше) списки за допомогою оператора «+», який створює *новий* список, що містить елементи всіх з'єднаних списків у заданому порядку:

```
a = [1, 2]
b = [3, 4]
c = a + b      # [1, 2, 3, 4]
```

Існує кілька способів *видалити* елементи зі списку. Кожен із них має свою специфіку, що корисно знати під час роботи з даними. Метод *remove(x)* видаляє *перше* входження елемента *x* зі списку. Якщо такого елемента немає, то виникає помилка:

```
numbers = [10, 20, 10, 30]
numbers.remove(10)      # [20, 10, 30]
```

Метод *pop(i)* видаляє елемент за *вказаним* індексом *i* та повертає його. Якщо індекс не вказано, видаляється *останній* елемент:

```
numbers = [10, 20, 30]
deleted = numbers.pop(2) # [10, 20]
```

Оператор *del* використовується для видалення елемента за індексом або діапазону (зрізу). На відміну від методів, *del* не повертає значення, але дозволяє гнучко працювати зі зрізами (наприклад, одразу видалити діапазон елементів):

```
numbers = [10, 20, 30, 40, 50]
# Видаляє другий елемент (20) [10, 30, 40, 50]
del numbers[1]
# Видаляє елементи з індексами 1 і 2 (30, 40) [10, 50]
del numbers[1:3]
```

Усі методи змінюють список *без створення нового*.

У Python для взаємодії користувача з програмою застосовуються дві базові функції:

- *print()* для виведення інформації на екран;
- *input()* для отримання даних, які вводить користувач з клавіатури.

Функція *print()* призначена для виведення тексту або значень змінних на екран, вона має такий синтаксис:

```
print(value1, value2, ..., sep=' ', end='\n') ,
```

де *value1, value2, ...* – один або декілька об'єктів, що підлягають виведенню;

sep – рядок, який розділяє елементи (за замовчуванням це пробіл);

end – рядок, який виводиться після усіх значень (за замовчуванням це символ нового рядка *\n*).

```
print("Привіт, світ!")          # простий вивід
print("Сума:", 5 + 3)           # вивід тексту та обчислення
print("A", "B", "C", sep="-")  # A-B-C
print("Кінець", end="!")       # виведе без переходу на новий рядок
```

Результат виведення тексту або значень змінних на екран у Python:

```
Привіт, світ!  
Сума: 8  
A-B-C  
Кінець!
```

Функція `input()` використовується для введення даних користувачем з клавіатури, вона має такий синтаксис:

```
variable = input([prompt])
```

`prompt` – необов'язковий рядок-підказка, який відображається користувачу перед введенням;

`variable` – змінна, у яку зберігається введене значення.

Введені дані завжди повертаються як рядок (тип `str`).

```
name = input("Введіть ваше ім'я: ")  
print("Привіт,", name)
```

Результат введення даних користувачем з клавіатури в Python:

```
Введіть ваше ім'я: Оксана  
Привіт, Оксана
```

Якщо потрібно працювати з числами, введене значення слід перетворити з рядка:

```
# перетворення на ціле число  
age = int(input("Введіть ваш вік: "))  
# перетворення на дійсне число  
height = float(input("Введіть зріст: "))
```

Результат перетворення типів у Python:

```
Введіть ваш вік: 20  
Введіть зріст: 170.5
```

Python дозволяє формувати текст за допомогою спеціальних конструкцій:

```
name = "Іван"  
age = 20  
# конкатенація рядків  
print("Привіт, " + name)  
# метод format()  
print("Мене звати {}. Мені {} років.".format(name, age))  
# F-рядки (f-strings)  
print(f"Мене звати {name}. Мені {age} років.")
```

Результат форматування тексту в Python:

```
Привіт, Іван  
Мене звати Іван. Мені 20 років.  
Мене звати Іван. Мені 20 років.
```

Ці методи дозволяють динамічно вставляти значення змінних у текстові рядки, що робить вивід даних гнучким та читабельним. Якщо

конкатенація просто «склеює» частини тексту (що незручно при великій кількості даних), то методи *format()* та *f-strings* використовують спеціальні заповнювачі `{}` як шаблони. Зокрема, *f-strings* забезпечують найвищу швидкість виконання та дозволяють писати код максимально лаконічно, підставляючи змінні безпосередньо всередину рядка.

Для створення незалежної копії списку використовують метод `.copy()`. Будь-які зміни в копії не впливають на оригінальний список, що зручно для збереження початкових даних:

```
A = [1, 2, 3]
# створюється новий список result
result = A.copy()
# змінюємо копію, додаючи елемент
result.append(4)
print("Оригінал A:", A)          # [1, 2, 3]
print("Копія result:", result)  # [1, 2, 3, 4]
```

Результат створення незалежної копії списку в Python:

```
Оригінал A: [1, 2, 3]
Копія result: [1, 2, 3, 4]
```

Перебирання елементів будь-якої ітерованої колекції (списку, множини, рядка тощо) можна виконати за допомогою циклу *for*, загальний синтаксис якого в Python такий:

```
for змінна in колекція:
    # дії з елементом
```

де

`змінна` – це тимчасова змінна, яка на кожній ітерації набуває значення чергового елемента з колекції;

`колекція` – будь-який об'єкт, що підтримує ітерацію: *list*, *set*, *tuple*, *str*, *dict*, *range*.

Відступ є обов'язковим, код всередині циклу має бути на одному рівні:

```
A = [1, 2, 3]
for item in A:
    print(item, end=" ")
```

Результат створення відступів у середині циклу в Python:

```
1 2 3
```

Умовний оператор *if* перевіряє умову та виконує код, якщо умова істинна, має такий синтаксис:

```
if умова:
    # код, якщо умова істинна
else:
    # код, якщо умова хибна
```

Для перевірки, чи заданий елемент міститься у колекції, наприклад у списку, множині, рядку або словнику, в Python використовується

оператор *in*. Це логічна операція, яка повертає *True*, якщо елемент присутній, і *False*, якщо його немає:

```
# список з цілих чисел
A = [1, 2, 3, 4]
# True – число 2 присутнє у списку
print("2 є в списку A:", 2 in A)
# False – числа 5 немає у списку
print("5 є в списку A:", 5 in A)
```

Результат використання оператора *in* у Python:

```
2 є в списку A: True
5 є в списку A: False
```

Використання структури *set*

Альтернативно для уникнення дублікатів можна використовувати вбудовану структуру Python *set*, яка автоматично видаляє дублікати та підтримує операції над множинами.

Множина (*set*) – це структура даних у Python, яка автоматично видаляє дублікати та підтримує математичні операції над множинами, такі як об'єднання, перетин, різниця та симетрична різниця.

Множину можна створити за допомогою фігурних дужок *{}* або функції *set()*:

```
A = {1, 2, 3, 4} # Множина A
B = set([3, 4, 5, 6]) # Множина B, створена зі списку
print("Множина A:", A)
print("Множина B:", B)
```

Результат створення множини в Python:

```
Множина A: {1, 2, 3, 4}
Множина B: {3, 4, 5, 6}
```

Якщо у множині є дублікати, вони автоматично видаляються:

```
A = {1, 2, 2, 3} # Створює множину {1, 2, 3} без дублікатів
print("Множина A:", A)
```

Результат отримання множини без дублікатів у Python:

```
Множина A: {1, 2, 3}
```

Метод *add(x)* додає елемент *x* до множини. Якщо такий елемент уже присутній у множині, він не буде доданий вдруге – множина залишиться без змін:

```
A = {1, 2, 3, 4}
print("Множина A:", A)
A.add(5) # Додається новий елемент
# Виведе: {1, 2, 3, 4, 5}
print("Множина A після додавання елементу:", A)
A.add(3) # Елемент 3 уже є, нічого не зміниться
# Виведе: {1, 2, 3, 4, 5}
print("Множина A після додавання дублікату:", A)
```

Результат додавання елементів до множини в Python:

```
Множина A: {1, 2, 3, 4}
Множина A після додавання елементу: {1, 2, 3, 4, 5}
Множина A після додавання дублікату: {1, 2, 3, 4, 5}
```

Python надає набір вбудованих операторів, що реалізують стандартні операції над множинами:

- «|» – об'єднання; «&» – перетин;
- «-» – різниця; «^» – симетрична різниця;

Для перевірки належності заданого елемента множині використовують оператор *in*, який працює з множинами швидше, ніж зі списками.

Приклад використання операцій над множинами і перевірки належності на основі символів:

```
A = {"a", "b", "c"}
B = {"b", "c", "d"}
print("Об'єднання A | B:", A | B)          # {"a", "b", "c", "d"}
print("Перетин A & B:", A & B)              # {"b", "c"}
print("Різниця A - B:", A - B)              # {"a"}
print("Симетрична різниця A ^ B:", A ^ B)  # {"a", "d"}
# Перевірка належності
print("Чи належить 'b' множині A?", "b" in A) # True
```

Результат виконання операцій над множинами в Python:

```
Об'єднання A | B: {'b', 'c', 'd', 'a'}
Перетин A & B: {'b', 'c'}
Різниця A - B: {'a'}
Симетрична різниця A ^ B: {'a', 'd'}
Чи належить 'b' множині A? True
```

1.7.3. Виконання завдань для програмної реалізації

Завдання 1. Без використання вбудованих функцій Python реалізувати основні операції над множинами.

Якщо множини подано списками, операції над множинами можна реалізувати таким чином:

– **перевірка належності:** перебираємо елементи списку циклом *for* і порівнюємо з шуканим елементом за допомогою *if*.

```
# Перевірка належності елемента множині
# Множина A
A = [1, 2, 3, 4]
# Елемент для перевірки
element = 2
found = False
for item in A:
    if item == element:
        found = True
        break
print("Множина A:", A)
print("Елемент", element, "належить A:", found)
```

Результат перевірки належності елемента множині (базовий приклад):

```
Множина A: [1, 2, 3, 4]
Елемент 2 належить A: True
```

Перевірка наявності певної бібліотеки Python у списку залежностей проєкту (нетривіальний приклад):

```
# Перевірка належності елемента множині
# Множина
libraries = ["numpy", "pandas", "matplotlib", "requests",
"tensorflow"]
# Елемент для перевірки
library = "pandas"
found = False
for item in libraries:
    if item == library:
        found = True
        break
print("Бібліотеки в проєкті:", libraries)
print("Бібліотека", library, "є в списку:", found)
```

Результат виконання коду:

```
Бібліотеки в проєкті: ['numpy', 'pandas', 'matplotlib', 'requests',
'tensorflow']
Бібліотека pandas є в списку: True
```

– **об'єднання**: копіюємо один список і додаємо до нього елементи іншого списку, якщо їх там ще немає (перевірка за допомогою *in*).

```
# Об'єднання двох множин
# Множина A
A = [1, 2, 3, 4]
# Множина B
B = [3, 4, 5, 6]
# Копіюємо A
result = A.copy()
for item in B:
    if item not in result:
        result.append(item)
print("Множина A:", A)
print("Множина B:", B)
print("Об'єднання A U B:", result)
```

Результат об'єднання двох множин (базовий приклад):

```
Множина A: [1, 2, 3, 4]
Множина B: [3, 4, 5, 6]
Об'єднання A U B: [1, 2, 3, 4, 5, 6]
```

Об'єднання бібліотек двох Python-проєктів (нетривіальний приклад):

```
# Об'єднання двох множин
# Бібліотеки проєкту 1 (множина 1)
project1 = ["numpy", "pandas", "matplotlib", "pandas"]
# Бібліотеки проєкту 2 (множина 2)
```

```

project2 = ["requests", "matplotlib", "tensorflow", "flask"]
# Копіюємо Бібліотеки проекту 1 (множина 1)
result = project1.copy()
for item in project2:
    if item not in result:
        result.append(item)
print("Бібліотеки проекту 1:", project1)
print("Бібліотеки проекту 2:", project2)
print("Усі бібліотеки A U B:", result)

```

Результат виконання коду:

```

Бібліотеки проекту 1: ['numpy', 'pandas', 'matplotlib', 'pandas']
Бібліотеки проекту 2: ['requests', 'matplotlib', 'tensorflow',
'flask']
Усі бібліотеки A U B: ['numpy', 'pandas', 'matplotlib', 'pandas',
'requests', 'tensorflow', 'flask']

```

– **перетин**: створюємо новий список, додаючи лише елементи, які є в обох списках (перевірка за допомогою in).

```

# Перетин двох множин
# Множина A
A = [1, 2, 3, 4]
# Множина B
B = [3, 4, 5, 6]
result = []
for item in A:
    if item in B:
        result.append(item)
print("Множина A:", A)
print("Множина B:", B)
print("Перетин A ∩ B:", result)

```

Результат отримання перетину двох множин (базовий приклад):

```

Множина A: [1, 2, 3, 4]
Множина B: [3, 4, 5, 6]
Перетин A ∩ B: [3, 4]

```

Знаходження спільних модулів двох Python-скриптів (нетривіальний приклад):

```

# Перетин двох множин
# Модулі Python-скрипта 1
script1 = ["os", "sys", "math", "random", "datetime"]
# Модулі Python-скрипта 2
script2 = ["math", "datetime", "json", "time"]
result = []

for item in script1:
    if item in script2:
        result.append(item)
print("Модулі скрипта 1:", script1)
print("Модулі скрипта 2:", script2)
print("Спільні модулі A ∩ B:", result)

```

Результат виконання коду:

```
Модулі скрипта 1: ['os', 'sys', 'math', 'random', 'datetime']
Модулі скрипта 2: ['math', 'datetime', 'json', 'time']
Спільні модулі  $A \cap B$ : ['math', 'datetime']
```

– **різниця**: створюємо новий список, додаючи елементи першого списку, яких немає в другому (перевірка за допомогою *in*).

```
# Різниця множин (A - B)
# Множина A
A = [1, 2, 3, 4]
# Множина B
B = [3, 4, 5, 6]
result = []
for item in A:
    if item not in B:
        result.append(item)
print("Множина A:", A)
print("Множина B:", B)
print("Різниця A \ B:", result)
```

Результат отримання різниці множин A – B (базовий приклад):

```
Множина A: [1, 2, 3, 4]
Множина B: [3, 4, 5, 6]
Різниця A \ B: [1, 2]
```

Імітування порівняння можливостей Python-модулів у програмуванні (нетривіальний приклад):

```
# Різниця множин
# Функції Python-модуля 1 (множина 1)
module1 = ["append", "pop", "sort", "reverse", "count"]
# Функції Python-модуля 2 (множина 2)
module2 = ["sort", "count", "index", "clear"]
result = []
for item in module1:
    if item not in module2:
        result.append(item)
print("Функції модуля 1:", module1)
print("Функції модуля 2:", module2)
print("Унікальні функції A \ B:", result)
```

Результат виконання коду:

```
Функції модуля 1: ['append', 'pop', 'sort', 'reverse', 'count']
Функції модуля 2: ['sort', 'count', 'index', 'clear']
Унікальні функції A \ B: ['append', 'pop', 'reverse']
```

– **симетрична різниця**: використовуємо об'єднання та перетин, щоб знайти елементи, які є в одному зі списків, але не в обох.

```
# Симетрична різниця множин
A = [1, 2, 3, 4] # Множина A
B = [3, 4, 5, 6] # Множина B
# Об'єднання (A U B)
```

```

union = A.copy()
for item in B:
    if item not in union:
        union.append(item)
# Перетин ( $A \cap B$ )
intersection = []
for item in A:
    if item in B:
        intersection.append(item)
# Симетрична різниця:  $(A \cup B) \setminus (A \cap B)$ 
result = []
for item in union:
    if item not in intersection:
        result.append(item)
print("Множина A:", A)
print("Множина B:", B)
print("Симетрична різниця  $A \Delta B$ :", result)

```

Результат отримання симетричної різниці множин (базовий приклад):

```

Множина A: [1, 2, 3, 4]
Множина B: [3, 4, 5, 6]
Симетрична різниця  $A \Delta B$ : [1, 2, 5, 6]

```

Знаходження унікальних бібліотек, які є лише в одному з двох проєктів (нетривіальний приклад):

```

# Симетрична різниця множин
project1 = ["numpy", "pandas", "matplotlib", "requests", "numpy"]
project2 = ["matplotlib", "tensorflow", "flask", "requests"]
union = project1.copy() # Об'єднання  $(A \cup B)$ 
for item in project2:
    if item not in union:
        union.append(item)
intersection = [] # Перетин  $(A \cap B)$ 
for item in project1:
    if item in project2:
        intersection.append(item)
# Симетрична різниця:  $(A \cup B) \setminus (A \cap B)$ 
result = []
for item in union:
    if item not in intersection:
        result.append(item)
print("Бібліотеки проєкту 1:", project1)
print("Бібліотеки проєкту 2:", project2)
print("Унікальні бібліотеки  $A \Delta B$ :", result)

```

Результат виконання коду:

```

Бібліотеки проєкту 1: ['numpy', 'pandas', 'matplotlib', 'requests',
'numpy']
Бібліотеки проєкту 2: ['matplotlib', 'tensorflow', 'flask',
'requests']
Унікальні бібліотеки  $A \Delta B$ : ['numpy', 'pandas', 'numpy',
'tensorflow', 'flask']

```

У математиці множини не містять дублікатів. У наведених програмах код спрощено, що дозволяє спискам мати дублікати. Для видалення дублікатів можна перебирати елементи та додавати їх до нового списку, якщо вони ще не додані:

```
# Видалення дублікатів
A = [1, 2, 2, 3, 3, 4] # Список з дублікатами
result = []           # Новий список без дублікатів
for item in A:
    if item not in result:
        result.append(item)
print("Множина без дублікатів:", result) # Виведе [1, 2, 3, 4]
```

Результат видалення дублікатів:

```
Множина без дублікатів: [1, 2, 3, 4]
```

Програмна реалізація розглянутих базових прикладів із використанням структури `set`:

– **перевірка** належності: за допомогою оператора `in`, який миттєво визначає, чи міститься елемент у множині, без перебирання всіх значень.

```
# Перевірка належності елемента множині
# Множина A
A = {1, 2, 3, 4}
# Елемент для перевірки
element = 2
print("Множина A:", A)
print("Елемент", element, "належить A:", element in A)
```

Результат перевірки належності елемента множині з використанням `set`:

```
Множина A: {1, 2, 3, 4}
Елемент 2 належить A: True
```

– **об'єднання**: оператор `|` створює нову множину, яка містить усі унікальні елементи з обох множин A і B , без повторень.

```
# Об'єднання двох множин
A = {1, 2, 3, 4} # Множина A
B = {3, 4, 5, 6} # Множина B
# Оператор об'єднання
result = A | B
print("Множина A:", A)
print("Множина B:", B)
print("Об'єднання A U B:", result)
```

Результат об'єднання двох множин із використанням `set`:

```
Множина A: {1, 2, 3, 4}
Множина B: {3, 4, 5, 6}
Об'єднання A U B: {1, 2, 3, 4, 5, 6}
```

– **перетин**: оператор `&` повертає нову множину, яка містить лише ті елементи, що одночасно присутні у множинах A і B .

```
# Перетин двох множин
A = {1, 2, 3, 4}      # Множина A
B = {3, 4, 5, 6}      # Множина B
result = A & B         # Оператор перетину
print("Множина A:", A)
print("Множина B:", B)
print("Перетин A ∩ B:", result)
```

Результат перетину двох множин із використанням `set`:

```
Множина A: {1, 2, 3, 4}
Множина B: {3, 4, 5, 6}
Перетин A ∩ B: {3, 4}
```

– **різниця**: оператор «-» створює нову множину, яка містить лише ті елементи, що є в A , але відсутні в B – тобто унікальні для множини A .

```
# Різниця двох множин
A = {1, 2, 3, 4}      # Множина A
B = {3, 4, 5, 6}      # Множина B
result = A - B         # Оператор різниці
print("Множина A:", A)
print("Множина B:", B)
print("Різниця A \ B:", result)
```

Результат різниці двох множин із використанням `set`:

```
Множина A: {1, 2, 3, 4}
Множина B: {3, 4, 5, 6}
Різниця A \ B: {1, 2}
```

– **симетрична різниця**: оператор «^» повертає множину з елементів, які містяться лише в одній із множин, тобто вилучає спільні елементи, залишаючи лише унікальні для A або B .

```
# Симетрична різниця множин
A = {1, 2, 3, 4}      # Множина A
B = {3, 4, 5, 6}      # Множина B
result = A ^ B         # Оператор симетричної різниці
print("Множина A:", A)
print("Множина B:", B)
print("Симетрична різниця A Δ B:", result)
```

Результат перевірки симетричної різниці двох множин із використанням `set`:

```
Множина A: {1, 2, 3, 4}
Множина B: {3, 4, 5, 6}
Симетрична різниця A Δ B: {1, 2, 5, 6}
```

Використання `set` значно спрощує код, оскільки ця структура автоматично видаляє дублікати та має вбудовані оператори («|», «&», «-», «^») для роботи з множинами. Однак для навчальних цілей корисно спочатку реалізувати операції вручну за допомогою списків, щоб зрозуміти їхню логіку.

Завдання 2. Для довільної множини написати програму, яка обчислює булеан.

Наприклад, для множини $\{4,5,7\}$ булеан дорівнює $\{\}, \{4\}, \{5\}, \{7\}, \{4,5\}, \{4,7\}, \{5,7\}, \{4,5,7\}$. Усього множин $2^3=8$.

Програма обчислює булеан (множину всіх підмножин) для довільної множини, заданої користувачем через введення з клавіатури (наприклад, «4 5 7»). Вона використовує двійкову систему числення для генерації всіх можливих підмножин.

Для множини з (n) елементів булеан містить (2^n) підмножин, які відповідають числам від 0 до $(2^n - 1)$ у двійковій системі. Кожний біт у двійковому числі вказує, чи додавати відповідний елемент множини до підмножини (1 – додати, 0 – вилючити).

Створена найпростіша програма використовує базові конструкції Python (списки, цикли, введення через `input()`) без функцій.

Опис змінних, які використовуються у програмі:

- `input_string`: рядок, який ввів користувач (наприклад, «4 5 7»);
- `input_set`: список елементів множини, отриманий за допомогою `split()`;
- `n`: кількість елементів, отримана через `len(input_set)`;
- `boolean`: список для зберігання всіх підмножин;
- `subset`: поточна підмножина, що генерується для кожного (i) ;
- `binary`: двійкове представлення числа (i) , отримане через `bin(i)[2:]` (видаляємо префікс «0b») з додаванням нулів зліва.

Користувач вводить елементи множини через пробіл, які метод рядка `split()` перетворює на список `input_set`. Цикл `for i in range(2 ** n)` генерує числа від 0 до $(2^n - 1)$.

Для кожного (i) обчислюється двійкове представлення через `bin(i)[2:]`, а нулі додаються вручну, щоб довжина була (n) . Потім перевіряється кожний біт у `binary`: якщо біт дорівнює «1», додаємо відповідний елемент із `input_set` до `subset`. Після формування поточної підмножини `subset` додаємо його до списку `boolean`.

Програма виводить введену множину і всі її підмножини у вигляді списків:

```
# Обчислення булеана для заданої множини
# Введення множини користувачем
input_string = input("Введіть елементи множини через пробіл: ")
# Розбиваємо введення на список
input_set = input_string.split()

# Кількість елементів у множині
n = len(input_set)
# Список для зберігання всіх підмножин
boolean = []
```



```

# Цикл від 0 до 2^n - 1
for i in range(2 ** n):
    # Поточна підмножина
    subset = []
    # Перетворення i у двійкове представлення
    binary = bin(i)[2:] # Видаляємо префікс "0b"
    # Додаємо нулі зліва, щоб довжина була n
    while len(binary) < n:
        binary = "0" + binary
    # Перевірка кожної позиції у двійковому числі

    for j in range(n):
        # Якщо на позиції 1, додаємо елемент
        if binary[j] == '1':
            subset.append(input_set[j])
    boolean.append(subset)

# Виведення булеана
print("Булеан множини", input_set, ":")
for subset in boolean:
    print(subset)

```

Результат роботи програми обчислення булеана для заданої множини (приклад 1):

```

Введіть елементи множини через пробіл: 4 7 9
Булеан множини ['4', '7', '9'] :
[]
['9']
['7']
['7', '9']
['4']
['4', '9']
['4', '7']
['4', '7', '9']

```

Результат роботи програми обчислення булеана для заданої множини (приклад 2):

```

Введіть елементи множини через пробіл: 4 7 9
Булеан множини ['4', '7', '9'] :
[]
['9']
['7']
['7', '9']
['4']
['4', '9']
['4', '7']
['4', '7', '9']

```

Ця програма є простою і зрозумілою; використовуючи базові конструкції Python, допомагає зрозуміти концепцію булеана множини та її зв'язок із двійковою системою числення.

2. АЛГЕБРА ВИСЛОВЛЮВАНЬ

2.1. Висловлювання й алгебра висловлювань. Тавтології і протиріччя

Висловлюванням називають розповідне речення, про яке можна сказати, що воно істинне або хибне.

Наприклад:

«Місяць – природний супутник Землі» є істинним висловлюванням;

«5 помножити на 5 дорівнює 25» є істинним висловлюванням;

«Дніпро – ріка в Африці» є хибним висловлюванням.

Речення «Яка зараз година?» або «Вивчіть цей розділ до наступного заняття!» не є висловлюваннями, оскільки перше є запитанням, а друге – наказом.

Алгебра висловлювань (логіка висловлювань) – це розділ математичної логіки, що вивчає перетворення і комбінації висловлювань, у яких істинність або хибність нових висловлювань визначається істинністю чи хибністю первинних припущень, названих аксіомами.

Домовимося позначати: якщо деяке висловлювання p – істинне, то $(p) = 1$ (або T від англ. *True*), якщо деяке висловлювання p – хибне, то $(p) = 0$ (або F від англ. *False*).

Це називається **істиннісним значенням висловлювання**.

Для позначення висловлювань використовують латинські літери: p, g, r, s, t, A, B, C тощо. Їх називають **пропозиційними змінними**.

Із простих висловлювань за допомогою логічних зв'язок («і», «або», «не», «якщо ..., то», «тільки якщо», «тоді») утворюють складні висловлювання.

Приклади

Нехай p – висловлювання «Сьогодні світить сонце», q – «Настрій Івана – гарний».

– **Імплікація**: «Якщо сьогодні світить сонце, то в Івана гарний настрій».

– **Еквіваленція**: «Сьогодні світить сонце тоді й тільки тоді, коли в Івана гарний настрій».

Висловлювання p і q (зв'язка «і» позначається символом « $\wedge$ ») має вигляд $p \wedge q$ та є складним висловлюванням: «Сьогодні світить сонце і настрій в Івана гарний». Таке висловлювання називається **кон'юнкцією** висловлювань p і q , або логічним «І».

Висловлювання « p або q » (зазвичай зв'язка «або» позначається символом « $\vee$ ») має вигляд $p \vee q$ та є складним висловлюванням: «Сьогодні або світить сонце, або настрій в Івана гарний». Таке висловлювання називається **диз'юнкцією** висловлювань p і q , або логічним «АБО».

Висловлювання «не p » (позначається символом « $\neg$ ») має вигляд $\neg p$ та є висловлюванням: «Сьогодні сонце не світить». Таке висловлювання називається **запереченням** висловлювання p , або логічним «**НЕ**».

Висловлювання «якщо p , то q » (зазвичай зв'язка «якщо ..., то» позначається символом « $\rightarrow$ ») має вигляд $p \rightarrow q$ та є складним висловлюванням: «Якщо сьогодні світить сонце, то в Івана гарний настрій». Таке висловлювання називається **імплікацією** висловлювань p і q , або логічним «**НАСЛІДКОМ**».

Висловлювання « p еквівалентне q » (зазвичай зв'язка «еквівалентність» позначається символом « $\leftrightarrow$ ») має вигляд $p \leftrightarrow q$ та є складним висловлюванням: «Сьогодні світить сонце, і це еквівалентно тому, що в Івана гарний настрій». Таке висловлювання називається **еквіваленцією** висловлювань p і q , або логічним «**ЕКВІВАЛЕНТНЕ**».

Для визначення основних логічних операцій використовують спеціальні таблиці, що прийнято називати **таблицями істинності** (табл. 2.1).

Таблиця 2.1

Істиннісні значення основних логічних операцій

p	q	$\neg$	$\neg q$	$p \vee q$	$p \wedge q$	$p \rightarrow q$	$p \leftrightarrow q$
0	0	1	1	0	0	1	1
0	1	1	0	1	0	1	0
1	0	0	1	1	0	0	0
1	1	0	0	1	1	1	1

У підручниках різних математичних шкіл позначення логічних операцій може відрізнятися. Різні позначення логічних операцій наведені в табл. 2.2.

Таблиця 2. 2

Таблиця позначень логічних операцій

Операція	Позначення
Заперечення (<i>NOT</i>)	$\neg A$, $\sim A$, $!A$, $\bar{A}$ – надрядкова риска ($\neg A$), <i>NOT</i>
Кон'юнкція (<i>AND</i>)	$A \wedge B$, $A \& B$, $A \cdot B$, $A \cap B$, $A \text{ AND } B$, <i>AND</i>
Диз'юнкція (<i>OR</i>)	$A \vee B$, $A + B$, $A \cup B$, $A \text{ OR } B$, <i>OR</i>
Виключне АБО (<i>XOR</i>)	$A \oplus B$, $A \text{ XOR } B$, $A \veebar B$, <i>XOR</i>
Імплікація	$A \supset B$, $A \rightarrow B$, $A \Rightarrow B$, $A \Rightarrow B$, $\supset$
Еквіваленція (<i>XNOR</i>)	$A \leftrightarrow B$, $A \equiv B$, $A \Leftrightarrow B$, <i>XNOR</i>
Штрих Шеффера (<i>NAND</i>)	$A \mid B$, $A \uparrow B$, <i>NAND</i>
Стрілка Пірса (<i>NOR</i>)	$A \downarrow B$, <i>NOR</i>

Дамо математичну інтерпретацію складних висловлювань, утворених за допомогою логічних операцій (табл. 2.1)

Уведемо алфавіт $A = A_1 \cup A_2 \cup A_3$, який утворено такими множинами:

$A_1 = \{p, q, r, s, \dots, A, B, \dots\}$ – літери латинського алфавіту, які використовують для позначення пропозиційних змінних (ПЗ);

$A_2 = \{\neg, \wedge, \vee, \supset, \equiv\}$ – це множина логічних зв'язувань, тобто операцій, визначених для висловлювань;

$A_3 = \{(\, , \,)\}$ – множина допоміжних символів – дужок.

Символи алфавіту $A = A_1 \cup A_2 \cup A_3$ дозволяють формувати деякі послідовності, названі **пропозиційними формулами (ПФ)**. Введемо індуктивне визначення ПФ.

Дамо індуктивне визначення пропозиційної формули (або форми). Пропозиційні змінні $p, q, r, s, A, B, C \dots$ – ПФ, якщо A та B – ПФ, тобто:

$\neg A, A \vee B, A \wedge B, A \supset B, A \equiv B$ – теж ПФ.

Дужки в ПФ виконують ту ж роль, що й у звичайній алгебрі, тобто дозволяють змінити порядок логічних операцій, встановлених за таким пріоритетом:

1 2 3 4 5
 $\neg$ $\wedge$ $\vee$ $\supset$ $\equiv$

Приклад 2.1

Нехай задано ПФ $\neg p \vee q \supset r$. Таблиця істинності для ПФ має 8 рядків, відповідно, з усіма можливими наборами значень її трьох змінних $2^3 = 8$. Згідно з пріоритетом логічних операцій побудуємо табл. 2.3.

Таблиця 2.3

Результати виконання логічних операцій за пріоритетом

p	q	r	$\neg p$	$\neg p \vee q$	$\neg p \vee q \supset r$
0	0	0	1	1	0
0	0	1	1	1	1
0	1	0	1	1	0
0	1	1	1	1	1
1	0	0	0	0	1
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	0	1	0

Наведений вище метод таблиць істинності дозволяє однозначно знайти значення ПФ за результатами кінцевого числа дій. Надалі будемо використовувати цей метод не тільки в алгебрі висловлювань.

За допомогою цього методу можна оцінити складні висловлювання, подавши їх у вигляді ПФ.

Приклад 2.2

Нехай ПФ p, q, r позначають висловлювання:

p : Джон любить футбол,

q : Джон любить баскетбол,

r : Джон любить бокс.

Потрібно записати висловлювання у вигляді ПФ і побудувати відповідну таблицю істинності: «Джон любить футбол, і невірно те, що він любить баскетбол чи бокс».

Замість «Джон любить футбол» у ПФ підставимо змінну p .

Замість виразу «Неправильно те, що він любить баскетбол чи бокс» запишемо: $\neg(q \vee r)$. А замінивши зв'язку «і» на логічну операцію « $\wedge$ », одержимо $p \wedge \neg(q \vee r)$.

За пріоритетом логічних операцій, використовуючи дужки в ПФ, спочатку необхідно виконати $q \vee r$, потім $\neg(q \vee r)$ і вже потім $p \wedge \neg(q \vee r)$.

Результат розміщено в таблиці значень ПФ (табл. 2.4).

Таблиця 2.4

Таблиця істинності для побудованої ПФ: $p \wedge \neg(q \vee r)$

p	q	r	$q \vee r$	$\neg(q \vee r)$	$p \wedge \neg(q \vee r)$
0	0	0	0	1	0
0	0	1	1	0	0
0	1	0	1	0	0
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

Приклад 2.3

Нехай пропозиційними змінними p, q, r позначені висловлювання:

p : «студент ходить на лекції»,

q : «студент виконує лабораторні роботи»,

r : «студент складає сесію».

Потрібно записати як складне висловлювання ПФ:

$$(p \supset q) \wedge (q \supset r)$$

і побудувати для нього таблицю істинності.

Оскільки логічна операція $\supset$ інтерпретується як «якщо, то», а символ « $\wedge$ » – як «і», одержимо:

«Якщо студент відвідує лекції, то він виконує лабораторні роботи, і якщо він виконує лабораторні роботи, то він складає сесію».

Позначимо цифрами порядок виконання логічних операцій:

$$\overset{1}{(p \supset q)} \wedge \overset{3}{(q \supset r)} \overset{2}{} \quad$$

і складемо для них таблицю істинності (табл. 2.5)

Таблиця 2.5

Результати виконання операцій для ПФ: $(p \supset q) \wedge (q \supset r)$

p	q	r	$p \supset q, (1)$	$q \supset r, (2)$	$(p \supset q) \wedge (q \supset r), (3)$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	0	1	0
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Визначення 2.1. ПФ називається **тотожно істинною, або тавтологією**, якщо ця ПФ набуває значення 1 (істинна) на всіх наборах значень її змінних.

Тавтологію прийнято позначати символом $\models A$ (A – тавтологія).

Визначення 2.2. ПФ називається **тотожно хибною, або протиріччям**, якщо ця ПФ набуває значення 0 (хибність) на всіх наборах значень своїх змінних.

Якщо ПФ A є тавтологією, то, підставляючи в A замість змінних (ПЗ) конкретних висловлювань, одержимо висловлювання, що буде **логічно істинним**. Аналогічно, підставляючи висловлювання у протиріччя, одержимо **логічно хибне** висловлювання. Зрозуміло, що існують такі ПФ, які не є ні тавтологіями, ні протиріччями. Для того щоб визначити, чи є ПФ тавтологією (протиріччям), досить побудувати її таблицю істинності.

Приклад 2.5

Довести, що ПФ $p \supset p \vee q$ є тавтологією. Побудуємо таблицю істинності для цієї ПФ (табл. 2.6).

Таблиця 2.6

Приклад доказу тавтології ПФ за таблицею істинності

p	q	$p \vee q$	$p \supset p \vee q$
0	0	0	1
0	1	1	1
1	0	1	1
1	1	1	1

Результуюча колонка складається тільки з одиниць – «1», отже ПФ є тавтологією:

$$\models p \supset p \vee q.$$

Існують деякі тавтології, що прийнято вважати основними, або базовими (табл. 2.7).

Таблиця 2.7

Фундаментальні формули логічної істинності (тотожності)

Номер з/п	Тавтологія	Назва
1	$\models p \vee \neg p$	Закон виключеного третього
2	$\models \neg \neg p \equiv p$	Закон подвійного заперечення
3	$\models p \wedge p \equiv p$	Закони ідемпотентності
4	$\models p \vee p \equiv p$	
5	$\models p \wedge q \supset p$	Закон спрощення
6	$\models p \supset p \vee q$	Закон введення диз'юнкції (Закон додавання)
7	$\models p \wedge q \equiv q \wedge p$	Закони комутативності
8	$\models p \vee q \equiv q \vee p$	
9	$\models p \wedge (q \wedge r) \equiv (p \wedge q) \wedge r$	Закони асоціативності
10	$\models p \vee (q \vee r) \equiv (p \vee q) \vee r$	
11	$\models p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Закони дистрибутивності
12	$\models p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	
13	$\models \neg(p \wedge q) \equiv \neg p \vee \neg q$	Закони де Моргана
14	$\models \neg(p \vee q) \equiv \neg p \wedge \neg q$	
15	$\models p \supset q \equiv \neg q \supset \neg p$	Закон контрапозиції
16	$\models (p \supset q) \wedge (q \supset r) \supset (p \supset r)$	Закон транзитивності імплікації
17	$\models (\neg p \supset q) \wedge (\neg p \supset \neg q) \supset p$	Закон побічного доказу
18	$\models (p \vee q) \wedge ((p \supset r) \wedge (q \supset r)) \supset r$	Закон розгляду випадків
19	$\models (p \equiv q) \wedge (q \equiv r) \supset (p \equiv r)$	Закон транзитивності еквіваленції
20	$\models (p \equiv q) \equiv (\neg p \equiv \neg q)$	Закон протилежності
21	$\models 1 \equiv p \vee \neg p, \quad \models 1 \equiv p \supset p$	Вираження 1 через $\neg, \vee, \supset$
22	$\models 0 \equiv p \wedge \neg p, \quad \models 0 \equiv \neg(p \supset p)$	Вираження 0 через $\neg, \wedge, \supset$
23	$\models p \supset q \equiv \neg p \vee q$	Вираження $\supset$ через $\neg, \vee$
24	$\models (p \equiv q) \equiv (p \supset q) \wedge (q \supset p)$	Вираження $\equiv$ через $\wedge, \supset$
25	$\models p \wedge q \equiv \neg(p \supset \neg q)$	Вираження $\wedge$ через $\neg, \supset$
26	$\models p \vee q \equiv \neg p \supset q$	Вираження $\vee$ через $\neg, \supset$

Приклад 2.6

Визначити, чи є ПФ, що має вигляд: $(p \supset q) \wedge r$, тавтологією? Будуємо таблицю істинності (табл. 2.7).

Із табл. 2.8 видно, що зазначена ПФ не є тавтологією. Але ж видно, що вона і протиріччям також не є. Така ПФ зветься *контингентною*.

Таблиця 2.8

Приклад контингентної пропозиційної формули

p	q	r	$p \supset q$	$(p \supset q) \wedge r$
0	0	0	1	0
0	0	1	1	1
0	1	0	1	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Приклад 2.7

Довести за допомогою таблиці істинності, що ПФ $p \supset q \equiv \neg p \vee q$ є тавтологією.

Проставимо порядок виконання логічних операцій:

1 4 2 3

$\models p \supset q \equiv \neg p \vee q$

і складемо таблицю істинності (табл. 2.9).

Таблиця 2.9

Результати виконання операцій для ПФ: $\models p \supset q \equiv \neg p \vee q$

p	q	$p \supset q, (1)$	$\neg p, (2)$	$\neg p \vee q, (3)$	$p \supset q \equiv \neg p \vee q, (4)$
0	0	1	1	1	1
0	1	1	1	1	1
1	0	0	0	0	1
1	1	1	0	1	1

Оскільки результуючий стовпець таблиці складається тільки з одиниць, то можна зробити висновок, що зазначена ПФ вигляду $p \supset q \equiv \neg p \vee q$ є тавтологією.

Очевидно, що, використовуючи основні тавтології для ПФ і два вище наданих твердження, можна виконувати перетворення або заміни одних ПФ на інші. Для цього введемо поняття рівносильності ПФ.

2.2. Рівносильні пропозиційні формули

Визначення 2.5. Дві ПФ $A(x_1, \dots, x_n)$ і $B(x_1, \dots, x_n)$ називаються **рівносильними**, якщо на будь-яких наборах значень змінних $x_1, \dots, x_n$ значення A і B збігаються.

Рівносильність ПФ A і B позначається $A \sim B$, або

$$A(x_1, \dots, x_n) \sim B(x_1, \dots, x_n).$$

Вочевидь, якщо $\models A$, то $A \sim 1$, а якщо A – протиріччя, то $A \sim 0$.

Зв'язок між поняттями *тавтологія* і *рівносильність* визначається теоремою.

Теорема 2.1. $A \sim B$ тоді і тільки тоді, коли $\models A \equiv B$.

Існує набір основних чи базових рівносильностей, визначених на ПФ: p, q, r (табл. 2.10).

Таблиця 2.10

Перелік логічних рівносильностей для ПФ

Номер з/п	Рівносильність	Номер з/п	Рівносильність
1°	$\neg\neg p \sim p$	14°	$p \wedge \neg p \sim 0$
2°	$p \wedge q \sim q \wedge p$	15°	$p \wedge 0 \sim 0$
3°	$p \vee q \sim q \vee p$	16°	$p \vee 1 \sim 1$
4°	$p \wedge (q \wedge r) \sim (p \wedge q) \wedge r$	17°	$1 \supset p \sim p$
5°	$p \vee (q \vee r) \sim (p \vee q) \vee r$	18°	$p \supset 1 \sim 1$
6°	$p \wedge (q \vee r) \sim (p \wedge q) \vee (p \wedge r)$	19°	$p \supset 0 \sim \neg p$
7°	$p \vee (q \wedge r) \sim (p \vee q) \wedge (p \vee r)$	20°	$0 \supset p \sim 1$
8°	$\neg(p \wedge q) \sim \neg p \vee \neg q$	21°	$p \supset q \sim \neg p \vee q$
9°	$\neg(p \vee q) \sim \neg p \wedge \neg q$	22°	$(p \equiv q) \sim (\neg p \vee q) \wedge (\neg q \vee p)$
10°	$p \vee p \sim p$	23°	$(p \equiv q) \sim p \wedge q \vee \neg p \wedge \neg q$
11°	$p \wedge p \sim p$	24°	$p \supset q \wedge r \sim (p \supset q) \wedge (p \supset r)$
12°	$p \wedge 1 \sim p$	25°	$p \wedge q \supset r \sim p \supset (q \supset r)$
13°	$p \vee 0 \sim p$	26°	$p \supset \neg q \sim q \supset \neg p$

Приклад 2.8

Доведемо рівносильність 9° для довільних ПФ A і B .

З рівносильності 8° маємо:

$$\neg(A \wedge B) \sim \neg A \vee \neg B.$$

Підставимо у формулу замість пропозиціональних змінних A і B пропозиціональні змінні $\neg A$ і $\neg B$:

$$\neg(\neg A \wedge \neg B) \sim \neg\neg A \vee \neg\neg B.$$

Застосувавши заперечення до обох частин отриманої рівносильності, одержимо:

$$\neg\neg(\neg A \wedge \neg B) \sim \neg(\neg\neg A \vee \neg\neg B).$$

Використовуємо рівносильність 1° для отримання рівносильності 9°:

$$\neg A \wedge \neg B \sim \neg(A \vee B).$$

Приклад 2.9

Довести рівносильність (нерівносильність) ПФ A і B двома способами: за допомогою таблиці істинності й аналітично, використовуючи основні рівносильності для ПФ:

$$A = p \vee q, \quad B = (p \supset q) \supset q.$$

Якщо ПФ A і B рівносильні, то:

$$p \vee q \sim (p \supset q) \supset q.$$

За теоремою 4.1, ПФ вигляду $\models p \vee q \equiv (p \supset q) \supset q$ є тавтологією. Доведемо це за допомогою таблиці істинності (табл. 2.11).

Таблиця 2.11

Приклад доведення рівносильності двох ПФ

p	q	$p \vee q$	$p \supset q$	$(p \supset q) \supset q$	$p \vee q \equiv (p \supset q) \supset q$
0	0	0	1	0	1
0	1	1	1	1	1
1	0	1	0	1	1
1	1	1	1	1	1

Тавтологію доведено, отже, доведено і рівносильність.

2.3. Логічний наслідок

Визначення 2.6. ПФ B називається **логічним наслідком** ПФ $A_1, A_2, \dots, A_m$, якщо для будь-якого набору значень змінних $x_1, \dots, x_n$, які входять до ПФ $A_1, \dots, A_m$ і B , значення B дорівнює одиниці усякий раз, коли на цьому ж наборі значення кожної з ПФ A_i ($1 \leq i \leq m$) дорівнює одиниці. Запис $A_1, A_2, \dots, A_m \models B$ означає, що «з $A_1, A_2, \dots, A_m$, логічно виходить B ».

Таким чином, із визначення виходить, що будь-яка ПФ є логічним наслідком протиріччя, а тавтологія – логічним наслідком будь-якої ПФ.

Теорема 2.2.

(а) $A \models B$ тоді і тільки тоді, коли $\models (A \supset B)$;

(б) $A_1, A_2, \dots, A_m \models B$ тоді і тільки тоді, коли:

$$\models (A_1 \supset (A_2 \supset (\dots (A_m \supset B) \dots))) ; \quad (m \geq 2).$$

Наведене вище визначення **логічного наслідку** відповідає логічному наслідку при дедуктивних висновках: висловлювання J є логічний наслідок

висловлень $w_1, w_2, \dots, w_m$, якщо висновок про істинність кожного з висловлень w_i ($i = 1, \dots, m$) дає істинність висловлювання J без додаткової інформації; у випадку хибності хоча б одного з w_i не можна стверджувати істинність J . У логіці висловлень міркування вважається **логічно правильним**, якщо воно є логічним наслідком.

Приклад 2.10

Висловлювання «студент піде додому (J) або залишиться в університеті (w); він не залишиться в університеті, отже, піде додому» є правильним.

Дійсно, розглядаючи елементи наведеного складного висловлювання символічно $(J \vee w, \neg w, J)$ як значення ПФ $p \vee q, \neg q, p$, переконуємося в тому, що:

$$\models ((p \vee q) \supset (\neg q \supset p))$$

Це ілюстрація теореми про дедукцію: оскільки формула є тавтологією, то висновок p логічно випливає із засновків $p \vee q$ та $\neg q$, тобто $p \vee q, \neg q \models p$.

2.4. Розкладання ПФ і доскональні нормальні форми

Будь-яку ПФ, виявляється, можна виразити рівносильною формулою у формі диз'юнктивного чи кон'юнктивного розкладання ПФ за якою-небудь її пропозиційною змінною.

Диз'юнктивне розкладання ПФ $A(x_1, \dots, x_i, \dots, x_n)$ за змінною x_i обґрунтовує теорема 2.3.

Теорема 2.3. Для будь-якої ПФ $A(x_1, \dots, x_i, \dots, x_n)$ справедлива рівносильність:

$$A(x_1, \dots, x_i, \dots, x_n) \equiv (x_i \wedge A(x_1, \dots, 1, \dots, x_n)) \vee (\neg x_i \wedge A(x_1, \dots, 0, \dots, x_n)) \quad (2.1)$$

Ця формула також відома як теорема розкладання Шеннона (або Буля). Вона стверджує, що будь-яку булеву функцію можна представити як суму двох підфункцій: перша домножується на змінну x_i (де $x_i = 1$), а друга – на її заперечення $\neg x_i$ (де $x_i = 0$).

Наслідок теореми 2.3. При позначенні $x^J = x$, якщо $J = 1$, та $x^J = \neg x$, якщо $J = 0$, для будь-якої ПФ $A(x_1, x_2, \dots, x_n)$ і будь-якого натурального k ($1 \leq k \leq n$) справедлива рівносильність розкладання за k змінними $x_1, \dots, x_k$:

$$A(x_1, \dots, x_n) \equiv \vee_{(J_1, \dots, J_k)} (x_1^{J_1} \wedge x_2^{J_2} \wedge \dots \wedge x_k^{J_k} \wedge A(J_1, \dots, J_k, x_{k+1}, \dots, x_n)), \quad (2.2)$$

що називається **диз'юнктивним розкладанням** ПФ A за k змінними: $x_1, x_2, \dots, x_k$, де $J_i \in \{0, 1\}, i = 1, \dots, k$.

З рівносильності (2.2) при $k = n$ одержуємо рівняння:

$$A(x_1, \dots, x_n) \equiv \vee_{(J_1, \dots, J_n)} (x_1^{J_1} \wedge x_2^{J_2} \wedge \dots \wedge x_n^{J_n} \wedge A(J_1, \dots, J_n)), \quad (2.3)$$

де $A(J_1, \dots, J_n)$ є нуль чи одиниця.

Звідси, якщо ПФ $A(x_1, \dots, x_n)$ не є протиріччям і містить усі змінні, то права частина рівносильності:

$$A(x_1, \dots, x_n) \equiv \bigvee_{(J_1, \dots, J_n) : A(J_1, \dots, J_n)=1} (x_1^{J_1} \wedge x_2^{J_2} \wedge \dots \wedge x_n^{J_n}) \quad (2.4)$$

називається **досконалою диз'юнктивною нормальною формою (ДДНФ)** ПФ $A(x_1, \dots, x_n)$.

Тут запис $A(J_1, \dots, J_n) = 1$ означає, що вибирають кон'юнкції лише тих наборів J_i ($i = 1, n$), на яких ПФ A має значення одиниці (1).

Приклад 2.11

Знайти диз'юнктивне розкладання за змінною p ПФ: $r \vee p \supset q \wedge r$.

Розв'язання

Згідно з рівнянням (2.1) маємо:

$$\begin{aligned} r \vee p \supset q \wedge r &\equiv (p \wedge (r \vee 1 \supset q \wedge r)) \vee (\neg p \wedge (r \vee 0 \supset q \wedge r)) \\ &\equiv p \wedge (1 \supset q \wedge r) \vee \neg p \wedge (r \supset q \wedge r) \\ &\equiv (p \wedge q \wedge r) \vee (\neg p \wedge (r \supset q \wedge r)). \end{aligned}$$

Приклад 2.12

ПФ $(q \supset r) \vee \neg q$ подати у ДДНФ.

Розв'язання

Згідно з виразом (2.4) для заданої ПФ знаходимо набори значень (J_1, J_2) , на яких функція дорівнює 1:

$$(J_1 \supset J_2) \vee \neg J_1 = 1$$

Перевіримо набори:

(0,0): $(0 \supset 0) \vee 1 = 1 \vee 1 = 1$ (підходить)

(0,1): $(0 \supset 1) \vee 1 = 1 \vee 1 = 1$ (підходить)

(1,0): $(1 \supset 0) \vee 0 = 0 \vee 0 = 0$ (не підходить)

(1,1): $(1 \supset 1) \vee 0 = 1 \vee 0 = 1$ (підходить)

Отже, об'єднуємо кон'юнкції для наборів (0,0), (0,1), (1,1):

$$(q \supset r) \vee \neg q \equiv \bigvee_{\substack{(J_1, J_2) \in \\ \{(0,0), (0,1), (1,1)\}}} (q^{J_1} \wedge r^{J_2}) \equiv (\neg q \wedge \neg r) \vee (\neg q \wedge r) \vee (q \wedge r) - \text{ДДНФ.}$$

За допомогою аналогічних міркувань приходимо до двоїстої теореми і наслідку, що обґрунтовує **кон'юнктивне розкладання ПФ** (КНФ та ДКНФ).

Теорема 2.4. Для будь-якої ПФ $A(x_1, \dots, x_i, \dots, x_n)$ справедлива рівносильність:

$$A(x_1, \dots, x_n) \equiv (x_i \vee A(x_1, \dots, 0, \dots, x_n)) \wedge (\neg x_i \vee A(x_1, \dots, 1, \dots, x_n)), \quad (2.5)$$

що називається **кон'юнктивним розкладанням ПФ** A за змінною x_i ($i = 1, \dots, n$).

З урахуванням позначень $x^J = x$ при $J = 1$ і $x^J = \neg x$ при $J = 0$, рівносильність (2.5) можна записати і так:

$$A(x_1, \dots, x_n) \equiv \bigwedge_{J_i \in \{0,1\}} (\neg x_i^{J_i} \vee A(x_1, \dots, J_i, \dots, x_n)), \quad (2.6)$$

що зберігає загальну форму розкладання ПФ за k змінними при $k = 1$.

Пояснення формули (2.6): у цій формулі використовується принцип двоїстості до диз'юнктивного розкладання:

- якщо $J_i = 0$, то множник перетворюється на:

$$(\neg x_i^0 \vee A(\dots, 0, \dots)) = (\neg(\neg x_i) \vee A(0)) = (x_i \vee A(0));$$

- якщо $J_i = 1$, то множник перетворюється на:

$$(\neg x_i^1 \vee A(\dots, 1, \dots)) = (\neg x_i \vee A(1)).$$

Це повністю відповідає формулі (2.5).

Наслідок теореми 2.4. Для всякої ПФ $A(x_1, x_2, \dots, x_n)$ і будь-якого натурального k ($1 \leq k \leq n$) справедлива рівносильність:

$$A(x_1, \dots, x_n) \equiv \bigwedge_{(J_1, \dots, J_k)} (\neg x_1^{J_1} \vee \dots \vee \neg x_k^{J_k} \vee A(J_1, \dots, J_k, x_{k+1}, \dots, x_n)) \quad (2.7)$$

що зветься **кон'юнктивним розкладанням ПФ** A за k змінними $x_1, x_2, \dots, x_k$, де $J_i \in \{0, 1\}, i = 1, \dots, k$.

Із рівносильності (2.7) при $k = n$ випливає:

$$A(x_1, \dots, x_n) \equiv \bigwedge_{(J_1, \dots, J_n)} (\neg x_1^{J_1} \vee \neg x_2^{J_2} \vee \dots \vee \neg x_n^{J_n} \vee A(J_1, \dots, J_n)), \quad (2.8)$$

де $A(J_1, \dots, J_n) \in \{0, 1\}$.

Звідси, якщо ПФ $A(x_1, \dots, x_n)$ не є тавтологією (тобто не дорівнює тотожній одиниці) і містить усі змінні, то права частина рівносильності:

$$A(x_1, \dots, x_n) \equiv \bigwedge_{\substack{(J_1, \dots, J_n): \\ A(J_1, \dots, J_n) = 0}} (\neg x_1^{(J_1)} \vee \neg x_2^{(J_2)} \vee \dots \vee \neg x_n^{(J_n)}) \quad (2.9)$$

називається **досконалою кон'юнктивною нормальною формою (ДКНФ)** ПФ $A(x_1, \dots, x_n)$.

Тут запис $A(J_1, \dots, J_n) = 0$ означає, що кон'юнктуються (об'єднуються через "і") диз'юнкції ("АБО") лише тих наборів $J_1, \dots, J_n$, на яких ПФ A має значення 0.

У цій нотації вираз $\neg x^J$ означає:

- $\neg x$, якщо $J = 1$ (щоб терм $\neg 1 = 0$ не впливав на диз'юнкцію);
- x (тобто $\neg(\neg x)$), якщо $J = 0$ (щоб терм 0 не впливав на диз'юнкцію).

Це відповідає правилу побудови ДКНФ: ми беремо «заперечення» змінних для наборів, що дають 0.

Приклад 2.13

Для ПФ $q \wedge r \supset \neg s$ знайти кон'юнктивне розкладання за змінними q і r , а також ДКНФ.

Розв'язання

Згідно з рівносильністю (2.7) маємо розкладання за $k = 2$ змінними (q, r) :

$$q \wedge r \supset \neg s \equiv \bigwedge_{(J_1, J_2)} (\neg q^{J_1} \vee \neg r^{J_2} \vee (J_1 \wedge J_2 \supset \neg s)).$$

Розпишемо для всіх наборів $(J_1, J_2) \in \{(0,0), (0,1), (1,0), (1,1)\}$:

$$\begin{aligned} &\equiv (q \vee r \vee (0 \wedge 0 \supset \neg s)) \wedge \\ &\quad (q \vee \neg r \vee (0 \wedge 1 \supset \neg s)) \wedge \\ &\quad (\neg q \vee r \vee (1 \wedge 0 \supset \neg s)) \wedge \\ &\quad (\neg q \vee \neg r \vee (1 \wedge 1 \supset \neg s)). \end{aligned}$$

Враховуючи властивість імплікації $(0 \supset A) \equiv 1$, перші три дужки спрощуємо до 1 (оскільки $X \vee 1 \equiv 1$):

$$\begin{aligned} &\equiv (q \vee r \vee 1) \wedge (q \vee \neg r \vee 1) \wedge (\neg q \vee r \vee 1) \wedge (\neg q \vee \neg r \vee \neg s) \\ &\equiv 1 \wedge 1 \wedge 1 \wedge (\neg q \vee \neg r \vee \neg s) \\ &\equiv \neg q \vee \neg r \vee \neg s. \end{aligned}$$

Враховуючи рівносильність (2.9), знаходимо ДКНФ за допомогою таблиці істинності ПФ (табл. 2.12). Нас цікавлять лише ті рядки, де функція дорівнює 0.

$$q \wedge r \supset \neg s \equiv \bigwedge_{\substack{(J_1, J_2, J_3): \\ F=0}} (\neg q^{J_1} \vee \neg r^{J_2} \vee \neg s^{J_3}).$$

Таблиця 2.12

Таблиця істинності пропозиційної формули $q \wedge r \supset \neg s$

q	r	s	$q \wedge r$	$\neg s$	$q \wedge r \supset \neg s$
0	0	0	0	1	1
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	0	1
1	1	0	1	1	1
1	1	1	1	0	0

З таблиці видно, що функція дорівнює 0 лише на наборі (1, 1, 1).

Отже:

$$\equiv (\neg q^1 \vee \neg r^1 \vee \neg s^1) \equiv \neg q \vee \neg r \vee \neg s - \text{ДКНФ.}$$

Потрібну ДКНФ простіше можна одержати з рівносильностей (відомих логічних законів) таким чином:

– заміна імплікації: $A \supset B \equiv \neg A \vee B$; (*)

застосовуємо до нашого виразу:

(*)

$$(q \wedge r) \supset \neg s \overset{(*)}{\equiv} \neg (q \wedge r) \vee \neg s;$$

– закон де Моргана: $\neg (A \wedge B) \equiv \neg A \vee \neg B$; (**).

розкриваємо дужки із запереченням:

(**)

$$\neg (q \wedge r) \vee \neg s \overset{(**)}{\equiv} \neg q \vee \neg r \vee \neg s.$$

Відповідь: $\neg q \vee \neg r \vee \neg s$.

2.5. Диз'юнктивна і кон'юнктивна нормальні форми

Нехай $J_1, \dots, J_k \in \{0,1\}$. Введемо позначення для літери зі степенем (2.10):

$$x^{J_k} = \begin{cases} x, & \text{якщо } J_k = 1, \\ \neg x, & \text{якщо } J_k = 0. \end{cases} \quad (2.10)$$

Пропозиційна формула (ПФ) вигляду:

$$x_{i_1}^{J_1} \wedge x_{i_2}^{J_2} \wedge \dots \wedge x_{i_k}^{J_k} \quad (k \geq 1) \quad (2.11)$$

називається **елементарною кон'юнкцією** (ЕК).

ПФ вигляду:

$$x_{i_1}^{J_1} \vee x_{i_2}^{J_2} \vee \dots \vee x_{i_k}^{J_k} \quad (k \geq 1) \quad (2.12)$$

називається **елементарною диз'юнкцією** (ЕД).

Дві ЕК (чи ЕД) називаються рівними, якщо вони або збігаються, або виходять одна із одної перестановкою членів, тобто є рівносильними ПФ.

– **Диз'юнктивною нормальною формою (ДНФ)** зазначеної ПФ називається диз'юнкція елементарних кон'юнкцій.

– **Кон'юнктивною нормальною формою (КНФ)** зазначеної ПФ називається кон'юнкція елементарних диз'юнкцій.

Теорема 2.5 (про існування ДНФ). Для будь-якої пропозиційної формули існує рівносильна їй диз'юнктивна нормальна форма.

Теорема 2.6 (про існування КНФ). Для будь-якої пропозиційної формули існує рівносильна їй кон'юнктивна нормальна форма.

Ці теореми гарантують, що яку б складну логічну структуру ми не розглядали (з імплікаціями, еквіваленціями тощо), її завжди можна звести до «плоского» вигляду: або як суму добутків (ДНФ), або як добуток сум (КНФ).

Важливо розуміти, що для конкретної ПФ її ДНФ і КНФ не є єдиними. Наприклад, функцію можна подати у вигляді повної ДНФ, а можна спростити, поглинувши деякі змінні. Обидва вирази будуть ДНФ і будуть рівносильними, але візуально різними.

Це створює незручності для автоматичного порівняння формул. Щоб уникнути цього, вводять поняття **досконалих** нормальних форм (ДДНФ та ДКНФ). Для будь-якої формули (яка не є тотожною константою) її ДДНФ та ДКНФ є єдиними з точністю до перестановки доданків чи множників.

Алгоритм отримання досконалих форм. Якщо ми маємо звичайну ДНФ, перехід до ДДНФ здійснюється шляхом розщеплення елементарних кон'юнкцій за відсутніми змінними.

Для елементарної кон'юнкції K , яка не містить змінної x_n , застосовують рівносильне перетворення (2.13):

$$K \equiv K \wedge (x_n \vee \neg x_n) \equiv (K \wedge x_n) \vee (K \wedge \neg x_n). \quad (2.13)$$

Аналогічно, для отримання ДКНФ з КНФ, до елементарної диз'юнкції D , що не містить змінної x_n , застосовують перетворення (2.14):

$$D \equiv D \vee (x_n \wedge \neg x_n) \equiv (D \vee x_n) \wedge (D \vee \neg x_n). \quad (2.14)$$

Приклад 2.14

Знайдемо диз'юнктивну нормальну форму (ДНФ) для пропозиційної формули $\neg(a \wedge b) \equiv c$.

Розв'язання

1. Скористаємося означенням еквівалентності:

$$(X \equiv Y \Leftrightarrow (X \wedge Y) \vee (\neg X \wedge \neg Y)).$$

У нашому випадку $X = \neg(a \wedge b)$, $Y = c$:

$$\neg(a \wedge b) \equiv c \Leftrightarrow (\neg(a \wedge b) \wedge c) \vee (\neg\neg(a \wedge b) \wedge \neg c)$$

2. Застосуємо закон подвійного заперечення ($\neg\neg A = A$) до другої дужки та закон де Моргана ($\neg(A \wedge B) = \neg A \vee \neg B$) до першої дужки:

$$\Leftrightarrow ((\neg a \vee \neg b) \wedge c) \vee (a \wedge b \wedge \neg c)$$

3. Використовуємо закон дистрибутивності:

$$((A \vee B) \wedge C = (A \wedge C) \vee (B \wedge C)),$$

щоб розкрити дужки в першій частині виразу:

$$\Leftrightarrow (\neg a \wedge c) \vee (\neg b \wedge c) \vee (a \wedge b \wedge \neg c).$$

Відповідь: $\neg a \wedge c \vee \neg b \wedge c \vee a \wedge b \wedge \neg c$.

Приклад 2.15

Перетворимо ДНФ пропозиційної формули $a \wedge (b \supset c)$ на ДДНФ (досконалу диз'юнктивну нормальну форму).

Розв'язання

1. Спочатку знайдемо звичайну ДНФ цієї формули, позбувшись імплікації за формулою $A \supset B \equiv \neg A \vee B$ та розкривши дужки за законом дистрибутивності:

$$a \wedge (b \supset c) \equiv a \wedge (\neg b \vee c) \equiv (a \wedge \neg b) \vee (a \wedge c).$$

Отримали ДНФ. Бачимо, що в першому доданку не вистачає змінної c , а в другому – змінної b .

2. Далі, скориставшись перетвореннями розщеплення (див. формулу 2.13), домножимо кожен доданок на «одиницю» у вигляді суми відсутньої змінної та її заперечення ($(c \vee \neg c)$ та $(b \vee \neg b)$ відповідно):

$$(a \wedge \neg b) \vee (a \wedge c) \equiv a \wedge \neg b \wedge (c \vee \neg c) \vee a \wedge c \wedge (b \vee \neg b).$$

3. Розкриємо дужки та впорядкуємо змінні:

$$\equiv (a \wedge \neg b \wedge c) \vee (a \wedge \neg b \wedge \neg c) \vee (a \wedge b \wedge c) \vee (a \wedge \neg b \wedge c).$$

4. Згідно із законом ідемпотентності ($A \vee A \equiv A$), видалимо дубльований доданок ($a \wedge \neg b \wedge c$):

$$\equiv a \wedge \neg b \wedge c \vee a \wedge \neg b \wedge \neg c \vee a \wedge b \wedge c.$$

Відповідь: $a \wedge \neg b \wedge c \vee a \wedge \neg b \wedge \neg c \vee a \wedge b \wedge c$.

Приклад 2.16

Для пропозиційної формули $p \wedge (q \supset \neg r)$ знайдемо ДКНФ (досконалу кон'юнктивну нормальну форму).

Розв'язання

1. Спочатку знайдемо КНФ цієї формули. Побудуємо імплікації

($A \supset B \equiv \neg A \vee B$):

$$p \wedge (q \supset \neg r) \equiv p \wedge (\neg q \vee \neg r).$$

2. Далі застосуємо перетворення розщеплення (див. формулу 2.14) для доповнення елементарних диз'юнкцій відсутніми змінними.

– Перший множник p не містить змінних q та r . Розщепимо його спочатку за q :

$$p \equiv p \vee (q \wedge \neg q) \equiv (p \vee q) \wedge (p \vee \neg q).$$

– Тепер розщепимо отримані дужки за змінною r :

$$(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r).$$

3. Другий множник початкової КНФ ($\neg q \vee \neg r$) не містить змінної p . Розщепимо його:

$$(\neg q \vee \neg r) \equiv (\neg q \vee \neg r \vee p) \wedge (\neg q \vee \neg r \vee \neg p).$$

4. Об'єднаємо всі отримані множники в один вираз і видалимо дублікати (згідно із законом ідемпотентності $A \wedge A \equiv A$):

$$(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg q \vee \neg r \vee p) \wedge (\neg q \vee \neg r \vee \neg p).$$

Доданок $(p \vee \neg q \vee \neg r)$ та $(\neg q \vee \neg r \vee p)$ є ідентичними з точністю до перестановки).

Відповідь: $(p \vee q \vee r) \wedge (p \vee q \vee \neg r) \wedge (p \vee \neg q \vee r) \wedge (p \vee \neg q \vee \neg r) \wedge (\neg p \vee \neg q \vee \neg r)$.

2.6. Закон подвійності

Пропозиційна формула A^* називається двоїстою до формули A , якщо A^* утворюється з A шляхом заміни всіх логічних операцій $\wedge$ на $\vee$, а $\vee$ на $\wedge$.

Якщо у формулі присутні константи 0 та 1, вони також замінюються одна на одну ($0 \rightarrow 1, 1 \rightarrow 0$).

Властивість симетрії: відношення подвійності є симетричним, тобто, якщо A^* двоїста до A , то $(A^*)^* = A$.

Приклад

Нехай $A = (p \wedge \neg q) \vee r$.

Тоді двоїста формула: $A^* = (p \vee \neg q) \wedge r$.

Якщо тепер взяти двоїсту до A^* , ми повернемося до початкової:

$$((p \vee \neg q) \wedge r)^* = (p \wedge \neg q) \vee r = A.$$

Зв'язок із запереченням

Безпосередньо із законів де Моргана випливає важливе твердження, що пов'язує заперечення формули з її двоїстою формою (2.15):

$$\neg A(x_1, \dots, x_n) \equiv A^*(\neg x_1, \dots, \neg x_n)$$

або

$$A(x_1, \dots, x_n) \equiv \neg A^*(\neg x_1, \dots, \neg x_n). \quad (2.15)$$

Основний закон подвійності:

Якщо дві формули рівносильні ($A \equiv B$), то рівносильні й двоїсті до них формули ($A^* \equiv B^*$).

Приклад 2.17

Застосуємо закон подвійності до відомих законів логіки.

1. Розглянемо дистрибутивний закон:

$$A \wedge (B \vee C) \equiv (A \wedge B) \vee (A \wedge C).$$

За законом подвійності, замінивши $\wedge \leftrightarrow \vee$, одержуємо другу форму дистрибутивного закону (2.16):

$$A^* \vee (B^* \wedge C^*) \equiv (A^* \vee B^*) \wedge (A^* \vee C^*). \quad (2.16)$$

Із рівносильності $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$ за законом подвійності випливає $p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$.

2. Розглянемо закон де Моргана:

$$\neg (A \vee B) \equiv \neg A \wedge \neg B.$$

Застосувавши принцип подвійності, маємо (2.17):

$$\neg (A^* \wedge B^*) \equiv \neg A^* \vee \neg B^*. \quad (2.17)$$

Із $\neg (p \vee q) \equiv \neg p \wedge \neg q$ одержуємо $\neg (p \wedge q) \equiv \neg p \vee \neg q$.

2.7. Повнота мов (системи зв'язувань)

Згідно з доведеними раніше рівносильностями, для будь-якої ПФ існує рівносильна їй формула, що містить лише константи 0, 1, заперечення $\neg$, кон'юнкцію $\wedge$ та диз'юнкцію $\vee$. Іншими словами, будь-яка формула є реалізовною в базисі $\{\neg, \wedge, \vee, 0, 1\}$. У цьому випадку система функцій $\{\neg, \wedge, \vee, 0, 1\}$ називається **повним базисом**. На підставі теорем 2.5 і 2.6 (про ДНФ і КНФ) доведено повноту скороченого базису $\{\wedge, \vee, \neg\}$.

Якщо в базисі $\{\wedge, \vee, \neg\}$ виразити одну з бінарних операцій через інші, можна отримати ще менші базиси. Скориставшись тавтологіями (законами де Моргана):

$$\models A \wedge B \equiv \neg(\neg A \vee \neg B),$$

$$\models A \vee B \equiv \neg(\neg A \wedge \neg B),$$

отримуємо, що системи $\{\neg, \vee\}$ та $\{\neg, \wedge\}$ також є повними базисами в класі ПФ.

З тавтологій:

$$\models A \wedge B \equiv \neg(A \supset \neg B), \quad (2.18)$$

$$\models A \vee B \equiv (\neg A) \supset B \quad (2.19)$$

впливає повнота базису $\{\supset, \neg\}$.

Нарешті, у класі ПФ є два базиси, що складаються тільки з однієї бінарної операції: $\{\downarrow\}$ («стрілка» Пірса) чи $\{\mid\}$ («штрих» Шеффера). Їх визначення наведено в табл. 2.13.

Таблиця 2.13

Операції Пірса та Шеффера

A	B	Стрілка Пірса ($A \downarrow B$)	Штрих Шеффера ($A \mid B$)
0	0	1	1
0	1	0	1
1	0	0	1
1	1	0	0

Дійсно, мова $\{\downarrow\}$ повна, оскільки в базисі $\{\neg, \wedge\}$ можна скористатися тавтологіями

$$\models \neg A \equiv A \downarrow A, \quad (2.20)$$

$$\models A \wedge B \equiv (A \downarrow A) \downarrow (B \downarrow B). \quad (2.21)$$

Мова $\{\mid\}$ повна, тому що в базисі $\{\neg, \vee\}$ можна скористатися тавтологіями:

$$\models \neg A \equiv A \mid A, \quad (2.22)$$

$$\models A \vee B \equiv (A \mid A) \mid (B \mid B). \quad (2.23)$$

Очевидно, операції $A \downarrow B$ і $A \mid B$ є двоїстими одна до одної.

2.8. Вправи до розділу

1. Чи є ці речення висловлюваннями? Указати їх істинне значення.
 - а) Який сьогодні день тижня?
 - б) Ціле число одиниця – це найменше додатне ціле число.
 - в) Не ходіть по залізничних коліях!
 - г) Якщо ціле число x – парне, то число $2 * x$ – теж парне.
 - д) Усі непарні числа діляться на 3.
 - е) Юпітер – найближча до Сонця планета сонячної системи.

- ж) Якщо $x = 3$, то $x * 2 = 6$.
- з) Бережіться автомобіля!
- и) Не зберігайте телефон у мікрохвильовій печі!
- к) Париж – столиця Франції.

2. Нехай пропозиціональні змінні (ПЗ) p , q , r позначають висловлювання:

- p : подорож у космос коштує дорого;
- q : я здійснию космічну подорож;
- r : у мене є гроші.

Записати висловлення у вигляді пропозиціональної формули (ПФ).

- а) У мене немає грошей, і я не здійснию подорож у космос.
- б) У мене немає грошей, і подорож у космос коштує дорого; чи здійснию космічну подорож?
- в) Правильно те, що в мене є гроші і я полечу в космос.
- г) Подорож у космос не є дорогою, і я здійснию космічну подорож; подорож у космос є дорогою, і я не полечу у космос.

3. Нехай ПЗ p , q , r позначають висловлювання:

- p : у мене потужний комп'ютер;
- q : я закінчу проект вчасно;
- r : я складу іспит з інформатики.

Записати у вигляді ПФ складні висловлювання й оцінити їх за допомогою таблиць істинності:

- а) У мене не потужний комп'ютер, я не завершу проєкт вчасно.
- б) Я не завершу проєкт вчасно і не складу іспит з інформатики.
- в) Неправильно те, що я закінчу проєкт вчасно і складу іспит з інформатики.
- г) У мене потужний комп'ютер, я не завершу проєкт вчасно і складу іспит з інформатики.

4. Нехай ПЗ p , q , r позначають висловлювання:

- p : ця гра занадто складна;
- q : я граю в шахи;
- r : гра в шахи потребує часу.

Перелічені нижче ПФ інтерпретувати як складні висловлювання і побудувати для них таблиці істинності:

- а) $q \wedge r$;
- б) $q \vee r$;
- в) $(p \vee r) \wedge q$;
- г) $\bar{p} \vee \bar{q}$.

5. Нехай ПЗ p, q, r позначають висловлювання:

- p : доги – великі собаки;
- q : у мене маленька квартира;
- r : у мене є дог.

Перелічені нижче ПФ інтерпретувати як складні висловлювання і побудувати для них таблиці істинності:

- а) $p \wedge q \vee \bar{r}$;
- б) $p \wedge (\bar{q} \vee \bar{r})$;
- в) $(p \vee \bar{q}) \wedge r$;
- г) $(p \wedge r) \vee (q \wedge \bar{r})$.

6. Нехай ПЗ p, q, r позначають висловлювання:

- p : він купить новий комп'ютер;
- q : він буде святкувати всю ніч;
- r : він виграє в лотерею.

Записати ці висловлювання у вигляді ПФ:

- а) Якщо він виграє в лотерею, то він купить новий комп'ютер і буде святкувати всю ніч.
- б) Якщо він не купить новий комп'ютер, то і святкувати всю ніч не буде.
- в) Якщо він виграє в лотерею, то буде святкувати всю ніч, і якщо він не виграє в лотерею, то не купить новий комп'ютер.
- г) Якщо він не виграє в лотерею і не купить новий комп'ютер, то святкувати всю ніч не буде.

7. Нехай ПЗ p, q, r позначають висловлювання:

- p : він читає спеціальну літературу;
- q : він любить наукову фантастику;
- r : він – програміст.

Записати подані далі висловлювання у вигляді ПФ і оцінити їх за допомогою таблиць істинності:

- а) Якщо він читає спеціальну літературу і любить наукову фантастику, то він – програміст.
- б) Якщо він не читає спеціальну літературу і не любить наукову фантастику, то він не програміст.
- в) Якщо він читає спеціальну літературу, то він любить наукову фантастику, і якщо він не читає спеціальну літературу, то він – програміст.
- г) Якщо він – програміст, то він читає спеціальну літературу або він не любить наукову фантастику.

8. Нехай ПЗ p, q, r позначають висловлювання:

p : йому подобаються яскраві краватки;

q : він популярний;

r : у нього незвичайні друзі.

Записати подані далі висловлювання у вигляді ПФ і оцінити їх за допомогою таблиць істинності:

а) $(p \wedge q) \supset r$;

б) $q \supset \neg r$;

в) $p \supset (q \vee r)$;

г) $(p \supset \neg q) \wedge (q \supset r)$.

9. Нехай ПЗ p, q, r позначають висловлювання:

p : він щасливий;

q : він популярний;

r : він багатий.

Записати висловлювання у вигляді ПФ і оцінити їх за допомогою таблиць істинності:

а) $\neg (p \supset q)$;

б) $(p \vee r) \supset q$;

в) $q \equiv (p \wedge r)$;

г) $(p \supset q) \wedge (\neg r \supset (\neg p \vee \neg q))$.

10. Побудувати для наступних ПФ $A(x, y, z)$ відповідні таблиці істинності:

а) $A = (x \supset y) \vee ((y \supset \bar{z}) \supset x \wedge y)$;

б) $A = \overline{(y \wedge z \supset x)}$;

в) $A = \overline{(y \supset (x \vee z))}$;

г) $A = x \supset ((y \supset z) \supset y \wedge z)$;

д) $A = \overline{(x \wedge y)} \vee (x \equiv z)$;

е) $A = \bar{x} \wedge (y \wedge z) \vee \overline{(x \supset z)}$;

ж) $A = (x \supset y \wedge z) \vee \overline{(x \supset y)}$;

з) $A = (\bar{x} \vee y) \supset (y \wedge \bar{z} \supset (x \equiv x \wedge z))$.

11. За допомогою таблиці істинності перевірити, чи є дані ПФ тавтологіями:

а) $\overline{(x \wedge y \wedge z)} \equiv \bar{x} \vee \bar{y} \vee \bar{z}$;

б) $(\bar{x} \vee \bar{z}) \wedge y \equiv (\bar{x} \supset z) \wedge (y \wedge (y \vee z))$;

в) $(x \supset (y \supset z)) \equiv (x \supset y) \supset (x \supset z)$;

г) $x \supset (y \wedge z) \equiv (x \supset y) \wedge (x \supset z)$;

- д) $x \vee y \equiv (x \supset y) \supset y$;
- е) $x \wedge (y \equiv z) \equiv (((x \wedge y) \equiv (x \wedge z)) \equiv x)$;
- ж) $((x \supset y) \supset y) \equiv (x \supset y)$;
- з) $\overline{(x \wedge \bar{y})} \vee z \equiv (\bar{x} \supset \bar{y}) \supset z$;
- и) $(x \wedge \bar{y} \wedge \bar{z}) \vee (\overline{(\bar{x} \supset \bar{y})} \supset \bar{z})$;
- к) $(\bar{z} \vee (y \supset x)) \vee (\bar{z} \supset \bar{y} \wedge \bar{x})$.

12. За допомогою таблиць істинності переконатися в рівносильності ПФ А і В:

- а) $A = x \vee (y \equiv z), B = (x \vee y) \equiv (x \vee z)$;
- б) $A = x \wedge (y \equiv z), B = (x \wedge y) \equiv (x \wedge z)$;
- в) $A = x \supset (y \equiv z), B = (x \supset y) \equiv (x \supset z)$;
- г) $A = x \vee (y \supset z), B = (x \vee y) \supset (x \vee z)$;
- д) $A = x \wedge (y \supset z), B = (x \wedge y) \supset (x \wedge z)$;
- е) $A = x \supset ((y \supset z) \supset y \wedge z), B = (x \vee (y \supset z)) \wedge (x \vee \bar{y})$;
- ж) $A = (\bar{x} \vee \overline{(y \supset z)} \supset (\bar{x} \vee \bar{y})), B = \overline{(x \wedge y \supset z)} \supset (\bar{y} \vee \bar{z})$.

13. Побудувати таблиці істинності для формул:

- а) $(p \supset q) \vee (p \supset (q \wedge p))$;
- б) $\neg((p \supset \neg(q \wedge p))) \supset \neg p$;
- в) $p \wedge (q \supset p) \supset \neg p$;
- г) $((p \wedge \neg q) \supset q) \supset (p \supset q)$;
- д) $(p \wedge (q \vee \neg p)) \wedge ((\neg q \supset p) \vee q)$;
- е) $(p \supset (q \supset r)) \supset ((p \supset q) \supset (p \supset r))$.

14. Довести, що наведені нижче пропозиційні формули (ПФ) є тавтологіями:

- а) $(p \supset q) \vee (p \supset \neg q)$;
- б) $p \supset (q \supset (p \wedge q))$;
- в) $(p \supset q) \supset ((q \supset r) \supset (p \supset r))$;
- г) $(\neg p \supset \neg q) \supset (q \supset p)$;
- д) $p \supset (q \supset p)$;
- е) $(p \supset q) \supset ((p \supset (q \supset r)) \supset (p \supset r))$;
- ж) $(p \wedge q) \supset q$;
- з) $(p \supset (p \vee q))$;
- и) $q \supset (p \vee q)$;
- к) $(p \supset r) \supset ((q \supset r) \supset ((p \vee q) \supset r))$;
- л) $((p \supset q) \supset ((p \supset \neg q) \supset \neg p))$;

- м) $(\neg q \supset \neg p) \supset ((\neg q \supset p) \supset q)$;
- н) $((q \supset r) \supset ((p \vee q) \supset (p \vee r)))$;
- о) $((p \supset q) \supset p) \supset p$;
- п) $\neg p \supset (p \supset q)$;
- р) $p \vee (p \wedge q) \equiv p$;
- с) $p \wedge (q \vee \neg q) \equiv p$;
- т) $p \vee (q \wedge \neg q) \equiv p$;
- у) $(p \vee q) \wedge (p \vee q) \equiv p$;
- ф) $(p \wedge q) \vee ((p \vee q) \wedge (\neg p \vee \neg q)) \equiv (p \vee q)$;
- х) $p \vee (\neg p \wedge q) \equiv (p \vee q)$;
- ц) $(p \wedge q) \supset p$.

15. Виконати відповідне розкладання ПФ за вказанною змінною:

- 1) $\neg p \vee (q \supset r) \vee (r \supset \neg p)$:
 - а) диз'юнктивне розкладання за змінною p ;
 - б) кон'юнктивне розкладання за змінною p ;
- 2) $(\neg r \equiv p) \supset (\neg q \supset r)$:
 - а) диз'юнктивне розкладання за змінною q ;
 - б) кон'юнктивне розкладання за змінною q ;
- 3) $(q \vee r) \supset \neg (p \vee q \vee r)$:
 - а) диз'юнктивне розкладання за змінною q ;
 - б) кон'юнктивне розкладання за змінною p ;
- 4) $((\neg p \vee \neg q) \supset r) \supset p$:
 - а) диз'юнктивне розкладання за змінною p ;
 - б) кон'юнктивне розкладання за змінною q ;
- 5) $(p \supset r) \vee (\neg q \equiv \neg r)$:
 - а) диз'юнктивне розкладання за змінною r ;
 - б) кон'юнктивне розкладання за змінною p ;
- 6) $\neg q \wedge (p \supset r) \supset (q \supset p)$:
 - а) диз'юнктивне розкладання за змінною q ;
 - б) кон'юнктивне розкладання за змінною p ;
- 7) $q \wedge \neg r \supset \neg p \vee \neg q$:
 - а) диз'юнктивне розкладання за змінною q ;
 - б) кон'юнктивне розкладання за змінною r ;
- 8) $(\neg p \supset q) \vee (p \supset r)$:
 - а) диз'юнктивне розкладання за змінною r ;
 - б) кон'юнктивне розкладання за змінною r ;

9) $(q \equiv p) \vee (p \supset r)$:

- а) диз'юнктивне розкладання за змінною r ;
- б) кон'юнктивне розкладання за змінною p ;

10) $(p \equiv r) \wedge (q \equiv \neg p)$:

- а) диз'юнктивне розкладання за змінною q ;
- б) кон'юнктивне розкладання за змінною p .

16. Виконати розкладання ПФ за двома змінними:

- а) $(p \supset q) \wedge (r \equiv p)$ – диз'юнктивне за p і q ;
- б) $(p \supset r) \equiv (\neg q \supset \neg p)$ – диз'юнктивне за q і r ;
- в) $\neg r \vee (q \supset p) \supset (r \supset \neg q)$ – диз'юнктивне за p і r ;
- г) $(\neg r \supset p) \supset (\neg q \equiv r)$ – диз'юнктивне за p і r ;
- д) $p \wedge \neg q \supset \neg r \wedge q$ – кон'юнктивне за p і r ;
- е) $\neg (q \wedge r \vee p) \supset (p \vee q)$ – кон'юнктивне за q і p ;
- ж) $(p \vee \neg r) \supset (q \equiv \neg p)$ – кон'юнктивне за r і p ;
- з) $(\neg r \supset \neg p) \equiv (q \vee r \vee \neg p)$ – кон'юнктивне за q і p .

17. Знайти досконалу диз'юнктивну нормальну форму (ДДНФ) зазначених ПФ двома способами (табличним і аналітичним):

- а) $\neg (q \vee p) \equiv (\neg p \supset r)$;
- б) $\neg (p \supset q) \equiv (\neg p \vee q \vee r)$;
- в) $\neg (p \supset q) \wedge (r \vee \neg q \vee p)$;
- г) $(\neg p \supset q) \supset (\neg r \equiv p)$;
- д) $(p \supset q) \supset (p \supset r) \vee (q \supset r)$;
- е) $\neg (p \vee q \supset r) \wedge (p \equiv r)$;
- ж) $((p \equiv q) \supset (\neg r \equiv q)) \wedge (r \supset q)$;
- з) $(\neg r \wedge \neg p \wedge (q \equiv r)) \supset \neg p \vee r$.

18. Перетворити задані пропозиційні формули із диз'юнктивної нормальної формули (ДНФ) на досконалу диз'юнктивну нормальну форму (ДДНФ):

- а) $p \wedge q \vee \neg r$;
- б) $\neg p \wedge \neg q \vee q \wedge \neg r \vee p \wedge \neg r$;
- в) $p \vee q \wedge r \vee \neg q \wedge \neg r$;
- г) $p \wedge q \wedge \neg r \vee p \wedge r \wedge \neg s$;
- д) $p \vee \neg q \vee \neg p \wedge r$;
- е) $p \wedge \neg q \vee p \wedge r \vee \neg r$;
- ж) $p \wedge q \vee \neg q \wedge s \vee r \wedge \neg s$;
- з) $p \vee q \wedge r \vee \neg p \wedge s$.

19. Використовуючи рівносильності для ПФ, перейти від заданої кон'юнктивної нормальної форми (КНФ) до ДНФ:

- а) $(\neg p \vee \neg q) \wedge (p \vee r) \wedge (\neg q \vee \neg r)$;
- б) $p \wedge (p \vee \neg q \vee \neg r) \wedge (q \vee \neg r)$;
- в) $(p \vee \neg q) \wedge (p \vee \neg r) \wedge (\neg p \vee \neg q \vee r)$;
- г) $(\neg p \vee q) \wedge (\neg p \vee \neg r) \wedge (\neg p \vee r) \wedge (q \vee r)$;
- д) $(p \vee \neg q \vee r) \wedge (\neg p \vee q \vee \neg r) \wedge (p \vee q \vee r)$;
- е) $(p \vee \neg q) \wedge (q \vee \neg r) \wedge (q \vee s) \wedge (r \vee \neg s)$;
- ж) $(\neg p \vee q \vee \neg r \vee s) \wedge (p \vee \neg q \vee r) \wedge (p \vee s)$;
- з) $(q \vee r \vee \neg s) \wedge (\neg p \vee \neg q \vee \neg r) \wedge (r \vee s)$.

20. Використовуючи рівносильності для ПФ, перейти від заданої диз'юнктивної нормальної форми (ДНФ) до кон'юнктивної нормальної форми (КНФ):

- а) $\neg p \wedge \neg q \vee r$;
- б) $p \wedge \neg q \vee q \wedge \neg r \vee \neg q \wedge r$;
- в) $\neg p \vee q \wedge r \vee \neg q \wedge \neg r$;
- г) $\neg p \vee q \vee p \wedge \neg q \wedge r$;
- д) $\neg p \wedge \neg q \vee \neg q \vee q \wedge \neg r$;
- е) $p \wedge q \vee q \wedge \neg r \vee \neg q \wedge s \vee r \wedge s$;
- ж) $p \wedge q \wedge \neg r \wedge \neg s \vee \neg p \wedge q \wedge \neg r \vee \neg q \wedge s$;
- з) $p \vee q \wedge \neg r \vee \neg p \wedge r \wedge s \vee \neg q \wedge r \wedge \neg s$.

21. Знайти досконалу диз'юнктивну нормальну форму (ДДНФ) для заданих ПФ двома способами (табличним і аналітичним):

- а) $(\neg p \supset \neg q) \supset ((q \wedge r) \supset (p \wedge r))$;
- б) $((p \supset q) \supset \neg p) \supset (p \supset (q \wedge p))$;
- в) $\neg((p \wedge q) \supset \neg p) \wedge \neg((p \wedge q) \supset \neg q)$.

22. Знайти досконалу кон'юнктивну нормальну форму (ДКНФ) зазначених ПФ двома способами (табличним і аналітичним):

- а) $(\neg p \vee \neg q \vee \neg r) \wedge (p \wedge q \vee r)$;
- б) $(y \vee z) \supset \neg(x \vee y \vee z)$;
- в) $((x \equiv z) \supset z) \vee \neg y \supset x$;
- г) $(x \supset z) \vee (\neg y \equiv \neg z)$;
- д) $(z \supset x) \supset (\neg(y \vee z) \supset x)$;
- е) $(\neg x \supset y) \equiv (\neg z \supset x)$;
- ж) $(x \wedge \neg z \wedge y) \supset (\neg x \vee z \supset y)$;
- з) $(x \supset z) \wedge (z \vee \neg x)$.

23. Переконатися в рівносильності ПФ А і В за допомогою закону подвійності.

- а) $A = x \wedge y \vee y \wedge \neg z \vee \neg y \wedge z, B = x \wedge \neg y \wedge \neg z \vee y \wedge z;$
- б) $A = p \vee q \wedge r, B = (p \vee q) \wedge (p \vee r);$
- в) $A = \neg (p \vee q), B = \neg p \wedge \neg q;$
- г) $A = \neg (p \wedge q), B = \neg p \vee \neg q;$
- д) $A = \neg p \vee q, B = p \supset q;$
- е) $A = \neg (\neg p \vee \neg q), B = p \wedge q;$
- ж) $A = \neg (p \vee q) \vee q, B = \neg q \supset \neg (p \wedge q);$
- з) $A = p \wedge (\neg q \vee \neg s), B = (p \wedge \neg q) \vee (p \wedge \neg s);$
- и) $A = \neg p \vee q \wedge \neg s, B = (\neg p \vee q) \wedge (\neg p \wedge \neg s);$
- к) $A = \neg (p \vee q) \vee q, B = \neg q \supset (p \wedge q).$

24. Довести повноту систем логічних зв'язувань за допомогою рівносильностей для ПФ:

- а) $\{ \neg, \wedge, \vee \};$
- б) $\{ \neg, \wedge \};$
- в) $\{ \neg, \vee \};$
- г) $\{ \neg, \supset \};$
- д) $\{ \neg, \equiv \};$
- е) $\{ | \};$
- ж) $\{ \downarrow \}.$

25. Довести таку рівносильність:

- а) $a \wedge \neg b \vee \neg a \wedge b \equiv (a \vee b) \wedge (\neg a \vee \neg b);$
- б) $a \vee b \vee c \vee d \equiv (\neg a \wedge \neg b \wedge \neg c) \rightarrow d;$
- в) $(a \wedge b) \rightarrow c \equiv (a \wedge \neg c) \rightarrow \neg b;$
- г) $(a \sim b) \wedge (a \wedge \neg b \vee b) \equiv a \wedge b.$

26. Довести, що формули алгебри висловлювань є рівносильними:

- а) $(a \wedge b) \rightarrow c;$
- б) $a \wedge \neg c) \rightarrow \neg b;$
- в) $(b \wedge \neg c) \rightarrow \neg a;$
- г) $a \rightarrow (b \rightarrow c).$

27. Визначити, яка із наведених чотирьох формул рівносильна формулі $\neg (a \rightarrow b):$

- а) $\neg a \rightarrow \neg b;$
- б) $\neg a \rightarrow b;$
- в) $a \rightarrow \neg b;$
- г) $a \wedge \neg b.$

28. Довести, що $a \rightarrow b \equiv \neg b \rightarrow \neg a$ (закон контрапозиції).

29. Перевірити (довести або спростувати), чи має місце така рівносильність:

- а) $\neg(a \sim \neg b) \equiv a \wedge b \vee \neg a \wedge \neg b$;
- б) $(a \vee c) \rightarrow (b \vee d) \equiv (a \rightarrow b) \wedge (c \rightarrow d)$;
- в) $a \rightarrow (a \sim b) \equiv (a \rightarrow b)$;
- г) $a \rightarrow (b \rightarrow c) \equiv (a \wedge b) \rightarrow c$;
- д) $(a \vee b \vee \neg c) \rightarrow a \equiv (b \rightarrow a) \wedge (\neg a \rightarrow c)$;
- е) $(a \wedge (a \vee c) \wedge (b \vee c)) \equiv ((a \wedge b) \vee (a \wedge c))$.

2.9. Програмна реалізація вправ

Завдання 1.

Написати програму, яка буде послідовно (згідно з пріоритетом логічних операцій) будувати таблиці істинності для пропозиційних формул та перевіряти, чи є вони тавтологіями.

У програмі повинна бути функція, яка буде приймати як вхідний параметр пропозиційну формулу як рядок, далі робити парсинг рядка, знаходити найбільш пріоритетну логічну операцію, виконувати її, послідовно будувати таблицю істинності і так далі, поки не виконаються усі логічні операції.

Завдання 2.

Написати програму, яка буде для ПФ будувати її ДДНФ.

Виконання завдання 1.

Написати програму, яка буде послідовно (згідно з пріоритетом логічних операцій) будувати таблиці істинності для пропозиційних формул та перевіряти, чи є вони тавтологіями.

У програмі повинна бути функція, яка буде приймати як вхідний параметр пропозиційну формулу як рядок, далі робити парсинг рядка, знаходити найбільш пріоритетну логічну операцію, виконувати її, послідовно будувати таблицю істинності і так далі, поки не виконаються усі логічні операції.

Для розв'язання завдання була створена програма, яка виконує автоматизований аналіз логічних виразів шляхом генерації відповідних таблиць істинності. Програмний комплекс складається з набору взаємопов'язаних функцій, кожна з яких відповідає за певний етап оброблення логічного виразу.

Введений користувачем логічний вираз піддається лексичному аналізу. Функція *parse_expression* розбиває вираз на послідовність лексем (токенів), забезпечуючи відокремлення операторів, змінних та дужок. Допоміжні функції *is_operator*, *is_bracket* та *is_variable_new* здійснюють ідентифікацію типу токена. Функція *get_variables* агрегує множину унікальних змінних.

Програма реалізує поетапний алгоритм обчислення логічного виразу, дотримуючись встановленої ієрархії пріоритетів логічних операторів: *NOT* (логічне заперечення) має найвищий пріоритет, за ним – *AND* (логічне «І»), *OR* (логічне «АБО»), *IMPL* (імплікація), *EQV* (еквівалентність) та *XOR* (виключне «АБО»). Для виконання обчислень використовується функція *apply_operator*, яка реалізує логіку як унарних (наприклад, заперечення), так і бінарних операцій. У процесі аналізу виразу функція *find_highest_priority_operator* визначає, який оператор слід застосувати наступним відповідно до його пріоритету. Після цього *evaluate_operation* виконує обчислення для знайденого оператора, замінюючи відповідні токени в списку на результат. У випадку відсутності дужок у виразі обчислення виконується функцією *evaluate_simple*, яка поступово застосовує оператори, поки не буде отримано фінальний булевий результат.

Для коректного оброблення складних виразів функція *evaluate_with_brackets* реалізує обчислення вкладених дужок. Вона ідентифікує найглибші вкладення дужок за допомогою *find_brackets*, обчислює вираз усередині них, а потім замінює весь блок дужок його обчисленим значенням, продовжуючи ітеративний процес до розкриття всіх дужок.

Програма генерує та аналізує таблиці істинності за допомогою функцій *print_table_header*, *print_table_row*, *analyze_results* та *print_truth_table*. Для створення всіх можливих комбінацій значень логічних змінних використовується функція *product* з бібліотеки *itertools*, що дозволяє перебрати 2^n варіантів, де n – кількість змінних у виразі. Форматований вивід забезпечується функціями *print_table_header* та *print_table_row*, які відповідають за створення зручної для сприйняття таблиці істинності.

Після генерації таблиці функція *analyze_results* виконує аналіз отриманих результатів, а саме встановлює, чи є логічний вираз *тавтологією* (усі значення істинні, *True*), *суперечністю* (усі значення хибні, *False*) або *контингентним*, тобто має як *True*, так і *False* у результатах.

Після запуску програми користувачу буде запропоновано ввести логічний вираз. Формат вводу передбачає використання змінних, позначених великими літерами латинського алфавіту (наприклад, *A*, *B*, *C*), а також логічних операторів у вигляді ключових слів великими літерами: *NOT* (логічне заперечення), *AND* (логічне «І»), *OR* (логічне «АБО»), *IMPL* (імплікація), *EQV* (еквівалентність) та *XOR* (виключне «АБО»).

Для уточнення порядку виконання операцій використовуються круглі дужки. Після введення виразу програма автоматично згенерує таблицю істинності, у якій буде показано всі можливі комбінації значень змінних та результат обчислення виразу. На основі цієї таблиці здійснюється класифікація: вираз визначається як тавтологія, суперечність або контингентний.

```

from itertools import product
def parse_expression(expression):
    """Розбиваємо вираз на слова (токени), додаючи пробіли навколо
    дужок"""
    return expression.replace('(', ' ( ').replace(')', ' ) ').split()
def is_operator(token):
    """Чи є токен оператором"""
    operators = ['NOT', 'AND', 'OR', 'IMPL', 'EQV',
    'XOR']
    for op in operators:
        if token == op:
            return True
    return False
def is_bracket(token):
    """Чи є токен дужкою"""
    brackets = ['(', ')']
    for bracket in brackets:
        if token == bracket:
            return True
    return False
def is_variable_new(token, variables):
    """Чи є токен новою змінною"""
    for var in variables:
        if token == var:
            return False
    return True
def get_variables(tokens):
    """Унікальні змінні у токенах, ігноруємо оператори та дужки"""
    variables = []
    for token in tokens:
        if is_operator(token) or is_bracket(token):
            continue
        if is_variable_new(token, variables):
            variables.append(token)
    return variables
def apply_operator(operator, value1, value2=None):
    """Результат логічної операції"""
    if operator == 'NOT':
        return not value1
    if operator == 'AND':
        return value1 and value2
    if operator == 'OR':
        return value1 or value2
    if operator == 'IMPL':
        return (not value1) or value2
    if operator == 'EQV':
        return value1 == value2
    if operator == 'XOR':
        return value1 != value2
    return None
def has_operators(tokens):
    """Чи є в токенах оператори"""
    operators = ['NOT', 'AND', 'OR', 'IMPL', 'EQV', 'XOR']

```



```

    for token in tokens:
        for op in operators:
            if token == op:
                return True
    return False
def find_highest_priority_operator(tokens):
    """Оператор із найвищим пріоритетом і його позиція"""
    operators = ['NOT', 'AND', 'OR', 'IMPL', 'EQV', 'XOR']
    priorities = [1, 2, 3, 4, 5, 6] # Менше число = вищий пріоритет
    min_priority = 999
    op_index = -1
    op_name = None
    for i in range(len(tokens)):
        for j in range(len(operators)):
            if tokens[i] == operators[j]:
                if priorities[j] < min_priority:
                    min_priority = priorities[j]
                    op_index = i
                    op_name = tokens[i]
    return op_name, op_index
def get_value(token, variable_names, variable_values):
    """Значення токена (змінна або вже обчислене значення)"""
    for i in range(len(variable_names)):
        if token == variable_names[i]:
            return variable_values[i]
    return token # Якщо це не змінна, повертаємо сам токен (вже обчислене значення)
def evaluate_operation(tokens, op_index, op_name, variable_names, variable_values):
    """Виконання однієї логічної операції, оновлення списку токенів"""
    tokens = tokens[:] # Копіюємо список
    if op_name == 'NOT':
        next_value = get_value(tokens[op_index + 1], variable_names, variable_values)
        result = apply_operator(op_name, next_value)
        tokens[op_index] = result
        tokens.pop(op_index + 1)
    else:
        left_value = get_value(tokens[op_index - 1], variable_names, variable_values)
        right_value = get_value(tokens[op_index + 1], variable_names, variable_values)
        result = apply_operator(op_name, left_value, right_value)
        tokens[op_index - 1] = result
        tokens.pop(op_index)
        tokens.pop(op_index)
    return tokens
def evaluate_simple(tokens, variable_names, variable_values):
    """Обчислення виразу без дужок, оператори за пріоритетами"""
    tokens = tokens[:] # Копіюємо список
    while has_operators(tokens):
        op_name, op_index = find_highest_priority_operator(tokens)

```

```

        tokens = evaluate_operation(tokens, op_index, op_name,
variable_names, variable_values)
    return tokens[0]
def find_brackets(tokens):
    """Позиції відкриваючої та закриваючої дужок"""
    close_index = -1
    for i in range(len(tokens)):
        if tokens[i] == ')':
            close_index = i
            break
    if close_index == -1:
        return -1, -1
    open_index = -1
    for i in range(close_index - 1, -1, -1):
        if tokens[i] == '(':
            open_index = i
            break
    return open_index, close_index
def evaluate_with_brackets(tokens, variable_names, variable_values):
    """Обчислюємо вираз із дужками"""
    tokens = tokens[:] # Копіюємо список
    while True:
        open_index, close_index = find_brackets(tokens)
        if open_index == -1:
            break
        sub_tokens = tokens[open_index + 1:close_index]
        result = evaluate_simple(sub_tokens, variable_names,
variable_values)
        tokens[open_index] = result
        for i in range(close_index, open_index, -1):
            tokens.pop(i)
    return evaluate_simple(tokens, variable_names, variable_values)
def print_table_header(variables):
    """Заголовок таблиці істинності"""
    header = ""
    for var in variables:
        header += var + " | "
    header += "Результат"
    print(header)
    print("-" * (6 * len(variables) + 10))
def print_table_row(values, result):
    """Один рядок таблиці істинності"""
    row = ""
    for value in values:
        if value:
            row += "1 | "
        else:
            row += "0 | "
    if result:
        row += "1"
    else:
        row += "0"
    print(row)

```

```

def analyze_results(results):
    """Аналіз результатів таблиці істинності"""
    all_true = True
    any_true = False
    for result in results:
        if not result:
            all_true = False
        if result:
            any_true = True
    if all_true:
        print("\nВираз завжди істинний (тавтологія)")
    elif not any_true:
        print("\nВираз завжди хибний (протиріччя)")
    else:
        print("\nВираз істинний у деяких випадках (контингентний)")
def print_truth_table(expression):
    """Таблиця істинності для виразу"""
    try:
        tokens = parse_expression(expression)
        variables = get_variables(tokens)
        print("Вираз:", expression)
        print("Змінні:", variables)
        print("\nТаблиця істинності:")
        print_table_header(variables)
        results = []
        for values in product([False, True], repeat=len(variables)):
            result = evaluate_with_brackets(tokens, variables,
list(values))
            results.append(result)
            print_table_row(values, result)
        analyze_results(results)
    except Exception as e:
        print(f"Помилка: {e}")
def main():
    """Запуск програми та діалог з користувачем"""
    print("Логічний аналізатор")
    print("Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D")
    print("Оператори: NOT, AND, OR, IMPL, EQV, XOR")
    print("Використовуйте великі літери для змінних (A, B, C тощо)")
    expression = input("\nВведіть ваш вираз: ")
    if expression == "":
        print("Вираз не може бути порожнім!")
        return
    print_truth_table(expression)
if __name__ == "__main__":
    main()

```

Перевіримо роботу програми на прикладах логічних виразів, які охоплюють тривіальні та нетривіальні випадки, а також ілюструють різні типи результатів (тавтологію, суперечність, контингентність):

- фундаментальний закон виключеного третього (рис. 2.1), незалежно від значення *A* вираз завжди буде істинним (тавтологія)

```

... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз: A OR (NOT A)
Вираз: A OR (NOT A)
Змінні: ['A']

Таблиця істинності:
A | Результат
-----
0 | 1
1 | 1

Вираз завжди істинний (тавтологія)

```

Рис. 2.1. Найпростіша тавтологія

– закон транзитивності імплікації (якщо A імплікує B , а B імплікує C , то A імплікує C). Цей вираз завжди істинний, що підтверджує його властивості у логіці (рис. 2.2).

```

... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз: ((A IMPL B) AND (B IMPL C)) IMPL (A IMPL C)
Вираз: ((A IMPL B) AND (B IMPL C)) IMPL (A IMPL C)
Змінні: ['A', 'B', 'C']

Таблиця істинності:
A | B | C | Результат
-----
0 | 0 | 0 | 1
0 | 0 | 1 | 1
0 | 1 | 0 | 1
0 | 1 | 1 | 1
1 | 0 | 0 | 1
1 | 0 | 1 | 1
1 | 1 | 0 | 1
1 | 1 | 1 | 1

Вираз завжди істинний (тавтологія)

```

Рис. 2.2. Нетривіальний приклад тавтології

– закон виключеного протиріччя (рис. 2.3), змінна не може бути одночасно істинною і хибною.

```
... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз: A AND (NOT A)
Вираз: A AND (NOT A)
Змінні: ['A']

Таблиця істинності:
A | Результат
-----
0 | 0
1 | 0

Вираз завжди хибний (протиріччя)
```

Рис. 2.3. Найпростіше протиріччя

– розширена форма тривіальної суперечності (демонструє, що будь-яке твердження (у даному випадку $A \text{ AND } B$) не може бути одночасно істинним і хибним) (рис. 2.4).

```
... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз: (A AND B) AND (NOT (A AND B))
Вираз: (A AND B) AND (NOT (A AND B))
Змінні: ['A', 'B']

Таблиця істинності:
A | B | Результат
-----
0 | 0 | 0
0 | 1 | 0
1 | 0 | 0
1 | 1 | 0

Вираз завжди хибний (протиріччя)
```

Рис. 2.4. Нетривіальний приклад протиріччя

– диз'юнкція двох змінних (рис. 2.5), результат залежить від вхідних значень.

```
... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз: A OR B
Вираз: A OR B
Змінні: ['A', 'B']

Таблиця істинності:
A | B | Результат
-----
0 | 0 | 0
0 | 1 | 1
1 | 0 | 1
1 | 1 | 1

Вираз істинний у деяких випадках (контингентний)
```

Рис. 2.5. Найпростіший контингентний вираз

– вираз, результат якого буде істинним у багатьох випадках, але хибним, наприклад, якщо A і B істинні, а C хибне (рис. 2.6).

```
... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз: (A AND B) IMPL (NOT C)
Вираз: (A AND B) IMPL (NOT C)
Змінні: ['A', 'B', 'C']

Таблиця істинності:
A | B | C | Результат
-----
0 | 0 | 0 | 1
0 | 0 | 1 | 1
0 | 1 | 0 | 1
0 | 1 | 1 | 1
1 | 0 | 0 | 1
1 | 0 | 1 | 1
1 | 1 | 0 | 1
1 | 1 | 1 | 0

Вираз істинний у деяких випадках (контингентний)
```

Рис. 2.6. Нетривіальний приклад контингентного виразу

– порожній рядок на вході програми (рис. 2.7), користувач випадково натиснути *Enter* без введення виразу.

```
... Логічний аналізатор
Введіть вираз, наприклад: (A OR (NOT B AND C)) IMPL D
Оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (A, B, C тощо)

Введіть ваш вираз:
Вираз не може бути порожнім!
```

Рис. 2.7. Коректна обробка порожнього рядку

Програма корисна для вивчення логіки, перевірки логічних виразів та аналізу їхніх властивостей, але для її вдосконалення можна додати більш детальні перевірки, щоб надавати користувачеві конкретніші та зрозуміліші повідомлення про тип допущеної помилки (наприклад, «незбалансовані дужки», «невідомий оператор»), що значно покращить зручність її використання.

Завдання 2. Написати програму, яка буде для ПФ будувати її ДДНФ.

Це завдання є логічним продовженням та розширенням функціоналу програми, розробленої для Завдання 1. Метою є не лише аналіз логічних виразів через побудову таблиць істинності, але й автоматична генерація їхньої досконалої диз'юнктивної нормальної форми (ДДНФ).

Для реалізації ДДНФ, базовий програмний код із **Завдання 1** було доопрацьовано та вдосконалено. Зокрема, було підвищено надійність оброблення виразів, додано більш детальні перевірки на некоректний синтаксис (наприклад, незбалансовані або порожні дужки, недостатність операндів) та забезпечено консистентне сортування змінних. Ці поліпшення важливі для коректної та уніфікованої побудови ДДНФ, оскільки вона спирається на точні та передбачувані результати таблиці істинності.

Ключовим доповненням цієї програми є її здатність до побудови ДДНФ. Для цього введено нову функцію *build_ddnf*, яка аналізує результати, отримані з таблиці істинності, ідентифікуючи всі випадки, коли кінцевий логічний вираз є істинним (*True*). На основі цих «істинних» рядків формуються окремі кон'юнктивні члени. Кожний такий член є кон'юнкцією змінних, де істинні змінні входять без заперечення, а хибні із запереченням (наприклад, *(NOT A AND B)*).

Усі отримані кон'юнкції потім об'єднуються великою диз'юнкцією (*OR*), що й утворює остаточну ДДНФ. Важливо зазначити, що для виразів, які є тавтологіями (завжди істинні), ДДНФ не будується, оскільки такий вираз еквівалентний просто *TRUE*. Аналогічно, якщо вираз є протиріччям (завжди

хибний), ДДНФ також не формується через відсутність «істинних» рядків, що еквівалентно *FALSE*.

Окрім додавання функціоналу ДДНФ, програма отримала ряд покращень, спрямованих на підвищення її коректності та надійності. Зокрема, було впроваджено сортування змінних у функції *get_variables*, що гарантує послідовний та передбачуваний порядок їх відображення як у таблицях істинності, так і під час побудови ДДНФ.

Функція *get_value* була вдосконалена для більш надійного розпізнавання всіх типів логічних значень, з огляду й на проміжні булеві результати. Додаткові перевірки операндів у *evaluate_operation* тепер запобігають помилкам у випадках, коли логічні оператори використовуються без достатньої кількості операндів. Валідація дужок, реалізована у функціях *find_brackets* та *evaluate_with_brackets*, ефективно запобігає обчисленню незбалансованих або порожніх дужок, підвищуючи стійкість програми до некоректного вводу.

Було оптимізовано взаємодію з користувачем: ввід автоматично перетворюється на верхній регістр та очищається від зайвих пробілів, забезпечуючи більшу гнучкість та меншу схильність до помилок форматування.

```
from itertools import product
def parse_expression(expression):
    """Розбиваємо вираз на слова (токени), додаючи пробіли навколо
    дужок"""
    return expression.replace('(', ' ( ').replace(')', ' ) ').split()
def is_operator(token):
    """Чи є токен логічним оператором"""
    operators = ['NOT', 'AND', 'OR', 'IMPL', 'EQV', 'XOR']
    return token in operators
def is_bracket(token):
    """Чи є токен дужкою"""
    brackets = ['(', ')']
    return token in brackets
def is_variable_new(token, variables):
    """Чи є токен новою змінною"""
    return token not in variables
def get_variables(tokens):
    """Унікальні змінні з токенів, змінні сортуються"""
    variables = []
    for token in tokens:
        if is_operator(token) or is_bracket(token):
            continue
        if is_variable_new(token, variables):
            variables.append(token)
    variables.sort()
    return variables
def apply_operator(operator, value1, value2=None):
    """Застосовуємо логічну операцію"""
    if operator == 'NOT':
```



```

        return not value1
    if operator == 'AND':
        return value1 and value2
    if operator == 'OR':
        return value1 or value2
    if operator == 'IMPL':
        return (not value1) or value2
    if operator == 'EQV':
        return value1 == value2
    if operator == 'XOR':
        return value1 != value2
    return None
def has_operators(tokens):
    """Чи містить список tokenів будь-які логічні оператори"""
    operators = ['NOT', 'AND', 'OR', 'IMPL', 'EQV', 'XOR']
    for token in tokens:
        if token in operators:
            return True
    return False
def find_highest_priority_operator(tokens):
    """Знаходимо оператор із найвищим пріоритетом"""
    operator_priority_order = ['NOT', 'AND', 'OR', 'IMPL', 'EQV',
    'XOR']
    min_priority_value = 999
    op_index = -1
    op_name = None
    for i in range(len(tokens)):
        current_token = tokens[i]
        for priority_rank, op_in_order in
enumerate(operator_priority_order):
            if current_token == op_in_order:
                if priority_rank < min_priority_value:
                    min_priority_value = priority_rank
                    op_index = i
                    op_name = current_token
                break
    return op_name, op_index
def get_value(token, variable_names, variable_values):
    """Булеве значення токена"""
    # Якщо токен - це змінна
    for i in range(len(variable_names)):
        if token == variable_names[i]:
            return variable_values[i]
    # Якщо токен - вже обчислене значення
    if token is True or token is False:
        return token
    # Якщо токен - рядкова константа
    if isinstance(token, str):
        if token.upper() == 'TRUE':
            return True
        if token.upper() == 'FALSE':
            return False

```

```

        raise ValueError(f"Невідома змінна або некоректний токен:
'{token}'")
def evaluate_operation(tokens, op_index, op_name, variable_names,
variable_values):
    """Виконує одну логічну операцію"""
    tokens = list(tokens)
    if op_name == 'NOT':
        if op_index + 1 >= len(tokens):
            raise ValueError(f"Оператор 'NOT' без операнда.")
        next_value = get_value(tokens[op_index + 1], variable_names,
variable_values)
        result = apply_operator(op_name, next_value)
        tokens[op_index] = result
        tokens.pop(op_index + 1)
    else:
        if op_index - 1 < 0 or op_index + 1 >= len(tokens):
            raise ValueError(f"Оператор '{op_name}' без достатніх
операндів.")
        left_value = get_value(tokens[op_index - 1], variable_names,
variable_values)
        right_value = get_value(tokens[op_index + 1],
variable_names, variable_values)
        result = apply_operator(op_name, left_value, right_value)
        tokens[op_index - 1] = result
        tokens.pop(op_index)
        tokens.pop(op_index)
    return tokens
def evaluate_simple(tokens, variable_names, variable_values):
    """Обчислюємо вираз без дужок"""
    tokens = list(tokens)
    while has_operators(tokens) or len(tokens) > 1:
        if has_operators(tokens):
            op_name, op_index =
find_highest_priority_operator(tokens)
            tokens = evaluate_operation(tokens, op_index, op_name,
variable_names, variable_values)
        elif len(tokens) == 1:
            break
        else:
            raise ValueError("Помилка обчислення.")
    return get_value(tokens[0], variable_names, variable_values)
def find_brackets(tokens):
    """Пошук позицій дужок"""
    close_index = -1
    for i in range(len(tokens)):
        if tokens[i] == ')':
            close_index = i
            break
    if close_index == -1:
        return -1, -1
    open_index = -1
    for i in range(close_index - 1, -1, -1):
        if tokens[i] == '(':

```

```

        open_index = i
        break
    if open_index == -1:
        raise ValueError("Незбалансовані дужки.")
    return open_index, close_index
def evaluate_with_brackets(tokens, variable_names, variable_values):
    """Обчислюємо вираз із дужками"""
    tokens = list(tokens)
    while True:
        open_index, close_index = find_brackets(tokens)
        if open_index == -1:
            break
        sub_tokens = tokens[open_index + 1:close_index]
        if not sub_tokens:
            raise ValueError("Порожні дужки.")
        result = evaluate_simple(sub_tokens, variable_names,
variable_values)
        tokens[open_index] = result
        for i in range(close_index, open_index, -1):
            tokens.pop(i)
        return evaluate_simple(tokens, variable_names, variable_values)
def print_table_header(variables):
    header = "".join([f"{var:^5} | " for var in variables]) +
"Результат"
    print(header)
    print("-" * (6 * len(variables) + 10))
def print_table_row(values, result):
    row = "".join([f"{v if v else '0':^5} | " for v in values]) +
f"{1 if result else '0'}"
    print(row)
def analyze_results(results):
    if all(results):
        print("\nВираз завжди істинний (тавтологія)")
        return "Tautology"
    elif not any(results):
        print("\nВираз завжди хибний (протиріччя)")
        return "Contradiction"
    else:
        print("\nВираз істинний у деяких випадках (контингентний)")
        return "Contingent"
def build_ddnf(variables, truth_table_rows):
    ddnf_terms = []
    true_rows = [row for row in truth_table_rows if row[1] is True]
    if not true_rows:
        return "ДДНФ не існує (протиріччя)."
    for values, result in true_rows:
        term = []
        for i, var in enumerate(variables):
            term.append(var if values[i] else f"NOT {var}")
        ddnf_terms.append(f"({ ' AND '.join(term) })")
    return ' OR '.join(ddnf_terms)
def print_truth_table(expression):
    try:

```

```

tokens = parse_expression(expression)
variables = get_variables(tokens)
print(f"\n--- Таблиця істинності ---")
if not variables:
    res = evaluate_with_brackets(tokens, [], [])
    print(f"Результат: {'1' if res else '0'}")
    return
print_table_header(variables)
rows_data, results = [], []
for v_tuple in product([False, True],
repeat=len(variables)):
    res = evaluate_with_brackets(tokens, variables,
list(v_tuple))
    rows_data.append((v_tuple, res))
    results.append(res)
    print_table_row(v_tuple, res)
print("-" * (6 * len(variables) + 10))
expr_type = analyze_results(results)
if expr_type == "Contingent":
    print(f"\nДДНФ:\n{build_ddnf(variables, rows_data)}")
except Exception as e:
    print(f"Помилка: {e}")

def main():
    # Текст із заголовка програми на картинках
    print("--- Логічний Аналізатор, побудовник таблиць істинності та
ДДНФ ---")
    print("Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV,
XOR")
    print("Використовуйте великі літери для змінних (наприклад, A,
B, C).")
    print("Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D")
    print("Для виходу з програми просто натисніть Enter без введення
виразу.")
    while True:
        # Текст запиту на картинках
        expr = input("\nВведіть ваш логічний вираз:
").strip().upper()

        # Перевірка на порожній рядок
        if not expr:
            break

        print_truth_table(expr)
if __name__ == "__main__":
    main()

```

Щоб продемонструвати можливості розробленої програми та перевірити її коректність, розглянемо кілька логічних виразів. Ці приклади охоплюють різні сценарії: від простих законів логіки до складних контингентних виразів, а також випадки, де ДДНФ будується або не застосовується.

– коректність обробки простих виразів з одним оператором та одним операндом (програма розпізнає тавтології і вказує, що для них ДДНФ еквівалентна *TRUE*) показано на рис. 2.8:

```
*** --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: A OR NOT A

--- Таблиця істинності ---
  A   | Результат
-----
  0   | 1
  1   | 1
-----

Вираз завжди істинний (тавтологія)

Введіть ваш логічний вираз: 
```

Рис. 2.8. ДДНФ тривіальної тавтології (закон виключеного третього)

– обробка базових протиріч (програма правильно визначає протиріччя і не будує ДДНФ, оскільки вона еквівалентна *FALSE*) (рис. 2.9):

```
*** --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: A AND NOT A

--- Таблиця істинності ---
  A   | Результат
-----
  0   | 0
  1   | 0
-----

Вираз завжди хибний (протиріччя)

Введіть ваш логічний вираз: 
```

Рис. 2.9. ДДНФ тривіального протиріччя (закон виключеного протиріччя)

– робота з кількома змінними та побудову ДДНФ для простого, але нетривіального випадку (диз'юнкція двох змінних), таблиця істинності буде мати чотири рядки і ДДНФ буде побудована для трьох з них (рис. 2.10);

```
... --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: A OR B

--- Таблиця істинності ---
  A  |  B  | Результат
-----
  0  |  0  |  0
  0  |  1  |  1
  1  |  0  |  1
  1  |  1  |  1
-----

Вираз істинний у деяких випадках (контингентний)

ДДНФ:
(NOT A AND B) OR (A AND NOT B) OR (A AND B)

Введіть ваш логічний вираз: 
```

Рис. 2.10. ДДНФ базового контингентний виразу (диз'юнкція двох змінних)

– коректність обробки дужок та пріоритетів операторів у складнішому виразі (програма буде ДДНФ для виразу з трьома змінними, де комбінації істинності та хибності більш різноманітні) показано на рис. 2.11;

```
... --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: (A AND B) OR (NOT C)

--- Таблиця істинності ---
  A  |  B  |  C  | Результат
-----
  0  |  0  |  0  |  1
  0  |  0  |  1  |  0
  0  |  1  |  0  |  1
  0  |  1  |  1  |  0
  1  |  0  |  0  |  1
  1  |  0  |  1  |  0
  1  |  1  |  0  |  1
  1  |  1  |  1  |  1
-----

Вираз істинний у деяких випадках (контингентний)

ДДНФ:
(NOT A AND NOT B AND NOT C) OR (NOT A AND B AND NOT C) OR (A AND NOT B AND NOT C) OR (A AND B AND NOT C) OR (A AND B AND C)

Введіть ваш логічний вираз: 
```

Рис. 2.11. ДДНФ нетривіального контингентного виразу з дужками

– робота з операторами імплікації та еквівалентності (демонструє один з ключових законів логіки – еквівалентність імплікації), показує, що навіть складні вирази можуть бути тавтологіями і для них ДДНФ не застосовується (рис. 2.12);

```

*** --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: (A IMPL B) EQV (NOT A OR B)

--- Таблиця істинності ---
  A  |  B  | Результат
-----
  0  |  0  |  1
  0  |  1  |  1
  1  |  0  |  1
  1  |  1  |  1
-----

Вираз завжди істинний (тавтологія)

Введіть ваш логічний вираз: 

```

Рис. 2.12. ДДНФ нетривіального прикладу з очікуваним результатом тавтологія

– перевірку, як програма обробляє логічні константи (TRUE, FALSE) усередині виразу зі змінними (здатність спрощувати такі вирази та коректно будувати ДДНФ), показано на рис. 2.13;

```

*** --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: TRUE AND (A OR FALSE)

--- Таблиця істинності ---
Введений вираз: 'TRUE AND (A OR FALSE)'
Виявлені змінні (у порядку обробки): ['A', 'FALSE', 'TRUE']
  A  | FALSE | TRUE | Результат
-----
  0  |  0    |  0   |  0
  0  |  0    |  1   |  0
  0  |  1    |  0   |  0
  0  |  1    |  1   |  1
  1  |  0    |  0   |  0
  1  |  0    |  1   |  1
  1  |  1    |  0   |  0
  1  |  1    |  1   |  1
-----

Вираз істинний у деяких випадках (контингентний)

Досконала Диз'юнктивна Нормальна Форма (ДДНФ):
(NOT A AND FALSE AND TRUE) OR (A AND NOT FALSE AND TRUE) OR (A AND FALSE AND TRUE)

Введіть ваш логічний вираз: 

```

Рис. 2.13. ДДНФ складного синтаксису та констант

– стійкість програми до некоректного вводу та її здатність надавати інформативні повідомлення про синтаксичні помилки (рис. 2.14–2.17):

Для перевірки неповного введення виразу було задано логічний вираз з недостатньою кількістю операндів (наприклад, вираз, у якому бінарний оператор не має другого операнда). Програма коректно виявила помилку та вивела зрозуміле повідомлення про відсутність необхідного операнда (рис. 2.14).

```
*** --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: A AND

--- Таблиця істинності ---
  A   | Результат
-----
Помилка: Оператор 'AND' без достатніх операндів.

Введіть ваш логічний вираз: 
```

Рис. 2.14. Приклад перевірки обробки помилок вводу
(недостатньо операндів)

У тесті на рисунку 2.15 було введено вираз, який містить синтаксично некоректні елементи (наприклад, два оператори поспіль або неочікуваний символ). Програма успішно розпізнала некоректність виразу та надала користувачеві інформативне повідомлення про виявлену синтаксичну помилку з вказівкою на її характер.

```
*** --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: NOT

--- Таблиця істинності ---
Помилка: Оператор 'NOT' без операнда.

Введіть ваш логічний вираз: 
```

Рис. 2.15. Приклад перевірки обробки помилок вводу
(некоректний вираз)

Для тестування обробки дужок було введено логічний вираз із незбалансованою кількістю відкриваючих та закриваючих дужок. Програма правильно виявила порушення балансу дужок і повідомила користувача про відповідну синтаксичну помилку з чітким описом проблеми (рис. 2.16).

```
... --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: A ((B)

--- Таблиця істинності ---
  A   |   B   | Результат
-----
Помилка: Помилка обчислення.

Введіть ваш логічний вираз: 
```

Рис. 2.16. Приклад перевірки обробки помилок вводу (незбалансовані дужки)

У випадку на рисунку 2.17 було перевірено реакцію на порожні дужки (наприклад, вираз типу «()» або «A ∧ ()»). Програма коректно ідентифікувала порожні дужки як недопустимий елемент і вивела зрозуміле повідомлення про помилку, пояснюючи, що дужки не можуть бути порожніми.

```
... --- Логічний Аналізатор, побудовник таблиць істинності та ДДНФ ---
Підтримувані логічні оператори: NOT, AND, OR, IMPL, EQV, XOR
Використовуйте великі літери для змінних (наприклад, A, B, C).
Приклад коректного виразу: (A OR (NOT B AND C)) IMPL D
Для виходу з програми просто натисніть Enter без введення виразу.

Введіть ваш логічний вираз: ( )

--- Таблиця істинності ---
Помилка: Порожні дужки.

Введіть ваш логічний вираз: 
```

Рис. 2.17. Приклад перевірки обробки помилок вводу (порожні дужки)

Ця програма є потужним інструментом для вивчення логіки, перевірки логічних виразів та аналізу їхніх властивостей, з огляду й на автоматизовану побудову ДДНФ.

3. БУЛЕВІ ФУНКЦІЇ ТА ЇХ ВИКОРИСТАННЯ

3.1. Класи булевих функцій. Повні системи. Теорема Яблонського

Функцією алгебри логіки, або **булевою** функцією (БФ), називається будь-яка ***n*-місна** функція, що відображає набір булевих змінних з $\{0,1\}$ у булеве значення $\{0,1\}$. У рамках математичних структур булеві функції утворюють **булеве кільце** F_n – функцію $F(x_1, \dots, x_n)$ від n змінних $x_1, \dots, x_n \in D$ зі значеннями з D , де $D = \{0,1\}$ – множина допустимих значень змінних.

У логіці висловлювань, як ми бачили раніше, кожна ПФ $A(x_1, x_2, \dots, x_n)$ задає (реалізує) відповідну булеву функцію $f(x_1, \dots, x_n)$, що називається **приєднаною** функцією ПФ.

Позначимо через $\tilde{X}$ чи $\tilde{x}^n = (x_1, x_2, \dots, x_n)$ вектор змінних $x_1, \dots, x_n$, що набуває значення $\alpha_1, \alpha_2, \dots, \alpha_n$ з множини $\{0, 1\}$, і назвемо його булевым вектором (або булевым набором) довжини n . Множина всіх булевих векторів довжини n утворює *одиначний n -вимірний куб* і позначається через B^n . Самі вектори $\tilde{x}^n$ утворюють вершини куба B^n . Кожний булевий вектор називається **вершиною куба**.

Множину всіх вершин куба B^n , що мають вагу K_n , іменують *k -м шаром* куба B^n і позначають через B_k^n . Вочевидь, булева функція $f(\tilde{x}^n) = f(x_1, \dots, x_n)$ визначена на множині B^n і набуває значення з $\{0, 1\}$; область визначення $f(\tilde{x}^n)$ складається з 2^n двійкових наборів значень аргументів, а число всіх БФ від n -змінних дорівнює 2^{2^n} . Множину всіх БФ позначимо через G_1 .

Надалі БФ розглядаються з точністю до фіктивних змінних. Зазначимо, якщо булеві функції, які тотожно дорівнюють 0 або 1 (тобто булеві константи), то вони **не мають істотних змінних** і трактуються як функції від **порожньої множини змінних**.

Якщо задана кінцева система БФ складається із $G_1: \{f_1, \dots, f_s\}$, $S \geq 1$, де S – кількість булевих функцій в системі, то, оскільки БФ подано з точністю до фіктивних змінних, можна вважати, що усі вони залежать від тих самих змінних $x_1, \dots, x_n$, тобто мають вигляд $f_1(x_1, \dots, x_n), \dots, f_s(x_1, \dots, x_n)$.

Звідси впливає коректність задач мінімізації БФ, наприклад у класі ДНФ чи КНФ. У табл. 3.1 наведено позначення, назви і задання елементарних булевих функцій.

У термінах мови першого порядку кожній n -місній БФ $f \in G \subset G_1$ зіставимо функціональний символ $F \in F_n$ мови Ω (1.4). Нехай x_0, x_1, x_2 – зліченна множина змінних з множини Ω .

Визначимо поняття терма в алгебрі логіки індуктивно:

- а) змінна є терм T ;
- б) якщо F – n -місна БФ із G і $T_1, \dots, T_n$ – терми, то $F(T_1, \dots, T_n)$ є терм;
- в) інших термів немає.

Приклади термів: $x_0, \bar{x}_5, y, \bar{z}_1 \vee \bar{z}_2, x_1 \rightarrow x_2 \sim x_1, (a \downarrow b)$.

Зіставимо кожну змінну $x_0, x_1, x_2, \dots, x_n$ та її значення з множини $\{0, 1\}$ і визначимо значення терма T при наведених нижче значеннях змінних:

а) якщо T – змінна, то значення терма збігається зі значенням цієї змінної;

б) якщо $T = F(T_1, \dots, T_n)$, а значення $T_1, \dots, T_n$ суть $\alpha_1, \dots, \alpha_n$ відповідно, то значення терма $T \in F(\alpha_1, \dots, \alpha_n)$.

Наприклад,

$$T_1 = x = 0, T_2 = y = 1, T = (T_1 \rightarrow T_2) \wedge T_1 = (0 \rightarrow 1) \wedge 1 = 1.$$

Затверджуємо, що n -місну БФ можна подати термом T , якщо всі змінні терма T містяться серед $x_1, \dots, x_n$ і для будь-яких значень змінних $x_1, \dots, x_n$ значення терма T збігається зі значенням $g(x_1, \dots, x_n)$.

Таблиця 3.1

Елементарні булеві операції над двома змінними

Функція	x_1	0	0	1	1	Назва БФ
	x_2	0	1	0	1	
$f_1(x_1) = 0$		0		0		тотожний нуль – константа
$f_2(x_2) = 1$		1		1		тотожна одиниця – константа
$f_3(x_1) = x_1$		0		1		тотожна функція x_1
$f_4(x_1) = \bar{x}_1$		1		0		заперечення x_1
$f_5(x_1, x_2) = x_1 \wedge x_2$		0	0	0	1	кон'юнкція – логічне множення
$f_6(x_1, x_2) = x_1 \vee x_2$		0	1	1	1	диз'юнкція – логічне додавання
$f_7(x_1, x_2) = x_1 \oplus x_2$		0	1	1	0	сума по mod 2 – заперечення рівнозначності
$f_8(x_1, x_2) = x_1 \sim x_2$		1	0	0	1	рівнозначність (еквівалентність)
$f_9(x_1, x_2) = x_1 \rightarrow x_2$		1	1	0	1	імплікація
$f_{10}(x_1, x_2) = x_1 x_2$		1	1	1	0	функція Шеффера («штрих Шеффера»)
$f_{11}(x_1, x_2) = x_1 \downarrow x_2$		1	0	0	0	функція Пірса («стрілка Пірса»)

Наприклад, БФ $(x \rightarrow y) \wedge \bar{x}$ можна подати термом $T = (T_1 \rightarrow T_2) \wedge \bar{T}_1$.

Тоді БФ – g є суперпозицією функцій $f_1, \dots, f_n$, якщо g можна подати як терм, усі функціональні символи якого містяться серед $F_1, \dots, F_n$, зіставлених з $f_1, \dots, f_n$.

Наприклад, БФ вигляду $x \wedge y \vee \bar{x} \wedge \bar{y}$ – суперпозиція БФ $x \wedge y$, $x \vee y$, $\bar{x}$. Система G булевих функцій називається повною, якщо будь-яка БФ із G_1 є суперпозицією функцій з G .

Системи БФ $\{xy, x \vee y, \bar{x}\}$, $\{\bar{x}, xy\}$, $\{x \downarrow y\}$, $\{x | y\}$ є повними. Система БФ G називається **незалежною**, якщо ніяку функцію $f \in G$ не можна подати як суперпозицію функцій з $G \setminus \{f\}$.

Системи БФ $\{\neg, \sim\}$, $\{\neg, \oplus\}$, $\{\sim, \oplus\}$, $\{\sim, \vee\}$ незалежні.

Клас функцій K називається **замкнутим**, якщо разом з будь-якими функціями він містить також усі їхні суперпозиції. У множині БФ G_1 містяться такі найважливіші замкнуті класи функцій:

G_2 – **клас функцій, що зберігають 1**, тобто задовольняють умову $f(1, 1, \dots, 1) = 1$;

G_3 – **клас функцій, що зберігають 0**, тобто задовольняють умову $f(0, 0, \dots, 0) = 0$;

L – **клас лінійних функцій** вигляду $x_1 \oplus x_2 \oplus \dots \oplus x_{n+\varepsilon}$, $\varepsilon \in \{0, 1\}$, тобто поліном за mod2: $\bigoplus_{i_1 \dots i_s} a_{i_1 \dots i_s} x_{i_1 \dots i_s}$ (поліном Жегалкіна);

D – **клас самодвоїстих функцій**, тобто таких, що задовольняють умову $f(x_1, \dots, x_n) = \bar{f}(\bar{x}_1, \dots, \bar{x}_n)$;

M – **клас монотонних функцій**, тобто задовольняють умову $x_1 \leq y_1, x_2 \leq y_2, \dots, x_n \leq y_n \Rightarrow f(x_1, \dots, x_n) \leq f(y_1, \dots, y_n)$.

Елементарна кон'юнкція (ЕК), яка не має заперечень змінних, є монотонною.

Можна показати, що система $\{\wedge, \rightarrow\}$ є базисом для G_2 , $\{\wedge, \oplus\}$ – базисом для G_3 , $\{0, \sim\}$ – базисом для L , $\{\neg, xy \oplus xz \oplus yz\}$ – базисом для D , $\{\vee, \wedge, 0, 1\}$ – базисом для M .

Теорема Поста-Яблонського про функціональну повноту. Для того щоб система булевих функцій G була повною, необхідно і достатньо, щоб вона *цілком не містилася в жодному з п'яти замкнутих класів* G_2 , G_3 , L , D і M .

Приклад 3.1

Система БФ $\{f_1 = x_1 x_2, f_2 = 0, f_3 = 1, f_4 = x_1 \oplus x_2 \oplus x_3\}$ є повною, оскільки $f_3 \notin G_3$, $f_2 \notin G_2$, $f_2 \notin D$, $f_4 \notin M$, $f_1 \notin L$.

Для БФ у булевій алгебрі справедливі усі раніше доведені теореми про розклад за змінними, про подання БФ у ДДНФ, ДНФ, ДКНФ, КНФ, ДІНФ, ІНФ та ін., використовуються також аналогічні поняття рівносильних і двоїстих БФ. Компоненти $x_{i_k}^{\sigma_k}$, $\sigma_k \in \{0, 1\}$, $i_k \in \{1, 2, \dots, n\}$, з яких складаються елементарні диз'юнкції (ЕД) та елементарні кон'юнкції (ЕК), позначають літерами.

Число літер у ЕК і ЕД називається **рангом** ЕК або ЕД. Константа 1 вважається ЕК нульового рангу, константа 0 – ЕД нульового рангу.

Число ЕК (ЕД), що містяться у ДНФ (у КНФ), називається **довжиною** ДНФ (довжиною КНФ). Сума рангів ЕК (ЕД) ДНФ (КНФ) називається **складністю** ДНФ (складністю КНФ).

Для перетворень БФ корисні такі рівносильності, у яких символ « $\circ$ » є умовним загальним позначенням зв'язок ($\wedge$, $\vee$, $\oplus$, $\sim$, $|$, $\downarrow$):

- 1) $x \circ y = y \circ x$ (комутативність для зв'язок ($\wedge$, $\vee$, $\oplus$, $\sim$, $|$, $\downarrow$);
- 2) $(x \circ y) \circ z = x \circ (y \circ z)$ (асоціативність для $\wedge$, $\vee$, $\oplus$, $\sim$);
- 3) $\overline{xy} = \bar{x} \vee \bar{y}$, $\overline{x \vee y} = \bar{x} \bar{y}$ (правила де Моргана);
- 4) $x \vee xy = x$, $x(x \vee y) = x$ (правила поглинання);
- 5) $y \vee y = y$, $y(x \vee \bar{x}) = y \cdot 1 = y$ (правила склеювання);
- 6) $(x \vee y) \cdot \bar{x} \vee y = y \vee (x \cdot \bar{x}) = y \vee 0 = y$ (добуток суми);
- 7) $x_1 x_2 \vee \bar{x}_1 x_3 = x_1 x_2 \vee x_1 x_2 x_3 \vee \bar{x}_1 x_3 \vee \bar{x}_1 x_2 x_3 = x_1 x_2 \vee \bar{x}_1 x_3 \vee x_1 x_2 x_3 \vee \bar{x}_1 x_2 x_3 = x_1 x_2 \vee \bar{x}_1 x_3 \vee x_2 x_3 (x_1 \vee \bar{x}_1) = x_1 x_2 \vee \bar{x}_1 x_3 \vee x_2 x_3$ (правило узагальненого склеювання);

8) за принципом подвійності:

$$\begin{aligned} (x \vee xy) &= (x \vee x)(x \vee y) = x(x \vee y) = x, \\ x(x \vee y) &= xx \vee xy = x \vee xy = x; \\ \overline{xy} &= \bar{x} \vee \bar{y}, \quad \overline{x \vee y} = \bar{x} \bar{y} \end{aligned}$$

9) $x(y \oplus z) = xy \oplus xz$ (дистрибутивність $\wedge$ відносно $\oplus$);

10) $0 = x \cdot \bar{x} = x \cdot 0 = x \oplus x$;

11) $1 = x \vee \bar{x} = x \vee 1 = x \oplus \bar{x}$;

12) $x = \bar{\bar{x}} = x \cdot x = x \vee x = x \cdot 1 = x \vee 0$;

13) $\bar{x} = x \oplus 1 = x \rightarrow 0 = x/x$;

14) $x \oplus y = x\bar{y} \vee \bar{x}y$;

15) $x \sim y = \bar{x}\bar{y} \vee xy = (x \oplus y) \oplus 1$;

16) $x \rightarrow y = \bar{x} \vee y = ((xy) \oplus x) \oplus 1$;

17) $x / y = \bar{x} \vee \bar{y}$.

Диз'юнктивний критерій поглинання (ДКП)

Для будь-яких ПФ $A, B_1, B_2, \dots, B_m$; якщо $(A \supset B_1 \vee B_2, \dots, B_m)$, то $A \vee B_1 \vee \dots \vee B_m \sim B_1 \vee B_2 \vee \dots \vee B_m$.

Кон'юнктивний критерій поглинання (ККП)

Для будь-яких ПФ $A, B_1, B_2, \dots, B_m$, якщо $(B_1 \wedge B_2 \wedge \dots \wedge B_m \supset A)$, то $A \wedge B_1 \wedge B_2 \wedge \dots \wedge B_m \sim B_1 \wedge B_2 \wedge \dots \wedge B_m$.

Приклад 3.2

Спростимо ДНФ:

$$\bar{x}\bar{y} \vee \bar{y}z \vee \bar{y}z \vee xz = \bar{x}\bar{y} \vee \bar{y}z \vee xz = \bar{x}\bar{y} \vee xz$$

3.2. Подання БФ у вигляді полінома Жегалкіна

Оскільки число поліномів у класі L дорівнює числу всіх БФ у G_1 , а саме 2^{2^n} , маємо однозначність подання БФ за допомогою полінома Жегалкіна.

Наприклад, $f(\tilde{x}^2) = x_1 \rightarrow x_2 = a_0 \oplus a_1 x_1 \oplus a_2 x_2 \oplus a_3 x_1 x_2$;

$$f(0,0) = 1 = a_0 \oplus a_1 \cdot 0 \oplus a_2 \cdot 0 \oplus a_3 \cdot 0 \cdot 0, \quad a_0 = 1;$$

$$f(0,1) = 1 = a_0 \oplus a_1 \cdot 0 \oplus a_2 \cdot 1 \oplus a_3 \cdot 0 \cdot 1, \quad a_2 = 0;$$

$$f(1,0) = 0 = a_0 \oplus a_1 \cdot 1 \oplus a_2 \cdot 0 \oplus a_3 \cdot 1 \cdot 0, \quad a_1 = 1;$$

$$f(1,1) = 1 = a_0 \oplus a_1 \cdot 1 \oplus a_2 \cdot 1 \oplus a_3 \cdot 1 \cdot 1, \quad a_3 = 1.$$

Отже, $x_1 \rightarrow x_2 = 1 \oplus x_1 \oplus x_1 x_2$. Цей результат можна отримати іншим методом. Вихідну БФ подамо в базисі $\{\wedge, \vee\}$, для чого замінімо скрізь $\bar{x}$ на $x \oplus 1$. Розкриємо дужки, скориставшись дистрибутивним законом: $(x \oplus y)z = xz \oplus yz$, а також зведемо подібні члени:

$$x_1 \rightarrow x_2 = \overline{(x_1 \wedge x_2)} = (x_1(x_2 \oplus 1)) \oplus 1 = 1 \oplus x_1 \oplus x_1 x_2.$$

3.3. Застосування булевих функцій

Булеві функції знайшли фундаментальне використання в теорії кінцевих автоматів, тобто дискретних пристроїв. Оскільки залежності між вхідними і вихідними сигналами дискретного пристрою описуються булевими функціями, аналіз і синтез кінцевих автоматів ґрунтуються на використанні алгебри логіки.

Як приклад розглянемо аналіз і синтез найпростішого кінцевого автомата без пам'яті – двійкового однорозрядного суматора.

Аналіз синтезованого пристрою полягає в побудові табл. 3.2, де наведено додавання двійкових однорозрядних чисел x_1 і x_2 , а також табл. 3.3, що описує залежність вихідних сигналів двійкового однорозрядного суматора Y_1, Y_2 від вхідних сигналів X_1, X_2 .

Останній має складатися з двох дискретних пристроїв без пам'яті. На виході Y_1 першого буде з'являтися сигнал старшого розряду суми, а на виході Y_2 другого – молодший розряд, оскільки у двійковій системі $1 + 1 = 10$.

Синтез двійкового суматора зводиться до визначення БФ, наприклад у формі ДНФ, яка описує залежність виходів Y_1 і Y_2 від входів X_1, X_2 (табл.3.2):

$$Y_1 = x_1 \wedge x_2;$$

$$Y_2 = \bar{x}_1 \wedge x_2 \vee x_1 \wedge \bar{x}_2 = x_1 \oplus x_2,$$

та побудові функціональної схеми суматора (рис.3.1).

У теорії релейно-контактних ланцюгів булевими функціями описуються так звані функції провідності ланцюгів, що характеризують умови, за яких у цих ланцюгах протікають струми.

Таблиця 3.2

x_1	x_2	x_1+x_2
0	0	0
0	1	1
1	0	1
1	1	10

Таблиця 3.3

x_1	x_2	y_1	y_2
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

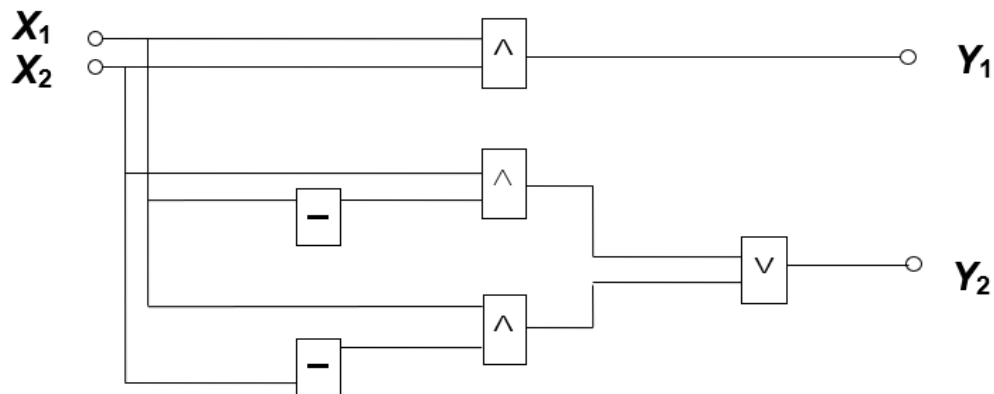


Рис. 3.1. Схема суматора

Перемикач (або контакт реле) у ланцюзі, через який проходить струм, позначається x_i , у протилежному випадку – $\bar{x}_i$. Наприклад, функція провідності ланцюга, зображеного на рис. 3.2, задається БФ $x_3(x_1 \vee \bar{x}_2) \vee x_1\bar{x}_3 \vee \bar{x}_2 \bar{x}_3$.

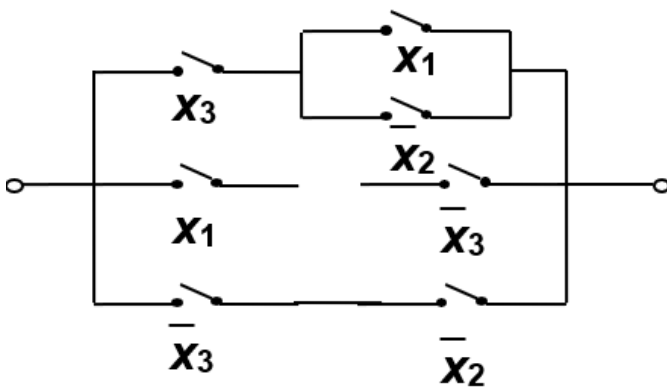


Рис. 3.2. Схема до спрощення

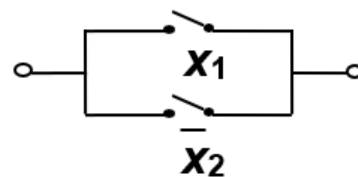


Рис. 3.3. Спрощена схема

Ця БФ може бути спрощеною:

$$\begin{aligned}
 x_3(x_1 \vee \bar{x}_2) \vee x_1\bar{x}_3 \vee \bar{x}_2 \bar{x}_3 &= x_1x_3 \vee \bar{x}_2 x_3 \vee x_1\bar{x}_3 \vee \bar{x}_2 \bar{x}_3 = \\
 &= x_1(x_3 \vee \bar{x}_3) \vee \bar{x}_2(x_3 \vee \bar{x}_3) = x_1 \vee \bar{x}_2.
 \end{aligned}$$

Отже, ланцюг на рис. 3.2 еквівалентний простішому ланцюгу, зображеному на рис. 3.3.

3.4. Задача мінімізації БФ у класі ДНФ і алгоритм її розв'язання

Проблема мінімізації функцій алгебри логіки (ФАЛ) пов'язана з поняттям найпростішого подання функцій і зводиться до проблеми вибору базису і найбільш компактного подання ФАЛ у цьому базисі.

Найбільш суттєві результати в розв'язанні задачі мінімізації отримані для базису $\{\wedge, \vee, \neg\}$. Розглянемо задачу мінімізації саме для цього базису в класі диз'юнктивних нормальних форм (ДНФ). Дамо такі визначення.

Визначення 3.3. Кон'юнкція $x_{i_1}^{\sigma_1}, x_{i_2}^{\sigma_2}, \dots, x_{i_k}^{\sigma_k}$ називається *елементарною*, якщо в ній кожна змінна зустрічається не більше одного разу.

Визначення 3.4. Рангом елементарної кон'юнкції називається число літер, що утворюють цю кон'юнкцію.

Визначення 3.5. Диз'юнкція елементарних кон'юнкцій називається *диз'юнктивною нормальною формою*.

Визначення 3.6. ДНФ для ФАЛ $f(x_1, \dots, x_n)$, що складається з кон'юнкцій рангу n , називається *довершеною диз'юнктивною нормальною формою (ДДНФ)*.

Вочевидь, ДДНФ складається з елементарних кон'юнкцій найбільшого для такої ФАЛ рангу.

Визначення 3.7. Число L елементарних кон'юнкцій, що утворюють ДНФ, називається *довжиною ДНФ*.

Визначення 3.8. ДНФ, що має найменшу довжину порівняно з усіма іншими ДНФ, які еквівалентні цій, називається *найкоротшою ДНФ (НДНФ)*.

Визначення 3.9. ДНФ, що містить найменше число літер $x_i^{\alpha_i} \bar{x}_i^{\alpha_i}$ порівняно з усіма іншими ДНФ, які еквівалентні цій, називається *мінімальною ДНФ (МДНФ)*.

Визначення, аналогічні наведеному, можна дати і для випадку кон'юнктивних нормальних форм. Обмежимося, однак, розглядом класу ДНФ.

Розглянемо тривимірний куб (рис. 3.4) як область визначення ФАЛ трьох змінних. Елементом куба припишемо кон'юнкції різного рангу: вершинам – кон'юнкції третього рангу, ребрам – кон'юнкції другого рангу і граням – кон'юнкції першого рангу.

Зазначимо, що кожний геометричний еквівалент меншої розмірності *покривається* всіма геометричними еквівалентами більшої розмірності. За аналогією будемо говорити про покриття кон'юнкції більшого рангу відповідними кон'юнкціями меншого рангу.

Наприклад, кон'юнкції $x_1 x_2 x_3$ і $x_1 x_2 \bar{x}_3$ покриваються кон'юнкцією $x_1 x_2$. Операцію покриття називають операцією склеювання і аналітично подають у вигляді:

$$Ax \vee A\bar{x} = A \quad (3.1)$$

(де A – це спільна частина, у нашому випадку $x_1 x_2$).

Умовимося називати геометричні еквіваленти інтервалами.

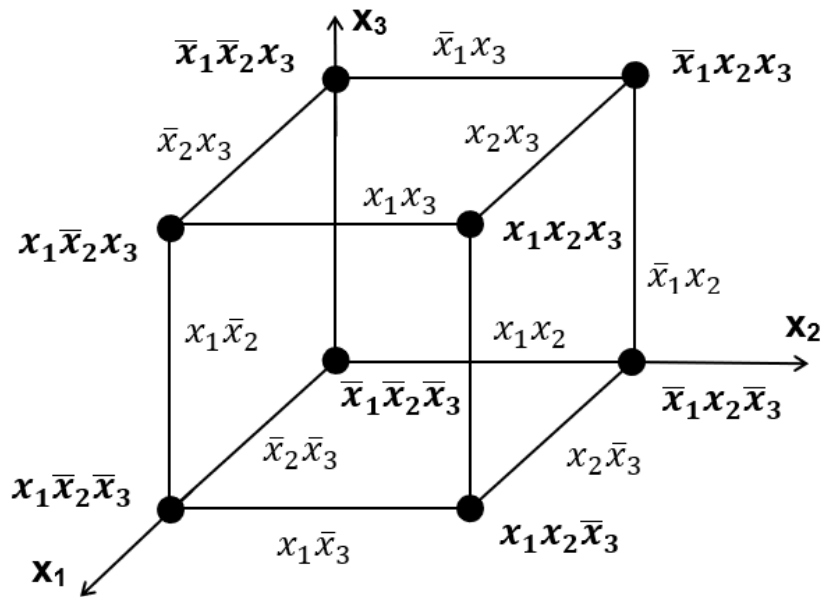


Рис. 3.4. Тривимірний куб як область визначення ФАЛ трьох змінних

Визначення 3.10. *Інтервалом i -го рангу* називається підмножина вершин куба, що відповідає кон'юнкції i -го рангу.

Наприклад, кон'юнкції x_1 відповідає множина вершин $(1,0,0), (1,0,1), (1,1,0), (1,1,1)$, тобто інтервал i -го рангу, що збігається з гранню куба та покриває чотири зазначені вершини. Оскільки вершини куба, що відповідають наборам аргументів, на яких ФАЛ набуває одиничного значення, утворюють множину T_1 , то задання деякої ДНФ для поданої ФАЛ еквівалентно заданню деякого покриття множини T_1 інтервалами, зумовленими кон'юнкціями, що входять у ДНФ. Іншими словами, існує взаємно однозначна відповідність між заданням функції у вигляді деякої ДНФ і покриттям множини T_1 для даної ФАЛ інтервалами деякого рангу.

Приклад 3.1

На рис 3.5(а) зображено покриття інтервалами рангу ФАЛ:

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3 \vee x_1 x_2 x_3,$$

поданої у вигляді ДДНФ, а на рис. 3.5(б) – покриття інтервалами *другого рангу* у вигляді спрощеної (за рахунок винесення за дужки $x_1 x_3, x_2 \bar{x}_3$) ДНФ:

$$f(x_1, x_2, x_3) = x_1 x_3 \vee x_2 \bar{x}_3$$

Таким чином, задачу про мінімізацію такої ФАЛ можна розглядати як задачу про знаходження мінімальної ДНФ для цієї функції.

Позначивши ранги всіх інтервалів, що утворять покриття для заданої ФАЛ, через $r_1, r_2, \dots, r_m$, отримуємо величину:

$$R = \sum_{i=1}^m r_i, \quad (3.2)$$

яка характеризує сумарний ранг ДНФ, що дорівнює числу букв, які входять у цю ДНФ.

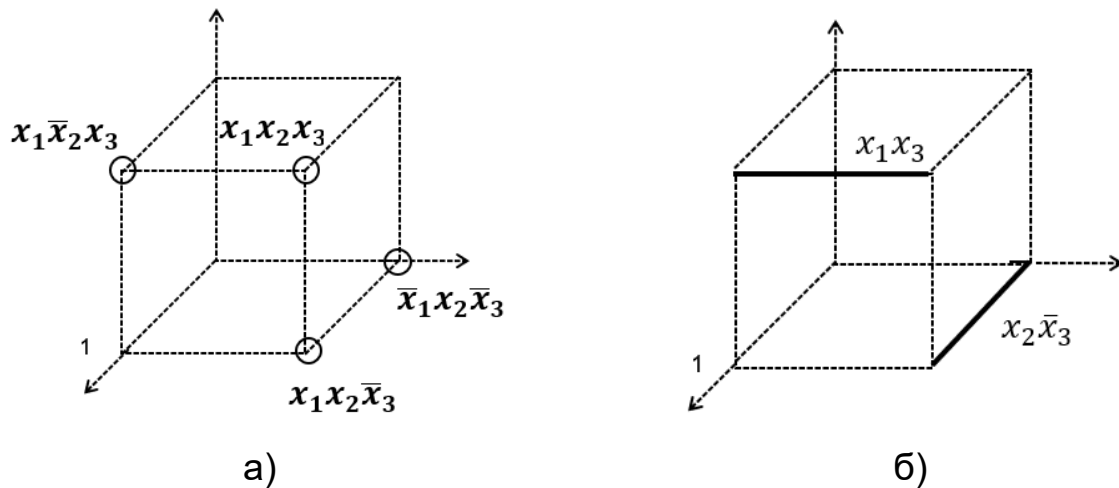


Рис. 3.5. Покриття інтервалами другого рангу

З огляду на визначення 3.6, сформулюємо задачу про мінімізацію ФАЛ як задачу про знаходження покриття множини, за якого величина R , обумовлена формулою (3.2), буде мінімальною.

Визначення 3.11. Інтервал J називається **максимальним**, якщо не існує інтервалу J' з рангом, меншим, ніж у J , і такого, що має місце співвідношення $J \subset J' \subset T_1$.

Наприклад, для функції вигляду:

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 x_3$$

максимальними будуть інтервали J_1, J_3 , а інтервал J_2 не є максимальним, оскільки має місце співвідношення $J_2 \subset J_1 \subset T_1$.

Визначення 3.12. ДНФ, що відповідає покриттю множини T_1 всіма максимальними інтервалами, називається скороченою ДНФ (СкДНФ).

Розглянемо це на прикладі функції:

$$f(x_1, x_2, x_3) = \bar{x}_1 x_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3 \vee x_1 x_2 x_3.$$

Покриття множини T_1 за допомогою максимальних інтервалів показано на рис. 3.6.–3.7. Спочатку знайдемо всі максимальні інтервали. Об'єднавши їх, отримуємо СкДНФ:

$$f(x_1, x_2, x_3) = x_1 x_3 \vee x_1 x_2 \vee x_2 \bar{x}_3.$$

Геометрична інтерпретація СкДНФ показана на рис. 3.6. Тут виділено три ребра, що відповідають усім можливим склейкам. Складність цієї форми (кількість літер) дорівнює 6.

Однак СкДНФ у загальному випадку не є мінімальною. Проаналізуємо отримані доданки: інтервал $x_1 x_2$ є зайвим, оскільки вершини, які він покриває, вже входять до інших інтервалів ($x_1 x_3$ та $x_2 \bar{x}_3$).

Вилучивши зайве, отримуємо МДНФ:

$$f(x_1, x_2, x_3) = x_1 x_3 \vee x_2 \bar{x}_3.$$

Геометрична інтерпретація МДНФ показана на рис. 3.7. Тут залишилися лише два необхідних ребра. Складність дорівнює 4.

Покриття на рис. 3.7 (МДНФ) є більш економним, ніж на рис. 3.6 (СкДНФ), співвідношення літер у яких дорівнює 4:6.

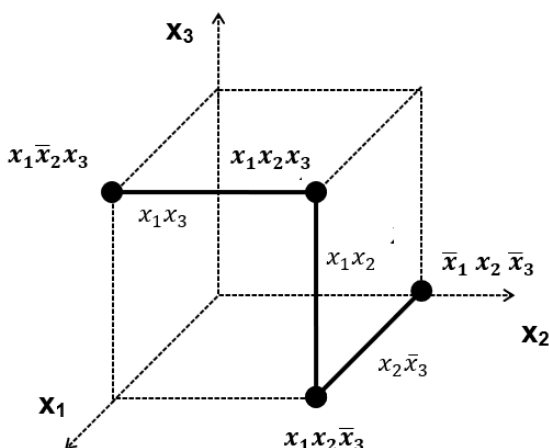


Рис. 3.6. Скорочена ДНФ (СкДНФ)

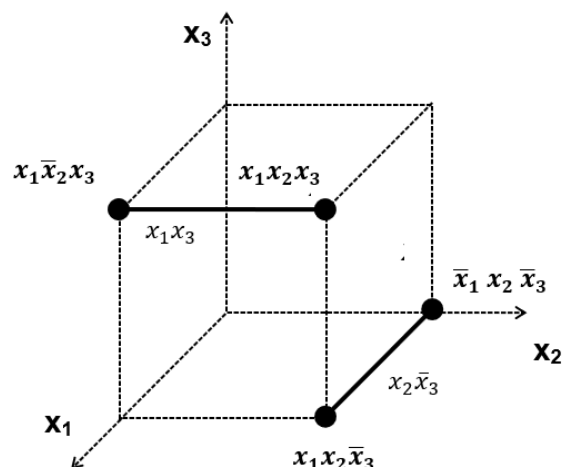


Рис. 3.7. Мінімальна ДНФ (МДНФ)

Теорема 3.1. Мінімальна ДНФ (МДНФ) виходить зі скороченої ДНФ (СкДНФ) шляхом видалення з покриття T_1 максимальними інтервалами деяких інтервалів.

Для доведення теореми достатньо показати, що покриття, яке відповідає МДНФ, складається тільки з максимальних інтервалів.

Припустимо протилежне: нехай у складі МДНФ є хоча б один немаксимальний інтервал. Тоді його можна було б замінити відповідним максимальним інтервалом (оскільки максимальний інтервал має менший ранг, тобто містить менше літер).

У результаті такої заміни кількість літер у ДНФ зменшилася б, але це суперечить визначенню МДНФ як мінімальної форми.

Отже, МДНФ складається винятково з максимальних інтервалів, тобто є частиною СкДНФ. Теорему доведено.

Визначення 3.13. ДНФ називається **тупиковою**, якщо у разі видалення з неї будь-якої кон'юнкції отримано ДНФ, яка не є еквівалентною вихідній.

3.5. Формулювання задачі мінімізації у класі ДНФ

Із наведених вище міркувань ясно, що задача мінімізації ФАЛ у класі ДНФ зводиться до відшукування МДНФ заданої ФАЛ у два етапи. На першому етапі однозначно за заданою ФАЛ будується СкДНФ, а на другому етапі шляхом видалення зайвих максимальних інтервалів із СкДНФ виконується пошук МДНФ.

На відміну від першого етапу другий етап є неоднозначним через те, що для заданої ФАЛ може існувати декілька МДНФ, а крім того, видалення зайвих максимальних інтервалів здійснюється шляхом перебирання.

Приклад 3.2

Для ФАЛ вигляду:

$$f(x_1, x_2, x_3) = \bar{x}_1 \bar{x}_2 x_3 \vee \bar{x}_1 x_2 \bar{x}_3 \vee \bar{x}_1 x_2 x_3 \vee x_1 \bar{x}_2 \bar{x}_3 \vee x_1 \bar{x}_2 x_3 \vee x_1 x_2 \bar{x}_3$$

покриття, що відповідає СДНФ, показано на рис. 3.8 (а).

На рис. 3.8 (б), (в) показано два можливих варіанти покриття, які отримано шляхом видалення зайвих інтервалів із попереднього покриття. Ці варіанти покриття відповідають двом можливим МДНФ для такої ФАЛ:

$$f_1(x_1, x_2, x_3) = x_1 \bar{x}_2 \vee \bar{x}_1 x_3 \vee x_2 \bar{x}_3$$

і

$$f_2(x_1, x_2, x_3) = \bar{x}_1 x_3 \vee \bar{x}_1 x_2 \vee \bar{x}_2 x_3.$$

Якщо до вихідного ДНФ застосувати лише операції склеювання (3.1) і елементарного поглинання

$$AB \vee A = A, \quad (3.3)$$

то може настати момент, коли операції виявляться непридатними; у такому випадку одержують тупикову ДНФ (ТДНФ). Серед множини ТДНФ міститься і МДНФ.

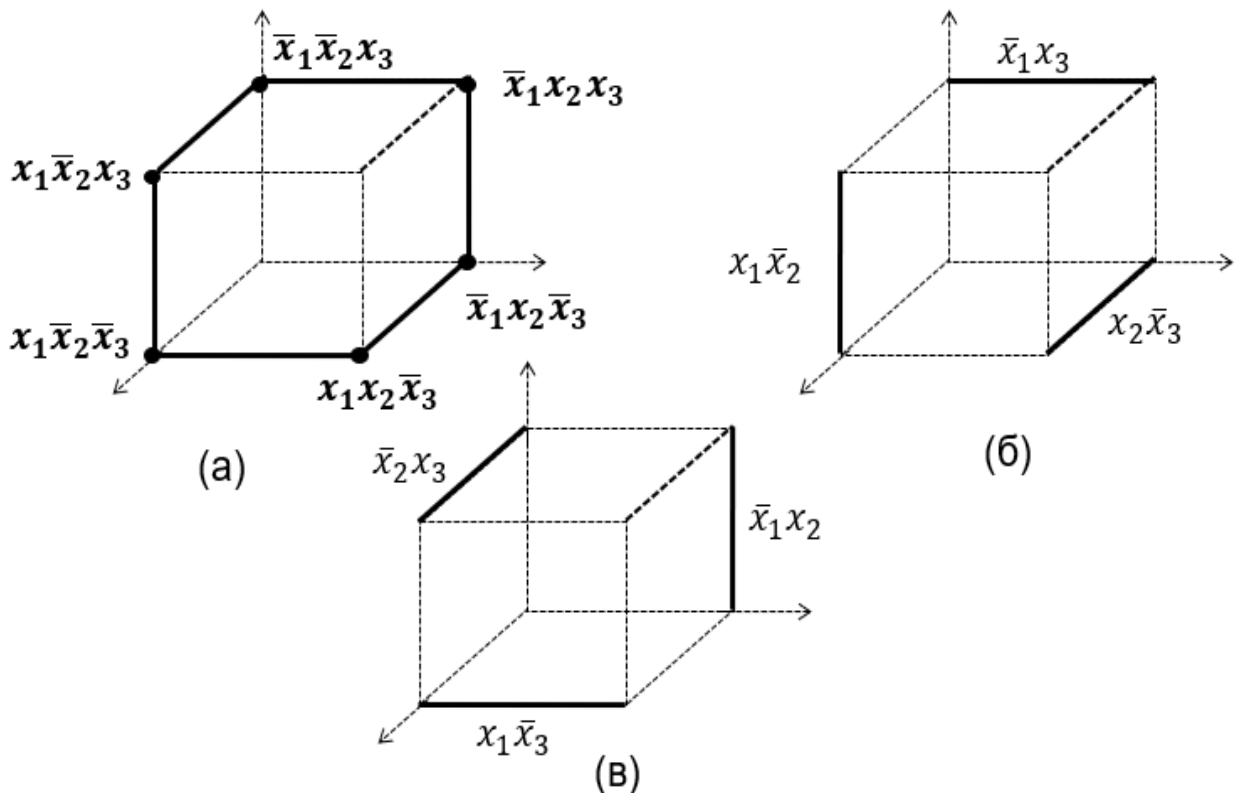


Рис. 3.8. Варіанти покриття ФАЛ

3.6. Методи мінімізації у класі ДНФ. Метод Квайна – Мак-Класкі

Метод Квайна ґрунтується на твердженні, що якщо в ДДНФ заданої ФАЛ виконати всі операції неповного склеювання:

$$(x \wedge F) \vee (\bar{x} \wedge F) = (x \wedge F) \vee (\bar{x} \vee F) \vee F = F,$$

а потім усі можливі поглинання $(F \wedge F) \vee F = F$, то отримаємо СкДНФ ФАЛ, з якої потім знаходять МДНФ шляхом вилучення зайвих імплікант.

Ця ідея методу реалізується у формі алгоритму.

1-й етап. Формування простих імплікант

Називаючи всі кон'юнкції n -го рангу, що входять до ДДНФ, мінітермами, виконуємо попарне склеювання. При цьому отримуємо імпліканти $(n - 1)$ -го рангу. Усі мінітерми n -го рангу, що вступили у склеювання, позначаємо (відмічаємо). Потім склеюємо отримані імпліканти $(n - 1)$ -го рангу, одержуючи мінітерми $(n-2)$ -го рангу, і т. д. Етап закінчується, коли отримані мінітерми i -го рангу вже не склеюються між собою. Усі невідмічені протягом цього процесу кон'юнкції (ті, що не взяли участь у склеюванні на своєму етапі) λ_i є простими імплікантами. У результаті одержують СкДНФ у вигляді:

$$f(x_1, \dots, x_n) = \lambda_1 \vee \lambda_2 \vee \dots \vee \lambda_k = \bigvee_{i=1}^k \lambda_i. \quad (3.4)$$

Приклад 3.4

Нехай потрібно знайти МДНФ:

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 \bar{x}_2 \bar{x}_3 x_4 \vee \bar{x}_1 \bar{x}_2 x_3 \bar{x}_4 \vee \bar{x}_1 \bar{x}_2 x_3 x_4 \vee \bar{x}_1 x_2 \bar{x}_3 \bar{x}_4 \vee \bar{x}_1 x_2 \bar{x}_3 x_4 \vee \bar{x}_1 x_2 x_3 \bar{x}_4 \vee \bar{x}_1 x_2 x_3 x_4 \vee x_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \vee x_1 \bar{x}_2 \bar{x}_3 x_4 \vee x_1 \bar{x}_2 x_3 \bar{x}_4 \vee x_1 \bar{x}_2 x_3 x_4 \vee x_1 x_2 \bar{x}_3 \bar{x}_4 \vee x_1 x_2 \bar{x}_3 x_4 \vee x_1 x_2 x_3 \bar{x}_4 \vee x_1 x_2 x_3 x_4. \quad (3.8)$$

Виконуємо 1-й етап алгоритму для вилучення простих імплікант.

Маємо мінітерми 3-го рангу (табл. 3.4).

Таблиця 3.4

Результати першого етапу склеювання

$\bar{x}_1 \bar{x}_2 \bar{x}_3$	$\bar{x}_1 \bar{x}_2 \bar{x}_4$	$\bar{x}_1 \bar{x}_3 \bar{x}_4$	$\bar{x}_2 \bar{x}_3 \bar{x}_4$
$\bar{x}_1 \bar{x}_2 x_4$	$\bar{x}_2 \bar{x}_3 x_4$	$\bar{x}_1 \bar{x}_2 x_3$	$\bar{x}_2 x_3 \bar{x}_4$
$\bar{x}_1 x_2 \bar{x}_4$	$x_1 \bar{x}_2 \bar{x}_3$	$\bar{x}_1 x_3 x_4$	$\bar{x}_2 x_3 x_4$
$\bar{x}_1 x_2 x_3$	$x_1 \bar{x}_2 x_4$	$x_2 x_3 x_4$	$x_1 x_3 x_4$

Мінітерми 2-го рангу наведено в табл. 3.5.

Таблиця 3.5

Список простих імплікант

Мінітерми	$\bar{x}_1 \bar{x}_2$	$\bar{x}_2 \bar{x}_3$	$\bar{x}_1 \bar{x}_4$
Прості імпліканти	$\bar{x}_2 x_4$	$\bar{x}_1 x_3$	$x_3 x_4$

Ці мінітерми склеїти вже не можна, і тому вони є простими імплікантами. Тоді маємо СкДНФ:

$$f(x_1, x_2, x_3, x_4) = \bar{x}_1 \bar{x}_2 \vee \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_4 \vee \bar{x}_2 x_4 \vee \bar{x}_1 x_3 \vee x_3 x_4.$$

2-й етап. Складання імплікантної таблиці (табл. 3.6)

У цій таблиці на перетині відповідного стовпця і рядка ставиться позначка «✓», якщо в деякий мінітерм ДДНФ входить яка-небудь із простих імплікант.

Таблиця 3.6

Імплікантна таблиця

Мінітерми		$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$	$\bar{x}_1$	x_1	x_1	x_1	x_1
		$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$	x_2	x_2	$\bar{x}_2$	$\bar{x}_2$	$\bar{x}_2$	x_2
Прості імпліканти		$\bar{x}_3$	$\bar{x}_3$	x_3	x_3	$\bar{x}_3$	x_3	$\bar{x}_3$	$\bar{x}_3$	x_3	x_3
		$\bar{x}_4$	x_4	$\bar{x}_4$	x_4	$\bar{x}_4$	$\bar{x}_4$	x_4	$\bar{x}_4$	x_4	x_4
	$\bar{x}_1\bar{x}_2$	✓	✓	✓	✓						
γ	$\bar{x}_2\bar{x}_3$	✓	✓					✓	✓		
γ	$\bar{x}_1\bar{x}_4$	✓		✓		✓	✓				
	$\bar{x}_2x_4$		✓		✓				✓	✓	
	$\bar{x}_1x_3$			✓	✓		✓	✓			
γ	x_3x_4				✓			✓		✓	✓

3-й етап. Вибір суттєвих імплікант

Суттєвою імплікантою називається проста імпліканта, що стоїть у рядку, який перетинається зі стовпцем таблиці, що має єдину позначку, і притому на перетині.

Суттєва імпліканта обов'язково повинна входити в шукану МДНФ, і таким чином, не може бути вилучена з виразу (3.4). З таблиці позначок викреслюються суттєві імпліканти, якщо відповідають їм рядки і стовпці мінітермів, які покриваються цими суттєвими імплікантами.

Для ФАЛ прикладу 3.4 суттєвими імплікантами є (див. табл. 3.1) $\bar{x}_2\bar{x}_3$, $\bar{x}_1\bar{x}_4$, x_3x_4 , позначені символом « γ ».

Після 3-го етапу маємо таблицю у вигляді трьох рядків, що залишилися з імплікантами $\bar{x}_1\bar{x}_2$, $\bar{x}_2x_4$, $\bar{x}_1x_3$ без жодного стовпця.

4-й етап. Вилучення зайвих стовпців імплікантної таблиці

Якщо в таблиці є два стовпці після 3-го етапу, в яких мітки стоять в однакових рядках, то один зі стовпців викреслюється. У прикладі 3.4 така ситуація відсутня.

5-й етап. Вилучення зайвих простих імплікант

Якщо після викреслювання зайвих стовпців на 4-му етапі в таблиці з'явилися рядки, що не містять жодної мітки, то прості імпліканти, що відповідають цим рядкам, виключають, оскільки вони не покривають стовбці з мінітермами.

У нашому прикладі 3.4 така ситуація відсутня.

6-й етап. Вибір мінімального покриття максимальними інтервалами, тобто вибір МДНФ

Щоб накрити одиницями множину T_1 , досить вибрати після 5-го етапу з імплікантної таблиці такі імпліканти, щоб вони усі разом своїми мітками покривали кожний стовпець хоча б один раз.

Під час перебирання різноманітних варіантів такого вибору перевага надається варіанту покриття з мінімальним сумарним числом букв у простих імплікантах, що утворюють покриття.

У прикладі 3.4 після третього етапу вже отримане мінімальне покриття, тобто МДНФ вихідної ФАЛ, що складається тільки з суттєвих імплікант:

$$f(x_1, x_2, x_3, x_4) = \bar{x}_2 \bar{x}_3 \vee \bar{x}_1 \bar{x}_4 \vee x_3 x_4. \quad (3.5)$$

Основним недоліком методу Квайна є необхідність повного попарного порівняння на 1-му етапі за наявності досить великого числа мінітермів вихідної ФАЛ.

У 1956 р. Мак-Класкі запропонував поліпшений варіант 1-го етапу алгоритму мінімізації Квайна, зменшивши число порівнянь.

Суть поліпшення полягає в тому, що мінітерми ДДНФ записуються у вигляді їхніх двійкових номерів, що розбиваються за числом одиниць у номерах на неперетинні групи так, що в -ту групу входять ті номери, що мають i одиниць. Попарне порівняння груп виробляється тільки між сусідніми за номером групами, оскільки лише ці групи відрізняються символами в одному розряді у мінітермах. У разі утворення мінітермів різних рангів у розрядах, що відповідають виключенням змінної, ставиться тире. Інші етапи аналогічні етапам методу Квайна.

Приклад 3.5

Мінімізуємо методом Квайна – Мак-Класкі ФАЛ з Приклада 3.4.
1-й етап: оскільки ФАЛ (3.8) задана ДДНФ, мінітермам якої відповідають двійкові номери 0000, 0001, 0010, 0011, 0100, 0110, 0111, 1000, 1001, 1011, 1111, то запис мінітермів по групах буде мати такий вигляд:

«0» – група: 0000;

«1» – група: 0001, 0010, 0100, 1000;

«2» – група: 0011, 0110, 1001;

«3» – група: 0111, 1011;

«4» – група: 1111.

Порівнявши сусідні групи, одержимо мінітерми 3-го і 2-го рангів за групами:

3-й ранг, мінітерми:

«0» – група: 000 – , 00 – 0, 0 – 00, –000;

«1» – група: 00 – 1, –001, 001 – , 0 – 10, 01 – 0, 100 – ;

«2» – група: 0 – 11, –011, 011 – , 10 – 1;

«3» – група: –111, 1 – 11.

2-й ранг, мінітерми:

«0» – група: 00 – –, –00 –, 0 – –0;

«1» – група: –0 – 1, 0 – 1 –;

«2» – група: – – 11.

Складаємо таблицю міток (табл. 3.7).

Далі, виконуючи етапи алгоритму Квайна, одержуємо МДНФ вихідної ФАЛ

$$f(x_1, x_2, x_3, x_4) = \overline{x_2}\overline{x_3} \vee \overline{x_1}\overline{x_4} \vee x_3x_4$$

Таким чином, результати мінімізації, що здійснена двома методами, збіглися. Аналогічна побудова та правила мінімізації у класі ДКНФ.

Таблиця 3.7

Таблиця покриття мінітермів простими імплікантами

Мінітерми	0000	0001	0010	0011	0100	0110	0111	1000	1001	1011	1111
Прості імпліканти	0000	0001	0010	0011	0100	0110	0111	1000	1001	1011	1111
0 0 – –	✓	✓	✓	✓							
– 0 0 –	✓	✓						✓	✓		
0 – –0	✓		✓		✓	✓					
–0 – 1		✓		✓					✓	✓	
0 – 1 –			✓	✓		✓	✓				
– – 11				✓			✓			✓	✓

Суттєвим недоліком розглядуваних методів мінімізації ФАЛ є громіздкість обов'язкового подання будь-якої ФАЛ у ДДНФ при досить невеликих n . У зв'язку з цим А. Блек і П. С. Порецький [7] запропонували метод мінімізації за довільною ДНФ. Ідея їхнього методу побудови СкДНФ за довільною ДНФ впливає з леми.

Лема. Якщо в ДНФ для заданої ФАЛ $f(x_1, \dots, x_n)$ входять два види кон'юнкції A_{x_i} і $B_{\bar{x}_i}$, то має місце рівність $F = F \vee AB$, де F – ДНФ, еквівалентна ФАЛ $f(x_1, x_2, \dots, x_n)$.

Виконавши після такого подання всі операції поглинання, приходимо до шуканої СкДНФ, а далі використовуємо метод Квайна.

3.7. Вправи до розділу

1. Використовуючи основні співвідношення алгебри логіки, довести еквівалентність формул булевих функцій (БФ) F і G :

- $F = xz \vee xy \vee xz, G = xyz \vee xz$;
- $F = x \rightarrow (xy \rightarrow ((x \rightarrow y) \rightarrow y)z), G = y \rightarrow (x \rightarrow z)$;
- $F = x \oplus (y \rightarrow z), G = (x \oplus y) \rightarrow (x \oplus z)$;
- $F = (x \vee y \vee z)(y \vee zu), G = xy \vee xu \vee yu \vee z$.

2. Звести формули F і G із завдання 1 до диз'юнктивної нормальної форми (ДНФ) і потім до досконалої диз'юнктивної нормальної форми (ДДНФ).

3. Звести до ДНФ і потім до ДДНФ такі формули:

- а) $F = (x_2 \vee x_1 x_0) \wedge (x_2 \vee x_0)$;
- б) $F = ((x_3 \vee x_2 x_1 x_0)((x_2 \vee x_0) \rightarrow x_3 x_1 x_0) \vee (x_3 \vee x_0))$;
- в) $F = (x_2 \oplus x_1) \rightarrow x_1 x_0$;
- г) $F = ((x_2 \rightarrow x_1) \vee x_1) \rightarrow x_1 x_0$;
- д) $F = (x_3 \sim x_2) \neg (x_1 \rightarrow x_0)$.

4. Подати БФ поліномом Жегалкіна:

- а) $x_1 \sim x_2$;
- б) $x_1 \downarrow x_2$.

5. Подати БФ у вигляді ДДНФ і ДКНФ:

- а) $x \mid y \rightarrow \bar{x}$;
- б) $x \downarrow y \rightarrow \bar{y}$;
- в) $\overline{(x \rightarrow y)} \vee (y \rightarrow x) \vee \bar{x}$;
- г) $(\bar{x}_1 \equiv x_2) \vee \bar{x}_3 \rightarrow x_1$

6. Довести повноту системи БФ:

- а) $\{\neg, \rightarrow\}$;
- б) $\{\neg, \sim\}$;
- в) $\{\oplus, \wedge, \vee\}$;
- г) $\{\sim, \vee, \oplus\}$.

7. Довести неповноту систем булевих функцій:

- а) $\{\rightarrow, \wedge, \vee\}$;
- б) $\{\oplus, \neg\}$.

8. Побудувати електричні ланцюги, для яких функції провідності задаються наданими ПФ; знайти еквівалентні їм простіші ланцюги:

- а) $x_2 \wedge (x_1 \vee \neg x_2) \vee x_1 \wedge \neg x_2 \vee x_3 \wedge (x_2 \vee x_1)$;
- б) $x_1 \wedge \neg x_3 \vee \neg x_1 \wedge x_2 \wedge x_3 \vee \neg x_1 \wedge (\neg x_2 \wedge \neg x_3 \vee x_2)$;
- в) $x_1 \wedge x_2 \wedge x_3 \vee x_1 \wedge x_3 \vee x_2 \wedge x_3 \vee (x_1 \wedge x_2 \wedge x_3 \vee x_2)$;
- г) $\neg x_1 \wedge x_2 \wedge \neg x_3 \wedge x_4 \vee \neg x_1 \wedge \neg x_3 \wedge x_4 \vee x_2 \wedge \neg x_3 \wedge x_4 \vee \neg x_3 \wedge x_4$.

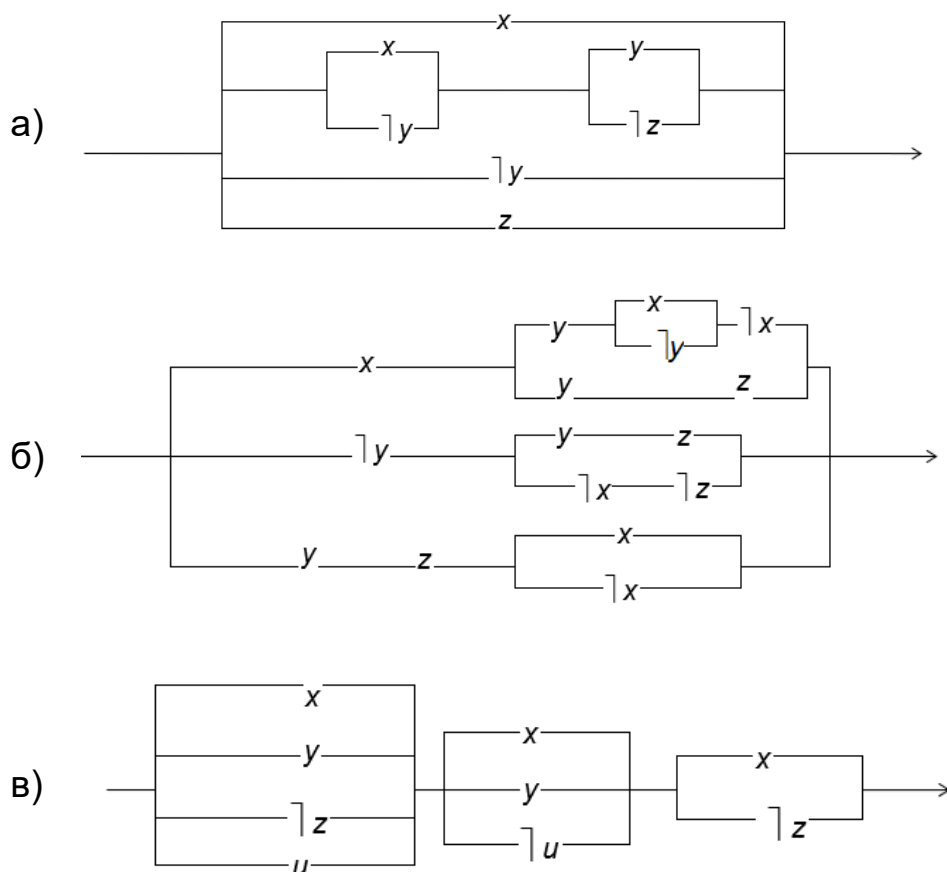
9. За поданими БФ побудувати релейно-контактну схему; БФ спростити, використовуючи формули поглинання і склеювання; побудувати релейно-контактну схему для спрощених БФ:

- а) $x_1 x_2 \vee x_3 \vee \bar{x}_1 \cdot (\bar{x}_2 \vee \bar{x}_3) \vee \bar{x}_2 \cdot (x_1 \vee x_2 \vee x_3)$;
- б) $x_1 \vee \bar{x}_2 x_3 \vee \bar{x}_1 \cdot (x_2 \vee x_3) \vee x_3 \vee \bar{x}_2 \cdot (x_1 \vee \bar{x}_2)$;
- в) $x_1 x_3 \vee x_1 \bar{x}_2 \vee x_2 \cdot (x_1 \vee x_3) \vee x_3 \cdot (\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3)$
- г) $x_2 \vee x_3 \cdot (\bar{x}_2 \vee x_1) \vee x_1 \cdot (x_3 \vee x_2) \vee \bar{x}_2 \vee \bar{x}_1$.

10. Спростити такі функції:

- а) $F = (x_1 \vee x_2)(\neg x_1 \vee x_3) \vee x_3(x_1 \vee x_2) \vee (x_4 \vee x_5);$
- б) $F = x \rightarrow (x \sim z) \wedge (y \rightarrow z) \wedge (x \rightarrow z);$
- в) $F = (xy \vee yz) \neg (\neg xu).$

11. Спростити схеми:



12. Довести повноту (неповноту) системи БФ, використавши теорему Поста – Яблонського:

- а) $\{ \vee, \sim, \neg \};$
- б) $\{ \oplus, \wedge, 1, 0 \};$
- в) $\{ \rightarrow, \sim, \vee \};$
- г) $\{ \sim, \oplus, 1 \}.$

13. Подати у вигляді полінома Жегалкіна такі булеві функції:

- а) $(xz \vee y)(x\bar{y} \vee \bar{x}\bar{y} \vee \bar{z})(x \vee \bar{y});$
- б) $((y \vee z)(t \vee y\bar{z})) \vee t\bar{x} \vee ((z \vee y)(\bar{t} \vee \bar{z}));$
- в) $yt \vee ((z \vee \bar{t})(x \vee z)(\bar{t} \vee \bar{z})(x \vee \bar{z})) \vee \bar{y}t;$
- г) $(\bar{z} \vee t)(t \vee x) \vee (\bar{z} \vee \bar{x})(\bar{z} \vee \bar{t})(\bar{t} \vee z);$
- д) $x\bar{t} \vee ((\bar{z}\bar{y}) \vee t)(z \vee y) \vee ((\bar{t} \vee \bar{z})(z \vee y));$
- е) $((t \vee \bar{t}z)\bar{t}) \vee \bar{y}((y \vee t)(y \vee x));$

- ж) $((z \vee \bar{x})(\bar{x} \vee \bar{t})(x \vee z)(\bar{y} \vee x)) \vee y\bar{t} \vee yt;$
 з) $(t \vee \bar{x}\bar{t} \vee x)(y \vee (t \vee tz))\bar{z}\bar{x}t;$
 и) $\bar{z}\bar{y} \vee tz \vee \bar{y}z \vee t\bar{z} \vee y\bar{t}.$

14. Із заданої множини елементарних кон'юнкцій K виділити прості імпліканти функції f , якщо:

- а) $K = \{x_2, \neg x_0, x_2x_1, x_2\neg x_2\}, f = (00101111);$
 б) $K = \{x_2\neg x_1, x_2x_1, x_2, x_2x_1x_0\}, f = (01111110).$

15. Знайти, які з кон'юнкцій цих ДНФ є простими імплікантами:

- а) $f = \neg x_3(\neg x_2) \vee x_3(\neg x_1)x_0 \vee x_2(\neg x_1x_0);$
 б) $f = \neg x_3(\neg x_2)x_1 \vee x_3(\neg x_2)x_0 \vee x_1x_0;$
 в) $f = x_3x_2 \vee \neg x_3x_1 \vee x_3(\neg x_2)x_1x_0 \vee (\neg x_3)x_2x_1(\neg x_0).$

16. З'ясувати, чи є тупиковими, найкоротшими чи мінімальними такі ДНФ:

- а) $f = x_1x_2 \vee \neg x_2;$
 б) $f = \neg x_3\neg x_4 \vee x_1\neg x_2 \vee \neg x_1\neg x_2x_3.$

17. Використовуючи формули поглинання і склеювання, мінімізувати БФ:

- а) $f(x_1, x_2, x_3) = x_1x_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3;$
 б) $f(x_1, x_2, x_3) = x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2x_3 \vee x_1x_2x_3;$
 в) $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee x_1x_2\bar{x}_3;$
 г) $f(x_1, x_2, x_3) = x_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3;$
 д) $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2x_3;$
 е) $f(x_1, x_2, x_3) = x_1x_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3.$

18. Мінімізувати БФ методом Квайна – Мак-Класкі:

- а) $f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3;$
 б) $f(x_1, x_2, x_3) = x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3;$
 в) $f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1x_2x_3 \vee x_1x_2\bar{x}_3;$
 г) $f(x_1, x_2, x_3) = x_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3;$
 д) $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2x_3;$
 е) $f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee x_1\bar{x}_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3.$

19. Використовуючи формули поглинання і склеювання та графічну інтерпретацію на одиничному кубі, мінімізувати БФ:

- а) $f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2\bar{x}_3 \vee x_1\bar{x}_2x_3;$
 б) $f(x_1, x_2, x_3) = x_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee x_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3;$
 в) $f(x_1, x_2, x_3) = x_1\bar{x}_2x_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee x_1x_2x_3 \vee x_1x_2\bar{x}_3;$
 г) $f(x_1, x_2, x_3) = x_1x_2\bar{x}_3 \vee x_1x_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2\bar{x}_3;$
 д) $f(x_1, x_2, x_3) = \bar{x}_1\bar{x}_2\bar{x}_3 \vee \bar{x}_1\bar{x}_2x_3 \vee \bar{x}_1x_2x_3 \vee \bar{x}_1x_2\bar{x}_3 \vee x_1x_2x_3;$

3.8. Програмна реалізація вправ

Завдання 1. Розробити програму, яка перетворює будь-яку задану БФ на вигляд полінома Жегалкіна.

Для розв'язання завдання була створена програма, яка дозволяє здійснювати аналіз булевих виразів з використанням таблиць істинності та полінома Жегалкіна. Програмний комплекс складається з кількох основних компонентів.

На першому етапі здійснюється токенізація введеного виразу. Функція `tokenize()` розбиває рядок на окремі токени (змінні $x_0, x_1, \dots$, логічні операції, дужки), що готує вхідні дані для подальшого аналізу.

Далі працює рекурсивний парсер (Parser), який враховує пріоритетність логічних операцій та перетворює інфіксний вираз на еквівалентний Python-вираз. Підтримуються базові логічні операції *and*, *or*, *not*, а також додаткові – *nand*, *implies*, *equiv*. Це дозволяє користувачеві вводити формули у звичному математичному вигляді.

На другому етапі здійснюється побудова таблиці істинності. Функція `compute_truth_table()` перебирає всі можливі комбінації значень змінних і для кожної з них обчислює результат виразу. Отримана таблиця відображається у зрозумілому для користувача форматі за допомогою `print_truth_table()`.

На третьому етапі реалізовано побудову полінома Жегалкіна. Для цього функція `get_coefficients()` обчислює коефіцієнти багаточлена методом поступового додавання за модулем два, а `zhegalkin_polynomial()` формує символічний вигляд полінома у вигляді суми мономів.

Головна функція `main()` організовує взаємодію з користувачем: зчитує кількість змінних і вираз, після чого керує усіма етапами – від обчислення таблиці істинності до виведення коефіцієнтів та полінома Жегалкіна.

Таким чином, створена програма забезпечує повний цикл роботи з булевими функціями: введення виразу, побудову таблиці істинності та отримання аналітичного представлення у вигляді полінома Жегалкіна.

```
import itertools
def nand(a, b):
    """NAND (штрих Шеффера). Повертає логічне значення
    (True/False)."""
    return not (a and b)
def implies(a, b):
    """Імплікація ( $a \rightarrow b$ ). Повертає логічне значення
    (True/False)."""
    return (not a) or b
def equiv(a, b):
    """Еквівалентність ( $a \leftrightarrow b$ ). Повертає логічне
    значення (True/False)."""
    return a == b
def tokenize(s):
    """
```



Розбиває рядок виразу на прості токени:

- знайомі ключові слова/імена змінних (x0, x1, ...),
- дужки '(' і ')'.
Не використовує регулярних виразів.

```
"""
```

```
tokens = []
i = 0
while i < len(s):
    c = s[i]
    if c.isspace():
        i += 1
        continue
    if c in '()':
        tokens.append(c)
        i += 1
        continue
    if c.isalpha():
        j = i + 1
        while j < len(s) and (s[j].isalpha() or
s[j].isdigit()):
            j += 1
        tokens.append(s[i:j])
        i = j
        continue
    if c.isdigit():
        j = i + 1
        while j < len(s) and s[j].isdigit():
            j += 1
        tokens.append(s[i:j])
        i = j
        continue
    # інші символи — беремо як один токен (на майбутнє)
    tokens.append(c)
    i += 1
return tokens
class Parser:
    """
    Простий рекурсивний парсер для інфіксних булевих виразів.
    Підтримує пріоритети: equiv < implies < or < and/nand < not <
atom.
    Перетворює на рядок Python-виразу, використовуючи:
        - 'and', 'or', 'not' як в Python,
        - 'nand(..., ...)', 'implies(..., ...)', 'equiv(..., ...)'
для відповідних операцій.
    """
    def __init__(self, tokens):
        self.toks = tokens
        self.pos = 0
    def peek(self):
        """Повертає поточний токен (не зрушуючи позицію)."""
        return self.toks[self.pos] if self.pos < len(self.toks)
else None
    def consume(self, expected=None):
```

```

        """Забирає поточний токен; якщо expected задано – перевіряє
збіг."""
        t = self.peek()
        if t is None:
            raise ValueError("Неочікуваний кінець виразу")
        if expected and t != expected:
            raise ValueError(f"Очікувався токен '{expected}',
знайдено '{t}'")
        self.pos += 1
        return t
    def parse_expr(self):
        """Точка входу парсера: expr → equiv."""
        return self.parse_equiv()
    def parse_equiv(self):
        """equiv → implies { 'equiv' implies }"""
        left = self.parse_implies()
        while self.peek() and self.peek().lower() == 'equiv':
            self.consume()
            right = self.parse_implies()
            left = f"equiv({left}, {right})"
        return left
    def parse_implies(self):
        """implies → or_expr { 'implies' or_expr }"""
        left = self.parse_or()
        while self.peek() and self.peek().lower() == 'implies':
            self.consume()
            right = self.parse_or()
            left = f"implies({left}, {right})"
        return left
    def parse_or(self):
        """or_expr → and_expr { 'or' and_expr }"""
        left = self.parse_and()
        while self.peek() and self.peek().lower() == 'or':
            self.consume()
            right = self.parse_and()
            left = f"({left} or {right})"
        return left
    def parse_and(self):
        """and_expr → not_expr { ('and' | 'nand') not_expr }"""
        left = self.parse_not()
        while self.peek() and self.peek().lower() in ('and',
'nand'):
            op = self.consume().lower()
            right = self.parse_not()
            if op == 'and':
                left = f"({left} and {right})"
            else:
                left = f"nand({left}, {right})"
        return left
    def parse_not(self):
        """not_expr → 'not' not_expr | atom"""
        if self.peek() and self.peek().lower() == 'not':
            self.consume()

```

```

        operand = self.parse_not()
        return f"(not {operand})"
    return self.parse_atom()
def parse_atom(self):
    """atom → xN | '(' expr ')"""
    t = self.peek()
    if t == '(':
        self.consume('(')
        inside = self.parse_expr()
        self.consume(')')
        return f"({inside})"
    # змінна: починається з 'x' і далі цифри, наприклад x0, x12
    if isinstance(t, str) and len(t) >= 2 and t[0].lower() ==
    'x' and t[1:].isdigit():
        return self.consume()
    raise ValueError(f"Неочікуваний токен: {t}")
def parse_to_python(expr_str):
    """
    Перетворює інфіксний рядок expr_str на Python-вираз,
    який можна передати в eval(..., locals).
    """
    tokens = tokenize(expr_str)
    p = Parser(tokens)
    py = p.parse_expr()
    if p.peek() is not None:
        raise ValueError(f"Зайві токени в кінці: {p.peek()}")
    return py
def get_coefficients(n, f):
    """
    Обчислює коефіцієнти полінома Жегалкіна для функції з n
    змінними.
    Вхід: f — список значень функції (0 або 1) довжиною 2^n.
    Повертає список коефіцієнтів (0/1).
    """
    coeff = f[:]
    total = 2**n
    for i in range(n):
        step = 2**(i+1)
        for j in range(0, total, step):
            for k in range(2**i):
                coeff[j+k+2**i] = (coeff[j+k+2**i] + coeff[j+k]) %
2
    return coeff
def make_monom(u, n):
    """
    Створює моном за індексом u (в двійковому представленні u
    показує, які змінні входять у моном).
    Наприклад: u=3, n=2 → 'x0*x1'; якщо u=0 → '1'.
    """
    vars_ = []
    for bit in range(n):
        if (u >> bit) & 1:
            vars_.append(f"x{bit}")

```

```

    return "".join(vars_) if vars_ else "1"
def zhegalkin_polynomial(n, coeff):
    """
    Формує рядок полінома Жегалкіна (АНФ) з коефіцієнтів.
    Повертає рядок виду: '1 ⊕ x1 ⊕ x0*x1' або '0'.
    """
    terms = [make_monom(i, n) for i, c in enumerate(coeff) if c]
    return " ⊕ ".join(terms) if terms else "0"
def compute_truth_table(n, expr):
    """
    Обчислює таблицю істинності для n змінних і інфіксного виразу
    expr.
    Повертає кортеж (combinations, f), де combinations – список
    кортежів (0/1),
    f – список значень функції (0/1) у відповідному порядку.
    """
    py = parse_to_python(expr)    # перетворюємо інфіксний вираз у
    python-вираз
    combos = list(itertools.product([0, 1], repeat=n))
    results = []
    # локальні функції передаємо як імена, вони працюють з булевими
    аргументами
    locals_env = {"nand": nand, "implies": implies, "equiv": equiv}
    for combo in combos:
        # ставимо значення змінних (булеві) у середовище виконання
        for i in range(n):
            locals_env[f"x{i}"] = bool(combo[i])
        # обчислюємо вираз; результат приводимо до 0/1
        val = eval(py, {"__builtins__": {}}, locals_env)
        results.append(1 if val else 0)
    return combos, results
def print_truth_table(n, combos, f):
    """Друкує таблицю істинності у зручному форматі."""
    header = ' | '.join([f"x{i}" for i in range(n)]) + " | f"
    print("\nТаблиця істинності:")
    print(header)
    print('-' * len(header))
    for combo, val in zip(combos, f):
        print(' | '.join(str(x) for x in combo) + " | " + str(val))
def main():
    """Основна функція: зчитує n і вираз, друкує таблицю та поліном
    Жегалкіна."""
    try:
        n = int(input("Введіть кількість змінних n: ").strip())
        if n < 1:
            print("n має бути >= 1")
            return
    except Exception:
        print("Невірне число n")
        return
    expr = input("Введіть булевий вираз (наприклад: (x0 nand x1)
    implies not x0): ").strip()

```

```

try:
    combos, f = compute_truth_table(n, expr)
except Exception as e:
    print("Помилка при розборі/обчисленні виразу:", e)
    return
print_truth_table(n, combos, f)
coeff = get_coefficients(n, f)
print("\nКоефіцієнти полінома Жегалкіна:", coeff)
poly = zhegalkin_polynomial(n, coeff)
print("Поліном Жегалкіна:", poly)
if __name__ == "__main__":
    main()

```

Перевіримо роботу програми на прикладах логічних виразів, які охоплюють тривіальні та нетривіальні випадки, а також ілюструють різні типи результатів (тавтологія, суперечність, контингентний):

– **фундаментальний закон виключеного третього** $x_0 \text{ or } \text{not } x_0$ (рис. 3.9). Незалежно від значення змінної вираз завжди буде істинним, що підтверджує його належність до класу тавтологій;

```

... Введіть кількість змінних n: 2
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): x0 or not x0

Таблиця істинності:
x0 | x1 | f
-----
0 | 0 | 1
0 | 1 | 1
1 | 0 | 1
1 | 1 | 1

Коефіцієнти полінома Жегалкіна: [1, 0, 0, 0]
Поліном Жегалкіна: 1

```

Рис. 3.9. Найпростіша тавтологія

– **закон виключеного протиріччя** $x_0 \text{ and } \text{not } x_0$ (рис. 3.10). Змінна не може бути одночасно істинною і хибною, тому вираз завжди набуває значення «0» і є протиріччям. Поліном Жегалкіна: 0;

```

... Введіть кількість змінних n: 2
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): x0 and not x0

Таблиця істинності:
x0 | x1 | f
-----
0 | 0 | 0
0 | 1 | 0
1 | 0 | 0
1 | 1 | 0

Коефіцієнти полінома Жегалкіна: [0, 0, 0, 0]
Поліном Жегалкіна: 0

```

Рис. 3.10. Найпростіше протиріччя

– **проста змінна** x_0 (рис. 3.11). Значення функції безпосередньо збігається зі значенням змінної, тому вираз є контингентним. Поліном Жегалкіна: x_0 ;

```

... Введіть кількість змінних n: 1
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): x0

Таблиця істинності:
x0 | f
-----
0 | 0
1 | 1

Коефіцієнти полінома Жегалкіна: [0, 1]
Поліном Жегалкіна: x0

```

Рис. 3.11. Найпростіший контингентний вираз

– **вираз із NAND та імплікацією** $(x_0 \text{ nand } x_1) \text{ implies not } x_0$ (рис. 3.12). Результат залежить від конкретних значень змінних: у деяких випадках вираз істинний, а в інших – хибний. Це нетривіальний приклад контингентного виразу;

```

... Введіть кількість змінних n: 2
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): (x0 nand x1) implies not x0

Таблиця істинності:
x0 | x1 | f
-----
0 | 0 | 1
0 | 1 | 1
1 | 0 | 0
1 | 1 | 1

Коефіцієнти полінома Жегалкіна: [1, 0, 1, 1]
Поліном Жегалкіна:  $1 \oplus x_1 \oplus x_0 * x_1$ 

```

Рис.3.12. Нетривіальний приклад контингентного виразу

– **еквівалентність кон'юнкції та диз'юнкції заперечень** $(x_0 \text{ and } x_1) \text{ equiv } (\text{not } x_0 \text{ or not } x_1)$ (рис. 3.13). Вираз завжди хибний, що демонструє нетривіальний приклад протиріччя;

```

... Введіть кількість змінних n: 2
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): (x0 and x1) equiv (not x0 or not x1)

Таблиця істинності:
x0 | x1 | f
-----
0 | 0 | 0
0 | 1 | 0
1 | 0 | 0
1 | 1 | 0

Коефіцієнти полінома Жегалкіна: [0, 0, 0, 0]
Поліном Жегалкіна: 0

```

Рис. 3.13. Нетривіальний приклад протиріччя

– **закон транзитивності імплікації**

$(x_0 \text{ implies } x_1) \text{ and } (x_1 \text{ implies } x_2) \text{ implies } (x_0 \text{ implies } x_2)$ (рис. 3.14)). Цей вираз завжди істинний, що підтверджує його властивості у логіці та демонструє нетривіальний приклад тавтології;

```
... Введіть кількість змінних n: 3
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): (x0 implies x1) and (x1 implies x2) implies (x0 implies x2)
```

Таблиця істинності:

x0	x1	x2	f
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Коефіцієнти полінома Жегалкіна: [1, 0, 0, 0, 0, 0, 0, 0]
Поліном Жегалкіна: 1

Рис.3.14. Нетривіальний приклад тавтології

– **складна диз'юнкція еквівалентностей** $(x_0 \text{ equiv } x_1) \text{ or } (x_1 \text{ equiv } x_2)$ (рис. 3.15). Вираз може набувати як значення істини, так і значення хибності залежно від набору змінних, що підтверджує його контингентний характер;

```
... Введіть кількість змінних n: 3
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): (x0 equiv x1) or (x1 equiv x2)
```

Таблиця істинності:

x0	x1	x2	f
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Коефіцієнти полінома Жегалкіна: [1, 0, 1, 1, 0, 1, 1, 0]
Поліном Жегалкіна: $1 \oplus x_1 \oplus x_0 * x_1 \oplus x_0 * x_2 \oplus x_1 * x_2$

Рис. 3.15. Ще один нетривіальний приклад контингентного виразу

– **стійкість програми до некоректного вводу** та її здатність надавати інформативні повідомлення про синтаксичні помилки (рис. 3.16–3.19).

Для перевірки стійкості програми до неповного введення було введено логічний вираз, який обривається на операторі без наступного операнда (незавершений вираз). Програма успішно виявила синтаксичну помилку та вивела зрозуміле повідомлення про відсутність операнда в кінці виразу (рис. 3.16).

```
... Введіть кількість змінних n: 2
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): x0 and
Помилка при розборі/обчисленні виразу: Неочікуваний токен: None
```

Рис. 3.16. Приклад обробки синтаксичної помилки (незавершений вираз)

У тесті на рисунку 3.17 надано реакцію на введення некоректної кількості змінних (наприклад, від'ємне число, нуль або занадто велике значення). Програма коректно перехопила виняткову ситуацію та повідомила користувача про недопустимість введеного значення з чітким поясненням обмежень.

```
... Введіть кількість змінних n: x0  
Невірне число n
```

Рис. 3.17. Обробка виняткової ситуації у разі некоректного введення кількості змінних

Для тестування було введено вираз, що містив зайві токени або символи після завершення логічного виразу. Програма правильно ідентифікувала наявність зайвих елементів і надала інформативне повідомлення про помилку, вказавши на присутність непередбачених токенів (рис. 3.18).

```
... Введіть кількість змінних n: 2  
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): x0 x1  
Помилка при розборі/обчисленні виразу: Зайві токени в кінці: x1
```

Рис. 3.18. Обробка помилки наявності зайвих токенів у виразі

У випадку на рисунку 3.19 перевірялася обробка унарного оператора «not», після якого не було вказано операнда. Програма коректно виявила відсутність операнда після оператора заперечення та вивела зрозуміле повідомлення про синтаксичну помилку з вказівкою на конкретну проблему.

```
... Введіть кількість змінних n: 2  
Введіть булевий вираз (наприклад: (x0 nand x1) implies not x0): not  
Помилка при розборі/обчисленні виразу: Неочікуваний токен: None
```

Рис. 3.19. Обробка помилки при відсутності операнда після «not»

Програма коректно обробляє як прості, так і складні вирази, будує таблиці істинності та формує поліном Жегалкіна. Це робить її корисним інструментом для навчання, перевірки та аналізу властивостей булевих функцій.

Для подальшого вдосконалення доцільно розширити механізми обробки помилок, щоб користувач отримував більш зрозумілі повідомлення, наприклад: «незбалансовані дужки», «невідомий оператор», що значно покращить зручність використання.

4. ЧИСЛЕННЯ ВИСЛОВЛЮВАНЬ (ТЕОРІЯ L)

4.1. Виведення у теорії L

Існує багато варіантів побудови числення висловлювань (різних наборів правил та аксіом). Ми зупинимося на одному з найпоширеніших варіантів, який у літературі прийнято позначати як «теорія L » (де буква L походить від англійського слова Logic – логіка). Така назва дозволяє чітко відрізнити цю конкретну систему від інших логічних систем.

Визначимо числення висловлювань як теорію L . Нехай мова теорії L містить такі компоненти:

1) **пропозиційні букви (змінні)** $A, B, C, \dots$ (як символи метатеорії для позначення довільних формул також вживатимемо $P, Q, R, P_1, Q_1, \dots$);

2) **логічні зв'язки** $\neg$ (заперечення) та $\rightarrow$ (імплікація). Інші зв'язки визначаються через основні:

$$A \wedge B \equiv \neg(A \rightarrow \neg B)$$

$$A \vee B \equiv \neg A \rightarrow B$$

$$A \leftrightarrow B \equiv (A \rightarrow B) \wedge (B \rightarrow A) \equiv \neg((A \rightarrow B) \rightarrow \neg(B \rightarrow A));$$

3) **дужки** $(,)$.

Формули теорії L визначаються індуктивно:

1) пропозиційні літери $A, B, C, \dots$ є **формулами** L ;

2) якщо A і B – формули, то формулами є і комбінації символів $\neg A$, $(A \rightarrow B)$.

Деякі з формул називаються аксіомами. Аксіомою (або варіантом схеми аксіом) називається вираз, отриманий підстановкою у схему аксіом конкретних формул замість символів.

У теорії L прийнято такі три **схеми аксіом**:

1) $(A_1) A \supset (B \supset A)$;

2) $(A_2) (A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$;

3) $(A_3) (\neg A \supset \neg B) \supset (B \supset A)$,

де A, B, C – будь-які ПФ.

Якщо в кожному з цих трьох схем аксіом підставити конкретні ПФ, то одержимо аксіому за такою схемою аксіом. Отже, усіх аксіом буде нескінченно більше.

Приклади

$P \supset (Q \supset P)$, $P \supset (P \supset P)$ – аксіоми, одержані за першою схемою аксіом (підстановки $A = P$, $B = Q$ та $A = P$, $B = P$ відповідно);

$(P \supset Q) \supset (R \supset (P \supset Q))$ – також аксіома, одержана за першою схемою (підстановки $A = P \rightarrow Q$, $B = R$);

$(\neg A \supset \neg (P \supset Q)) \supset (P \supset Q) \supset A$ – аксіома, одержана за третьою схемою аксіом (підстановки $B = P \rightarrow Q$).

Уведемо два правила висновку в теорії L .

1. Правило виведення (називається *modus ponens*)

Modus ponens (MP) – латинське словосполучення, що означає «спосіб ствердження». Це одне з базових правил логічного виведення, яке формалізує перехід від умовного твердження до його наслідку.

Формулювання правила

Якщо формули A і $A \supset B$ (тобто $A \rightarrow B$) є теоремами, то формула B також є теоремою.

Формальний запис

$$\text{MP} = \frac{A, A \supset B}{B}.$$

Пояснення

Правило MP вказує, що пропозиційна формула B виводиться з формул A і $A \supset B$. Іншими словами, якщо A істинне, і з нього випливає B , то B також істинне.

2. Правило підстановки (називається *rule S*)

Rule substitution (S) – походить від англійського слова «substitution», що означає «підстановка». Це структурне правило виведення, яке дозволяє отримувати нові істинні формули (теореми) з уже доведених шляхом конкретизації змінних.

Формулювання правила

Якщо формула A є теоремою, що містить пропозиційні літери $Q_1, \dots, Q_m$, і ми замінюємо кожне входження цих літер на довільні формули $B_1, \dots, B_m$, відповідно, то отримана формула також є теоремою.

Формальний запис

$$S = \frac{A}{A(Q_1, \dots, Q_m \parallel B_1, \dots, B_m)},$$

де $A, B_1, \dots, B_m$ – формули;

$Q_1, \dots, Q_m$ – попарно різні пропозиційні літери (змінні);

$A(Q_1, \dots, Q_m \parallel B_1, \dots, B_m)$ – результат одночасного заміщення усіх входжень літер $Q_1, \dots, Q_m$ в A на формули $B_1, \dots, B_m$ відповідно.

Пояснення

Це правило гарантує, що закони логіки діють універсально. Якщо якась структура (наприклад, $P \rightarrow P$) є істинною для простих букв, то вона залишиться істинною, хоч би які складні вирази ми підставили замість P (наприклад, $(X \wedge Y) \rightarrow (X \wedge Y)$). Ключова вимога – підстановка має бути одночасною для всіх входжень змінної, щоб не порушити структуру формули.

Виведенням у теорії L називається будь-яка скінченна послідовність формул $A_1, A_2, \dots, A_n$, така, що для кожного i ($1 \leq i \leq n$) формула A_i є:

- або аксіомою;
- або збігається з якою-небудь попередньою формулою послідовності;

– або впливає з якихось попередніх формул за допомогою правил МР чи S.

Останню ПФ A_n множини $A_1, A_2, \dots, A_n$ називають **вивідною формулою**, або **теоремою теорії L**, і записують $\vdash A_n$, (читається: « A_n виведена» або « A_n є теоремою»).

Поняття виведення можна узагальнити шляхом уведення поняття «виведення з гіпотез».

Виведенням ПФ B з кінцевої множини формул Γ (можливо, $\Gamma = \emptyset$) чи гіпотез називається така кінцева послідовність формул $B_1, B_2, \dots, B_m$, що $B_m \in B$ і для кожного i ($1 \leq i \leq m$) B_i є або одна з формул множини Γ , або аксіома, або отримана з якихось попередніх ПФ за допомогою правил виведення МР чи S. У цьому випадку записують $\Gamma \vdash B$, якщо $\Gamma = \{A_1, A_2, \dots, A_n\}$, то $A_1, A_2, \dots, A_n \vdash B$.

Наприклад, послідовність $A \rightarrow B, A, B$ є виведенням ПФ B із гіпотез $A \rightarrow B$ і A, але не із гіпотези $A \rightarrow B$, тобто $A \rightarrow B, A \vdash B$.

Звичайно використовують скорочене виведення, коли як B_i можуть стояти теореми теорії L, отримані раніше.

Виведення з гіпотез

Поняття виведення можна узагальнити шляхом уведення поняття «виведення з гіпотез».

Надамо без доказів такі **властивості** виведення із гіпотез.

1. $\Gamma, B \vdash B$.
2. Якщо $\Gamma \vdash B$, то для будь-якої ПФ A: $\Gamma, A \vdash B$.
3. Якщо $\Gamma, A, C \vdash B$, то $\Gamma, C, A \vdash B$.
4. Якщо $\Gamma, A, A \vdash B$, то $\Gamma, A \vdash B$.
5. Якщо $\Gamma, A \vdash B$ та $\Gamma \vdash A$, то $\Gamma \vdash B$, зокрема, якщо $\Gamma, A \vdash B$ і $\vdash A$, то $\Gamma \vdash B$.
6. Якщо $\Gamma \vdash A_1, \Gamma \vdash A_2, \dots, \Gamma \vdash A_n$ і $A_1, A_2, \dots, A_n \vdash B$, то $\Gamma \vdash B$.
7. Якщо $\Gamma \vdash A \supset B$, то $\Gamma, A \vdash B$.
8. Якщо $\Gamma, A \vdash B$, то $\Gamma \vdash A \supset B$ (теорема про дедукцію).

4.2. Приклади доведення теорем в теорії L

Розглянемо приклади доведення теорем в теорії L.

Приклад 4.1. Довести в теорії L теорему $(A \rightarrow A)$.

Доведення

1. Для виведення першої формули A_1 використаємо другу схему аксіом (A2). Застосувавши до неї правило (S) у вигляді підстановки $(A, B, C \parallel A, (A \rightarrow A), A)$, отримаємо:

$$\vdash (A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow ((A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A));$$

2. Згідно із (A1) та підстановкою $(A, B \parallel A, (A \rightarrow A))$ одержимо:

$$\vdash (A \rightarrow ((A \rightarrow A) \rightarrow A));$$

3. Застосуємо МР до п.1 і п.2:

$$\vdash ((A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A));$$

4. Згідно із (A1) та підстановкою $(A, B \parallel A, A)$ маємо

$$\vdash (A \rightarrow (A \rightarrow A));$$

5. Застосувавши МР до п.3 і п.4, наприкінці одержимо

$$\vdash (A \rightarrow A).$$

Приклад 4.2. Довести: $\Gamma, A \rightarrow B, B \rightarrow C \vdash A \rightarrow C$ (транзитивність).

Доведення

За властивістю 2 достатньо довести транзитивність, беручи до уваги, що $\Gamma = \emptyset$.

1. $A \rightarrow B, B \rightarrow C, A \vdash C$, оскільки виведенням ПФ C із системи гіпотез $A \rightarrow B, B \rightarrow C, A$ є, наприклад, послідовність $A \rightarrow B, B \rightarrow C, A, B, C$. Тут перші три ПФ – гіпотези, четверта виводиться із 1-ї та 3-ї за правилом МР, 5-а – із 2-ї та 4-ї.

2. $A \rightarrow B, B \rightarrow C \vdash A \rightarrow C$ виводиться із твердження (1) за теоремою про дедукцію (властивість 8).

Приклад 4.3 Чи є виведенням послідовність виду:

$$\neg\neg\neg A,$$

$$(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A),$$

$$\neg\neg\neg A \rightarrow (\neg A \rightarrow \neg\neg\neg A),$$

$$\neg A \rightarrow \neg\neg\neg A, \neg\neg A \rightarrow A ?$$

Доведення. Ця послідовність дійсно є виведенням оскільки:

$\neg\neg\neg A$ – гіпотеза.

$(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A)$ – аксіома за схемою аксіом (A3), яка отримана підстановкою $(A, B \parallel A), \rightarrow \neg\neg\neg A, \neg A$.

$(A \rightarrow (A \rightarrow A)) \rightarrow ((A \rightarrow \neg A) \rightarrow \neg A)$ – аксіома за схемою аксіом A1, причому $(A, B \parallel \neg\neg\neg A, \neg A)$.

1. $\neg A \rightarrow \neg\neg\neg A$ – формула, що отримана за допомогою правила виведення МР до п.1 і 3.

2. $\neg\neg A \rightarrow A$ – формула, що отримана за допомогою за допомогою правила виведення МР до п.2 і 4.

Таким чином, можна записати: $\neg\neg\neg A \vdash \neg\neg A \rightarrow A$.

Методика розв'язання задач

Задача 1. Чи є виводом послідовність виду:

а) $\neg\neg\neg A,$

б) $(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A),$

в) $\neg\neg\neg A \rightarrow (\neg A \rightarrow \neg\neg\neg A),$

г) $\neg A \rightarrow \neg\neg\neg A,$

д) $\neg\neg A \rightarrow A ?$

Розв'язання. Ця послідовність дійсно є виводом, оскільки:

- а) $\neg\neg\neg A$ – гіпотеза;
- б) $(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A)$ – аксіома за схемою аксіом 3, яка отримана підстановкою $(A, B \parallel A), \rightarrow \neg\neg\neg A, \neg A$;
- в) $(A \rightarrow (A \rightarrow A)) \rightarrow ((A \rightarrow \neg A) \rightarrow \neg A)$ – аксіома за схемою аксіом 1, причому $(A, B \parallel \neg\neg\neg A, \neg A)$;
- г) $\neg A \rightarrow \neg\neg\neg A$ – формула, що отримана за допомогою правила виводу МР до п.1 и 3;
- д) $\neg\neg A \rightarrow A$ – формула, що отримана за допомогою правила виводу МР до п.2 и 4.

Таким чином, можна записати: $\neg\neg\neg A \vdash \neg\neg A \rightarrow A$.

Задача 2. Вивести в теорії L такі ПФ:

- а) $\neg A \rightarrow (A \rightarrow B)$;
- б) $\neg\neg A \rightarrow A$;
- в) $A \rightarrow \neg\neg A$;
- г) $A \rightarrow B \rightarrow (\neg B \rightarrow \neg A)$.

Розв'язання:

- а) $\vdash \neg A \rightarrow (A \rightarrow B)$:
 - 1) $\neg A$ – гіпотеза;
 - 2) $\neg A \rightarrow (\neg B \rightarrow \neg A)$ – аксіома за схемою А1;
 - 3) $\neg B \rightarrow \neg A$ – з п.1 та п.2 за правилом МР;
 - 4) $(\neg B \rightarrow \neg A) \rightarrow (A \rightarrow B)$ – аксіома за схемою А3;
 - 5) $(A \rightarrow B)$ – з п.3 та п.4 за правилом МР;
 - 6) $\neg A \vdash (A \rightarrow B)$ – за п.1–5;
 - 7) $\vdash \neg A \rightarrow (A \rightarrow B)$ – за теоремою про дедукцію.
- б) $\vdash (\neg\neg A \rightarrow A)$:
 - 1) $\neg\neg A$ – гіпотеза;
 - 2) $\neg\neg A \rightarrow (\neg A \rightarrow \neg\neg\neg A)$ – з п. а), доведеного вище (підстановка: A замінили на $\neg A$, B на $\neg\neg\neg A$);
 - 3) $\neg A \rightarrow \neg\neg\neg A$ – з п.1 та п.2 за правилом МР;
 - 4) $(\neg A \rightarrow \neg\neg\neg A) \rightarrow (\neg\neg A \rightarrow A)$ – аксіома за схемою А3 (підстановка: A замінили на A , B на $\neg\neg A$);
 - 5) $\neg\neg A \rightarrow A$ – з п.3 та п.4 за правилом МР;
 - 6) A – з п.1 та п.5 за правилом МР;
 - 7) $\neg\neg A \vdash A$ – (підсумок виведення з гіпотези);
 - 8) $\neg\neg A \rightarrow A$ – за теоремою про дедукцію.
- в) $\vdash \neg A \rightarrow \neg\neg A$:
 - 1) $\neg\neg\neg A \rightarrow \neg A$ – взято з п. б), доведеного вище (у доведеній формулі $\neg\neg B \rightarrow B$ зробили підстановку $B = \neg A$, отримавши $\neg\neg(\neg A) \rightarrow (\neg A)$);

2) $(\neg\neg\neg A \rightarrow \neg A) \rightarrow (A \rightarrow \neg\neg A)$ – схема аксіом А3 (підстановка у схемі $(\neg X \rightarrow \neg Y) \rightarrow (Y \rightarrow X)$ взяли $X = \neg\neg A$ та $Y = A$);

3) $A \rightarrow \neg\neg A$ – з п. 1 та п. 2 за правилом МР.

г) $\vdash (A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$.

Ідея доведення: використовуємо результати п. б) та п. в), щоб перейти від A до $\neg\neg A$ і назад, зв'язавши це з аксіомою А3:

1) $A \rightarrow B$ – гіпотеза 1 (основна);

2) $\neg\neg A$ – гіпотеза 2 (допоміжна для побудови імплікації);

3) $\neg\neg A \rightarrow A$ – п. б), доведений раніше;

4) A – з п. 2 та п. 3 за правилом МР;

5) B – з п. 1 та п. 4 за правилом МР (використали гіпотезу $A \rightarrow B$);

6) $B \rightarrow \neg\neg B$ – п. в), доведений раніше;

7) $\neg\neg B$ – з п. 5 та п. 6 за правилом МР;

8) $A \rightarrow B, \neg\neg A \vdash \neg\neg B$ – підсумок п.1– п.7);

9) $A \rightarrow B \vdash \neg\neg A \rightarrow \neg\neg B$ – за теоремою про дедукцію (зняли гіпотезу $\neg\neg A$);

10) $(\neg\neg A \rightarrow \neg\neg B) \rightarrow (\neg B \rightarrow \neg A)$ – схема аксіом А3 (підстановка: $X = \neg A, Y = \neg B$. Схема $(\neg X \rightarrow \neg Y) \rightarrow (Y \rightarrow X)$ стає $((\neg\neg A \rightarrow \neg\neg B) \rightarrow (\neg B \rightarrow \neg A))$);

11) $\neg B \rightarrow \neg A$ – з п. 9 та п. 10 за правилом МР;

12) $A \rightarrow B \vdash \neg B \rightarrow \neg A$ – фіксація результату;

13) $\vdash (A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$ – за теоремою про дедукцію (зняли гіпотезу $A \rightarrow B$).

4.3. Вправи до розділу

1. Чи можна вивести в теорії L такі послідовності:

а) $B \supset (\neg A \supset B)$;

б) $((A \supset B) \supset ((B \supset C) \supset (A \supset C)))$;

в) $((A \supset B) \supset (\neg B \supset \neg A))$?

2. Вивести в теорії L наступні ПФ:

а) $(A \supset A)$;

б) $A \supset \neg\neg A$;

в) $\neg\neg A \supset A$;

г) $\neg A \supset (A \supset B)$;

д) $(A \supset B) \supset (\neg B \supset \neg A)$;

е) $\vdash (A \supset (B \supset C)) \supset ((A \supset B) \supset (A \supset C))$;

ж) $\vdash (A \supset B) \supset ((A \supset \neg B) \supset \neg A)$;

з) $\vdash (A \supset B) \supset ((\neg A \supset B) \supset B)$ і) $\vdash (\neg A \supset A) \supset A$.

5. ЛОГІКА І ЧИСЛЕННЯ ПРЕДИКАТІВ

5.1. Формули логіки предикатів і їх перетворення

Незважаючи на значну роль розглянутої раніше логіки висловлень, її засобів так виявляється недостатньо для вивчення й опису найпростіших і досить складних логічних висновків у теорії та практиці.

Можливості математичної логіки виявляються значно розширеними, якщо замість булевих функцій від змінних одного типу розглядати функції, значення яких, як і раніше, належать множині $Y = \{0,1\}$, а аргументи – множині найбільш загального вигляду. Такі логічні функції називають *предикатами з двозначною логікою*.

Логіка предикатів припускає логіку висловлювань уже відомою і починається з аналізу побудови простих висловлювань. При цьому прості висловлювання виражають, що деякі об'єкти (чи об'єкт) мають визначені властивості, чи властивості знаходяться між собою в деяких співвідношеннях.

Якщо є деяка множина об'єктів M , то під *предикатом-властивістю*, визначеним на цій множині, розуміється функція, визначена на множині M , що набуває значення «ІСТИНА» (1) або «ХИБА» (0). Предикати-властивості називаються *унарними* (однорідними) предикатами і вважаються окремим випадком *предикатів-відношень*. Залежно від того, між скількома об'єктами встановлюються відношення, розрізняють *двомісні* (бінарні), *тримісні* (тернарні) і т. д., у загальному випадку – *n-місні* предикати. Наприклад, нехай M – множина натуральних чисел, тоді одномісним (унарним) предикатом $P(x)$, визначеним на M , може бути предикат-властивість P «бути простим числом».

Наприклад, при $n = 2, 3, 4, 10, 11$ і т. д. предикат $P(n)$ набуває значення:

$$P(2) = 1, P(3) = 1, P(4) = 0, P(10) = 0, P(11) = 1.$$

Аналогічно можна побудувати бінарні предикати « x більше за y », « x поділяє y », що скорочено позначимо як $P(x, y)$, $Q(x, y)$ для натуральних x і y .

Наприклад:

$$P(2,5) = 0, P(3,1) = 1, Q(2,6) = 1, Q(7,10) = 0.$$

Співвідношення

$$y = P(x_1, \dots, x_n), \quad (5.1)$$

де $x_1 = \{x_{11}, \dots, x_{1p}\}, \dots, x_n = \{x_{n1}, \dots, x_{ng}\}$, – предметні змінні та їхні алфавіти, є *n-місним* предикатом. Якщо у *n-місному* предикаті $P(x_1, \dots, x_n)$ зафіксувати деяке число m змінних і підставити замість них конкретні значення, то предикат перетворюється на *(n – m)-місний*. Самі висловлювання прийнято вважати *0-місними* предикатами.

Оскільки двозначні предикати набувають значення з множини $\{0,1\}$, вони можуть виступати аргументами булевих функцій і до них застосовуються операції алгебри логіки.

Нехай, наприклад, $P(x)$ і $Q(x)$ – одномісні предикати, визначені на множині M . Тоді за допомогою операцій алгебри логіки $\wedge, \vee, \neg, \rightarrow, \sim, /$ можна будувати нові предикати.

Предикат $R_1(x) = P(x) \wedge Q(x)$ – істинний для тих $x \in M$, для яких істинні обидва предикати $P(x)$ і $Q(x)$.

Предикат $R_2(x) = P(x) \vee Q(x)$ – істинний для тих $x \in M$, для яких істинний хоча б один із предикатів $P(x)$ або $Q(x)$.

Аналогічно вводяться операції $P(x) \rightarrow Q(x)$, $P(x) \sim Q(x)$, $P(x) / Q(x)$.

Крім того, із предикатів $P(x_1)$ і $Q(x_2)$ та булевих змінних x_3, x_4 можна будувати складні функції, наприклад:

$$Z = \{P(x_1) \rightarrow [Q(x_2) \vee (x_3 \wedge x_4)]\} \sim x_3.$$

У складних ФАЛ незалежними змінними були лише елементарні висловлювання, а в численні предикатів додаються ще предметні змінні предикатів і самі предикати. Це і становить головну особливість предикатного числення: воно дозволяє застосовувати операції, що якісно відрізняються від операцій числення висловлювань.

5.2. Оператори кватифікації та їхні властивості

Оператори, що відповідають новим операціям, називаються *кванторами*.

У численні предикатів використовуються квантори двох видів: квантор спільності та квантор існування.

Квантор спільності – це оператор, що будь-якому одномісному предикату $y = P(x)$ ставить у відповідність двозначну логічну змінну z , яка набуває значення 1 тоді й тільки тоді, коли $y = 1$ при всіх значеннях $x \in M$. Символічно це можна записати формулою:

$$y = \forall x P(x), \quad (5.2)$$

де $\forall x$ – символ квантора спільності ($\forall$ нагадує про німецьке слово «alle» – «усі»). Запис (5.2) читається так: «для всіх x має місце $P(x)$ » і означає, що це висловлювання істинне тоді й тільки тоді, коли предикат $P(x)$ істинний для всіх $x \in M$.

Квантор існування – це оператор, що ставить у відповідність будь-якому одномісному предикату $y = P(x)$ двозначну логічну змінну z , яка набуває значення 1 тоді й тільки тоді, коли існує хоча б одне значення $x \in M$, для якого $y = 1$. Якщо ж для всіх $x \in M$ виконується $y = 0$, то змінна z набуває значення 0. Символічний запис має вигляд:

$$z = \exists x P(x), \quad (5.3)$$

де $\exists x$ – символ квантора існування ($\exists$ нагадує німецьке слово «existieren» – «існувати»). Запис (5.3) читається так: «існує таке x , що має місце $P(x)$ »

і означає, що це висловлювання істинне тоді й тільки тоді, коли знайдеться хоча б один елемент множини M , для якого предикат $P(x)$ істинний.

Введені оператори мають такі властивості.

1. Відповідно до визначень кванторів логічна змінна z у виразах

$$z = \forall x P(x), z = \exists x P(x)$$

вже не є функцією від предметної змінної x . У цьому випадку змінна, до якої був застосований квантор, називається *зв'язаною змінною*. Незв'язані змінні називаються *вільними*.

2. Якщо квантори $\forall x$ і $\exists x$ застосовуються до об'єкта ФАЛ, то вони виражаються у вигляді скінченної кон'юнкції та диз'юнкції. Дійсно, подаючи задану ФАЛ

$$y = y(x_1, x_2, \dots, x_n)$$

у вигляді

$$y = y(k),$$

де k – двійковий номер точки n -вимірного двійкового простору, відповідно до визначень кванторів маємо:

$$\begin{aligned} \forall k y(k) &= y(0) \wedge y(1) \wedge \dots \wedge y(k) \wedge \dots \wedge y(2^n - 1), \\ \exists k y(k) &= y(0) \vee y(1) \vee \dots \vee y(k) \vee \dots \vee y(2^n - 1). \end{aligned} \quad (5.4)$$

Ця обставина дозволяє вважати квантори $\forall x$ і $\exists x$ узагальненою кон'юнкцією і узагальненою диз'юнкцією.

3. Якщо квантор ($\forall x$ чи $\exists x$) застосовується до m -місного предиката, то в результаті виходить знову предикат, але, завдяки зв'язуванню однієї змінної, цей предикат буде $(m - 1)$ -місним.

Наприклад, якщо $A(x, y)$ – двомісний предикат, визначений на множині M , то $\forall x A(x, y) = F(y)$ – одномісний предикат від змінної y , істинний для тих $y = b \in M$, для яких одномісний предикат $A(x, b)$ істинний для всіх x .

Застосувавши до вільної змінної y квантори $\forall y$ чи $\exists y$, одержимо такі висловлювання (тобто 0-місні предикати):

$$\forall y (\forall x A(x, y)) = \forall y \forall x A(x, y) \sim \forall x \forall y A(x, y),$$

$$\exists y (\exists x A(x, y)) = \exists y \exists x A(x, y) \sim \exists x \exists y A(x, y),$$

де « $\sim$ » – символ еквівалентності ($\equiv$).

Два предикати $A(x_1, \dots, x_n) \equiv B(x_1, \dots, x_n)$ – *еквівалентні*, якщо в області їхнього визначення M вони мають те саме значення істинності.

5.3. Перетворення формул логіки предикатів

Для перетворення формул, що містять предикати, операції логіки висловлень і квантори, у логіці предикатів використовується ряд законів. Розглянемо деякі з них у вигляді прикладів.

1. Винесення кванторів за дужки (дистрибутивність):

$$\forall x P(x) \wedge \forall x Q(x) \equiv \forall x (P(x) \wedge Q(x)); \quad (5.5)$$

$$\exists x P(x) \vee \exists x Q(x) \equiv \exists x (P(x) \vee Q(x)). \quad (5.6)$$

2. Перехід від квантора існування до квантора загальності й навпаки (закони де Моргана для кванторів):

$$\overline{\exists x P(x)} \equiv \forall x \overline{P(x)}; \quad (5.7)$$

$$\overline{\forall x P(x)} \equiv \exists x \overline{P(x)}. \quad (5.8)$$

Еквівалентності (5.7) і (5.8) є узагальненнями законів де Моргана, до яких вони зводяться у випадку скінченності множин $M = \{x\}$.

Застосовуючи заперечення до загальних частин (5.7) і (5.8), маємо:

$$\exists x P(x) \equiv \overline{\forall x \overline{P(x)}}; \quad (5.9)$$

$$\forall x P(x) \equiv \overline{\exists x \overline{P(x)}}. \quad (5.10)$$

3. Диз'юнкція і кон'юнкція кванторів (односторонні імплікації):

$$(\forall x P(x) \vee \forall x Q(x)) \rightarrow \forall x (P(x) \vee Q(x)); \quad (5.11)$$

$$\exists x (P(x) \wedge \exists x Q(x)) \rightarrow (\exists x P(x) \wedge Q(x)). \quad (5.12)$$

4. Еквівалентності (винесення незалежного висловлювання):

$$\forall x P(x) \vee C \equiv \forall x (P(x) \vee C); \quad (5.13)$$

$$\exists x P(x) \vee C \equiv \exists x (P(x) \vee C), \quad (5.14)$$

якщо C – висловлювання або предикат, який не залежить від x . Якщо позначення змінної в предикаті $P(x)$ довільне, то $\forall x P(x) \equiv \forall y P(y)$.

Отже, згідно з (5.13) можна записати такі еквівалентності:

$$\begin{aligned} \forall x P(x) \vee \forall x Q(x) &\equiv \forall x P(x) \vee \forall y Q(y) \equiv \forall x (P(x) \vee \forall y Q(y)) \equiv \\ &\equiv \forall x \forall y (P(x) \vee Q(y)), \end{aligned}$$

тобто має місце еквівалентність

$$\forall x P(x) \vee \forall x Q(x) \equiv \forall x \forall y (P(x) \vee Q(y)). \quad (5.15)$$

Аналогічно виводять еквівалентність

$$\exists x P(x) \wedge \exists x Q(x) \equiv \exists x \exists y (P(x) \wedge Q(y)). \quad (5.16)$$

Еквівалентності (5.15) і (5.16) дозволяють будь-які вирази логіки предикатів зіставляти з еквівалентними формулами деякого стандартного вигляду, так званими *попередніми нормальними формами*.

5.4. Основні визначення і приклади предикатів

Визначення 5.1. Предикатом називається функція, аргументи якої взято з довільної множини, а сама функція набуває значення 1 («істина») чи 0 («хибність»).

Предикат називається *n-місним* ($n = 1, 2, \dots$), якщо відповідна функція залежить від n аргументів.

Нехай $P(x_1, \dots, x_n)$ є n -місний предикат, аргументи $x_1, \dots, x_n$ якого набувають значення з деякої множини M . Якщо при $x_1 = a_1, \dots, x_n = a_n$, де $a_i \in M$ ($i = 1, 2, \dots, n$), маємо $P(a_1, \dots, a_n) = 1$ (або $P(a_1, \dots, a_n) = 0$), то набір точок $(a_1, \dots, a_n)$ задовольняє (або не задовольняє) предикату $P(x_1, \dots, x_n)$, що визначений на множині M .

Приклад 5.1

Одномісний предикат $R(x) = (x + 2 = 0)$:

точка $x = -2$, задовольняє предикат, оскільки

$$R(-2) = (-2 + 2 = 0) = 1,$$

точка $x = 1$ не задовольняє предикат, оскільки

$$R(1) = (1 + 2 = 0) = 0.$$

Приклад 5.2

Двомісний предикат

$$Q(x, y) = \left(\begin{cases} x + y = 3 \\ x - y = 1 \end{cases} \right):$$

точка $(2, 1)$ задовольняє предикат, бо $Q(2, 1) = \left(\begin{cases} 2 + 1 = 3 \\ 2 - 1 = 1 \end{cases} \right) = 1$,

точка $(1, 2)$ не задовольняє предикат, бо $Q(1, 2) = \left(\begin{cases} 1 + 2 = 3 \\ 1 - 2 \neq 1 \end{cases} \right) = 0$.

Якщо покласти $x = 3$, то одержимо предикат

$$Q(3, y) = \left(\begin{cases} 3 + y = 3 \\ 3 - y = 1 \end{cases} \right),$$

який набуває значення 0 (тобто хибність) при будь-якому значенні y .

Приклад 5.3

n -місний предикат

$$P(x_1, \dots, x_n) = (x_1^2 + \dots + x_n^2 \geq 0)$$

задовольняє будь-який набір $(x_1, \dots, x_n) \in \mathbb{R}^n$ оскільки сума квадратів завжди невід'ємна.

Якщо фіксуємо всі змінні, крім x_1 , покладемо $x_2 = 2, x_3 = 3, \dots, x_n = n$, то отримаємо одномісний предикат

$$P(x_1, 2, 3, \dots, n) = (x_1^2 + 2^2 + \dots + n^2 \geq 0),$$

який задовольняє кожне значення $x_1 \in \mathbb{R}$.

Однак кожний набір $(a_1, \dots, a_n)$ значень аргументів $x_1, \dots, x_n$ не задовольняє n -місний предикат

$$P_1(x_1, \dots, x_n) = (x_1^2 + \dots + x_n^2 < 0),$$

оскільки сума квадратів ніколи не буває від'ємною.

Часто **одномісний предикат** називають *властивістю*.

Наприклад, предикат $P_1(x) = (x > 0) = 1$ на множині дійсних чисел визначає властивість числа x бути додатним.

Нульмісний предикат не залежить від жодної змінної, тобто є висловлюванням, наприклад: $2 = 2$ (істина), $5 < 3$ (хибність), $12 + 2 = 4$ (хибність).

Визначення 5.2. Для $n \geq 1$ n -місний предикат $P(x_1, \dots, x_n)$ на деякій множині M називається:

- **тотожно-істинним**, якщо для будь-якого набору $(a_1, \dots, a_n) \in M$ значень аргументів $x_1, \dots, x_n$ предикат $P(a_1, \dots, a_n)$ дорівнює 1;

- **тотожно-хибним**, якщо для будь-якого набору $(a_1, \dots, a_n) \in M$ предикат $P(a_1, \dots, a_n)$ дорівнює 0;

- **виконаним**, якщо існує хоча б один набір значень аргументів, для якого предикат $P(a_1, \dots, a_n)$ дорівнює 1.

Наприклад,

- предикат $P(x_1, \dots, x_n)$ (див. приклад 5.3) – *тотожно-істинний*, бо сума квадратів завжди ≥ 0 ;

- предикат $P_1(x_1, \dots, x_n)$ – *тотожно-хибний*, бо сума квадратів ніколи не буває < 0 ;

- предикат $Q(x, y)$ (див. приклад 5.2) – *виконаний*, бо існують пари (x, y) , для яких $Q(x, y) = 1$.

Істинне висловлювання є нульмісним тотожно-істинним предикатом, а хибне – нульмісним тотожно-хибним предикатом.

Визначення 5.3. Із двох n -місних предикатів $R(x_1, \dots, x_n)$ і $Q(x_1, \dots, x_n)$ на тій самій множині предикат $Q(x_1, \dots, x_n)$ називається наслідком $R(x_1, \dots, x_n)$, якщо будь-який набір значень змінних, що задовольняє $R(x_1, \dots, x_n)$, задовольняє і $Q(x_1, \dots, x_n)$.

Скорочено наслідок позначається $R(x_1, \dots, x_n) \Rightarrow Q(x_1, \dots, x_n)$.

Приклад

1. Із предикатів $R(n)$: (n ділиться на 4) і $Q(n)$: (n ділиться на 2) маємо: $Q(n)$ – наслідок $R(n)$ на множині натуральних чисел, тобто: $R(n) \Rightarrow Q(n)$.

2. Предикат $Q_1(x_1, x_2) = ((x_1 - x_2) \geq 5)$ є наслідком $R_1(x_1, x_2) = ((x_1 - x_2) = 5)$ на множині дійсних чисел, але не навпаки, тобто $R_1(x_1, x_2) \Rightarrow Q_1(x_1, x_2)$, але не $Q_1(x_1, x_2) \Rightarrow R_1(x_1, x_2)$.

3. Для предикатів $R_2(n) = (n + 2 = 4)$ і $Q_2(n) = (n = 2)$ на множині натуральних чисел кожний є наслідком іншого, тобто

$$R_2(n) \Rightarrow Q_2(n) \text{ і } Q_2(n) \Rightarrow R_2(n).$$

Таким чином, із визначення 5.3 випливає:

- будь-який предикат на зазначеній множині є наслідком *тотожно-хибного предиката* на цій же множині;

- *тотожно-істинний предикат* – наслідок будь-якого предиката на зазначеній множині.

Визначення 5.4. Предикати $R(x_1, \dots, x_n)$ і $Q(x_1, \dots, x_n)$ на тій самій множині називаються **рівносильними**, якщо їхні значення збігаються для будь-якого набору значень змінних $x_1, \dots, x_n$, і позначаються:

$$R(x_1, \dots, x_n) \equiv Q(x_1, \dots, x_n).$$

Наприклад, $R_2(n) \equiv Q_2(n)$, де $R_2(n)$ і $Q_2(n)$, – предикати з попереднього прикладу 5.3:

$$R(x, y) = (x^2 - y^2 \geq 1), Q(x, y) = ((x - y)(x + y) \geq 1)$$

на множині дійсних чисел є рівносильними, оскільки вирази $(x^2 - y^2 \geq 1)$ та $(x - y)(x + y) \geq 1$ тотожно рівні:

$$R(x, y) \equiv Q(x, y).$$

Із визначення 5.4 виходить, що:

- рівносильні рівняння, системи рівнянь, нерівності та системи нерівностей задають рівносильні предикати;
- при підстановці наборів значень змінних предикати утворюють висловлювання;
- отже, над предикатами можна робити операції логіки висловлень і в результаті отримувати нові предикати.

Приклад 5.4

Прямокутна область $ABCD$ на рис.5.1 описується предикатом:

$$S(x, y) = R(x, a) \wedge Q(b, y), \text{ де } R(x, a) = (a^2 - x^2 \geq 0), Q(b, y) = (b^2 - y^2 \geq 0). \text{ Отже } S(x, y) = (a^2 - x^2 \geq 0) \wedge (b^2 - y^2 \geq 0)$$

на множині дійсних чисел описує *внутрішність* прямокутника $ABCD$.

Предикат $\neg S(x, y) = (a^2 - x^2 < 0) \vee (b^2 - y^2 < 0)$ описує область, що лежить поза прямокутником $ABCD$.

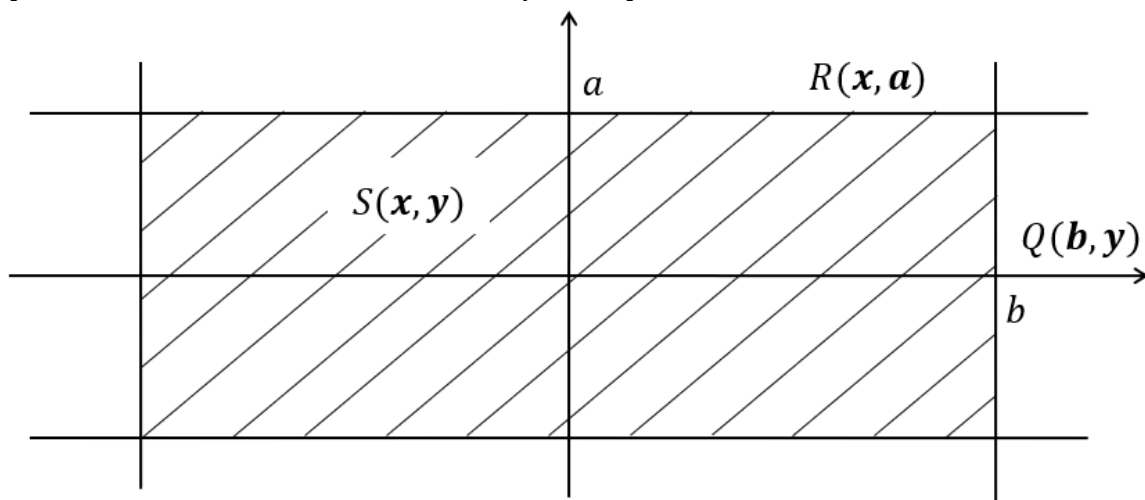


Рис. 5.1. Прямокутна область $ABCD$

Введемо поняття операції квантифікації над предикатами. Якщо предикат $P(x)$ означає, що x має властивість P , то:

- твердження «усі x мають властивість P » записується як $\forall x P(x)$;
- твердження «існує x , що має властивість P » записується як $\exists x P(x)$.

У цьому контексті:

- символ $\forall x$ називається *квантором загальності* (або *спільності*) за змінною x ;
- символ $\exists x$ – *квантором існування* за змінною x .

Визначення 5.5. Застосування квантора $\forall x_i$ ($1 \leq i \leq n$) до n -місного предиката $R(x_1, \dots, x_n)$ на множині M ($n \geq 1$) дає $(n - 1)$ -місний предикат $\forall x_i R(x_1, \dots, x_n)$ на множині M , який залежить від змінних $x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n$ і набуває значення 1 на заданому наборі значень змінних $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ тоді й тільки тоді, коли предикат $R(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ є тотожно-істинним за всіма можливими значеннями змінної $x_i \in M$.

Якщо $R(x_1)$ – одномісний предикат на множині M , то $\forall x_1 R(x_1)$ є нульмісним предикатом, або так званим *універсальним висловлюванням*, що відповідає предикату $R(x_1)$, яке істинне, якщо предикат $R(x_1)$ – тотожно-істинний, і хибне – у супротивному випадку.

Приклад 5.5

Нехай $R(x) = (2x + x = 3x)$ і $Q(x, y) = (x + y = 8)$ – предикати на множині цілих чисел. Тоді:

а) $\forall x R(x) = \forall x (2x + x = 3x)$ – істинне універсальне висловлювання, оскільки предикат $2x + x = 3x$ є тотожно-істинний;

б) $\forall y Q(x, y) = \forall y (x + y = 8)$ – одномісний предикат, що залежить від змінної x . Його значення

при $x = 2$:

$\forall y (2 + y = 8) = 0$, оскільки одномісний предикат $2 + y = 8$ не є тотожно-істинним,

при $x = 5$:

$\forall y (5 + y = 8) = 0$, оскільки предикат $5 + y = 8$ – тотожно-хибний;

в) $\forall x \forall y Q(x, y) = \forall x \forall y (x + y = 8)$ – хибне універсальне висловлювання, що відповідає одномісному предикату $\forall y (x + y = 8)$.

Визначення 5.6. Застосування квантора $\exists x_i$ ($1 \leq i \leq n$) до n -місного предиката $R(x_1, \dots, x_n)$ на множині M ($n \geq 1$) дає $(n - 1)$ -місний предикат $\exists x_i R(x_1, \dots, x_n)$ на M , що залежить від змінних $x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n$ і набуває значення 1 на цьому наборі $(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ значень змінних тоді й тільки тоді, коли предикат $R(a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_n)$ є виконаним, тобто існує хоча б одне значення $x_i \in M$, для якого він істинний.

Із цього визначення випливає, що для одномісного предиката $R(x_1)$ на множині M предикат $\exists x_1 R(x_1)$ є *нульмісним предикатом*, тобто *екзистенціальним висловлюванням*, що відповідає предикату $R(x_1)$.

– Це висловлювання істинне, якщо $R(x_1)$ є виконаним предикатом (тобто існує хоча б одне значення x_1 , для якого він істинний).

– Це висловлювання хибне, якщо $R(x_1)$ є тотожно-хибним предикатом (тобто не існує жодного значення x_1 , для якого він істинний).

Приклад 5.6

До предикатів прикладу 5.5 застосуємо квантор існування:

а) $\exists x R(x) = \exists x(2x + x = 3x)$ – істинне екзистенціальне висловлювання, оскільки предикат $2x + x = 3x$ є тотожно-істинним;

б) $\exists y Q(x, y) = \exists y(x + y = 8)$ – одномісний предикат, що залежить від змінної x .

Знайдемо деякі його значення

при $x = 5$:

$\exists y(5 + y = 8) = 1$, оскільки одномісний предикат $5 + y = 8$ виконаний (наприклад, при $y = 3$ він істинний);

в) $\exists x \exists y Q(x, y) = \exists x \exists y(x + y = 8)$ – істинне екзистенціальне висловлювання, що відповідає одномісному предикату $\exists y(x + y = 8)$;

г) $\forall y \exists x Q(x, y) = \forall y \exists x(x + y = 8)$ – універсальне висловлювання, що відповідає одномісному предикату $\exists x(x + y = 8)$, і воно також істинне, оскільки для кожного y існує відповідне x , яке задовольняє рівність.

Отже, $(n - 1)$ -місні предикати $\forall x_i R(x_1, \dots, x_n)$ та $\exists x_i R(x_1, \dots, x_n)$ ($1 \leq i \leq n$) залежать від змінної x_i , яка називається *зв'язаною*.

Змінна, що не є зв'язаною, називається *вільною*.

Прийнято вважати, що *квантори зв'язують сильніше*, ніж операції логіки висловлювань.

Наприклад, предикат $\exists x_1 (x_1 = x_2 \Rightarrow \forall x_2 (x_{12} + x_2 > 0))$ на множині натуральних чисел залежить від змінної x_1 , яка зв'язана квантором існування $\exists x_1$, і є результатом застосування квантора $\exists x_1$ до предиката-імплікації $x_1 = x_2$ та $\forall x_2 (x_{12} + x_2 > 0)$.

5.5. Основні властивості операцій квантифікації

Як основні властивості операцій квантифікації наведемо ряд доведених у [1, 4] теорем.

Теорема 5.1. Якщо n -місний предикат $P(x_1, \dots, x_n)$ заданий на скінченній множині $M = \{a_1, a_2, \dots, a_m\}$ ($n \geq 1, m \geq 1$), то $(n - 1)$ -місний предикат $\forall x_1 P(x_1, \dots, x_n)$ є рівносильним предикату

$$P(a_1, x_2, \dots, x_n) \wedge P(a_2, x_2, \dots, x_n) \wedge \dots \wedge P(a_m, x_2, \dots, x_n),$$

а предикат $\exists x_1 P(x_1, \dots, x_n)$ є рівносильним предикату

$$P(a_1, x_2, \dots, x_n) \vee P(a_2, x_2, \dots, x_n) \vee \dots \vee P(a_m, x_2, \dots, x_n).$$

Якщо ж множина M є нескінченною, то операції $\forall x$ і $\exists x$ є істотно новими, тобто не зводяться до операцій алгебри логіки.

Приклад 5.7

Нехай $Q(x, y)$ – двомісний предикат на множині $\{\alpha_1, \alpha_2, \alpha_3\}$. Тоді формула $\forall x \exists y Q(x, y)$ означає, що для кожного x із M існує хоча б один y із M , для якого $Q(x, y)$ істинне. Це розписується через логічні операції так:

$$\begin{aligned} \forall x \exists y Q(x, y) &\equiv \exists y Q(\alpha_1, y) \wedge \exists y Q(\alpha_2, y) \wedge \exists y Q(\alpha_3, y) \equiv \\ &\equiv (Q(\alpha_1, \alpha_1) \vee Q(\alpha_1, \alpha_2) \vee Q(\alpha_1, \alpha_3)) \wedge \\ &(Q(\alpha_2, \alpha_1) \vee Q(\alpha_2, \alpha_2) \vee Q(\alpha_2, \alpha_3)) \wedge \\ &(Q(\alpha_3, \alpha_1) \vee Q(\alpha_3, \alpha_2) \vee Q(\alpha_3, \alpha_3)). \end{aligned}$$

Теорема 5.2. $(n - 1)$ -місний предикат $\forall x_1 P(x_1, \dots, x_n)$ $(n \geq 2)$, заданий на множині M , є тотожно істинним тоді й тільки тоді, коли n -місний предикат $P(x_1, \dots, x_n)$ на множині M є тотожно істинним.

Теорема 5.3. $(n - 1)$ -місний предикат $\exists x_1 P(x_1, \dots, x_n)$ $(n \geq 2)$ на деякій множині M є тотожно хибним тоді й тільки тоді, коли n -місний предикат $P(x_1, \dots, x_n)$ на множині M є тотожно-хибним.

Приклад 5.8

Нехай M – множина натуральних чисел.

а) одномісні предикати $\forall m (\frac{m+n}{2} \geq \sqrt{mn})$ і $\forall n (\frac{m+n}{2} \geq \sqrt{mn})$ є тотожно істинними за теоремою 5.2, оскільки двомісний предикат $\frac{m+n}{2} \geq \sqrt{mn}$ є тотожно істинним на множині M ;

б) аналогічно за теоремою 5.3 предикати

$\exists x_1 (x_1 + x_2 < 0)$, $\exists x_2 (x_1 + x_2 < 0)$ є тотожно хибними, оскільки двомісний предикат $x_1 + x_2 < 0$ є тотожно хибним на множині M .

5.6. Тотожно-істинні предикатні формули

Як і в логіці висловлювань, для предикатних формул вводиться поняття тавтології, тобто **тотожно-істинної** чи **загальнозначущої формули**.

Визначення 5.7. Предикатна формула називається **тотожно істинною** (загальнозначущою) на множині M , якщо після заміни в ній простих формул будь-якими предикатами на M виходить тотожно істинний предикат на M . Формула, яка є тотожно істинною на будь-якій множині, називається **тотожно істинною** (загальнозначущою).

Приклад 5.9

Перевіримо, чи є загальнозначущою формула $\forall x R(x) \Rightarrow \exists x R(x)$.

Очевидно, у цій формулі немає вільних змінних, а є тільки одна одномісна проста формула $R(x)$. Тому потрібно перевірити, чи є вихідна формула тотожно істинною на множині з двох елементів $\{a, b\}$. Використовуючи еквівалентну формулу $R(a) \wedge R(b) \Rightarrow R(a) \vee R(b)$ і покладаючи $R(a) = p$, $R(b) = q$, отримуємо пропозиційну формулу $p \wedge q \Rightarrow p \vee q$, тобто тавтологію. Отже, вихідна формула є загальнозначущою.

У логіці предикатів немає загального алгоритму для з'ясування того, чи є будь-яка предикатна формула загальнозначущою. Більше того, А. Черч у 1936 р. довів теорему [8] про те, що не існує способу, відповідно до якого для будь-якої предикатної формули за скінченне число кроків можна вирішити, чи є вона загальнозначущою.

5.7. Числення предикатів

На підставі аксіоматичної формалізації теорії загальнозначущих формул логіки предикатів будується **числення предикатів**, аналогічне численню висловлювань.

Визначення виведення з гіпотез знаходять з відповідних визначень числення висловлювань шляхом додавання до них четвертої і п'ятої схем аксіом та використання двох правил виведення.

Схеми аксіом:

1) $\forall u A(u) \Rightarrow A(t)$,

де $A(u)$ – формула, t – змінна, вільна для u у $A(u)$. Змінна t може збігатися з u .

2) $\forall u (A \Rightarrow B) \Rightarrow (A \Rightarrow \forall u B)$, якщо A не містить вільно u ;

Правила виведення:

1) **modus ponens**: з $A \Rightarrow B$ і A випливає B ;

2) **правило узагальнення**: з A випливає $\forall u A$, де u – змінна.

Властивості числення предикатів

1. Доводиться [1], що числення предикатів є *несуперечливим*. Для будь-якої формули A буде неправильним, якщо одночасно виконуються $\vdash A$ і $\vdash \neg A$.

2. Воно є *повним*, тобто всяка загальнозначуща формула доводиться в численні предикатів (теорема Геделя).

5.8. Вправи до розділу

1. Наведіть по два приклади предикатів різного типу: тотожно-істинних, тотожно-хибних, здійснених, предикатів-наслідків.

2. Виведіть предикатні формули, рівносильні поданим формулам, такі, що не містять кванторів. Усі предикати визначені на множині $\{a, b, c\}$:

а) $\forall x \exists y (P(x, y) \rightarrow \neg Q(x))$;

б) $\exists x \neg (P(x, y) \vee \forall y R(y))$;

в) $\exists y \forall x (P(x, y) \rightarrow Q(y))$;

г) $\exists x \forall y (P(x) \rightarrow Q(x, y))$;

д) $\forall y \neg (R(y) \wedge \exists x S(x, y))$;

е) $\forall x \neg (Q(x, y) \wedge P(y)) \vee \exists x R(x)$.

3. Запишіть у предикативній формі такі фрази:

а) усі голодні крокодили не налаштовані дружньо;

б) жоден крокодил не налаштований дружньо, коли він голодний.

Предикати визначити на множині крокодилів, увести ознаки «бути голодним» і «бути дружнім».

4. Опишіть геометричні області, що зображені на рис. 5.2–5.25, за допомогою предикатних формул.

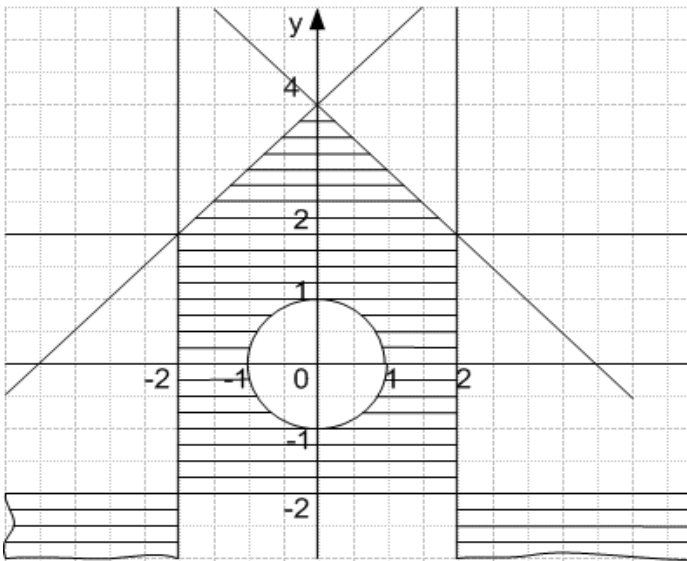


Рис. 5.2

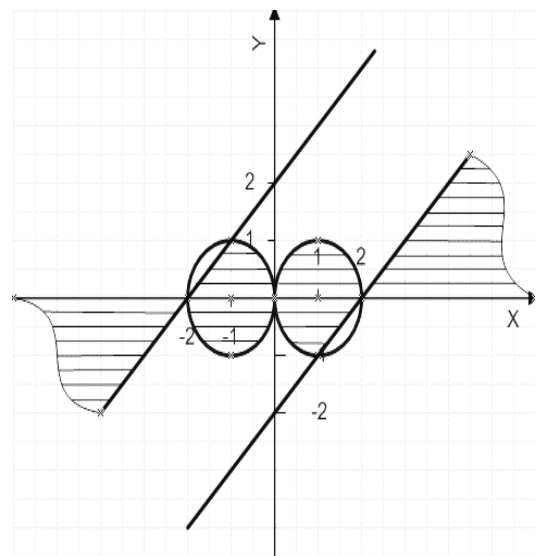


Рис. 5.3

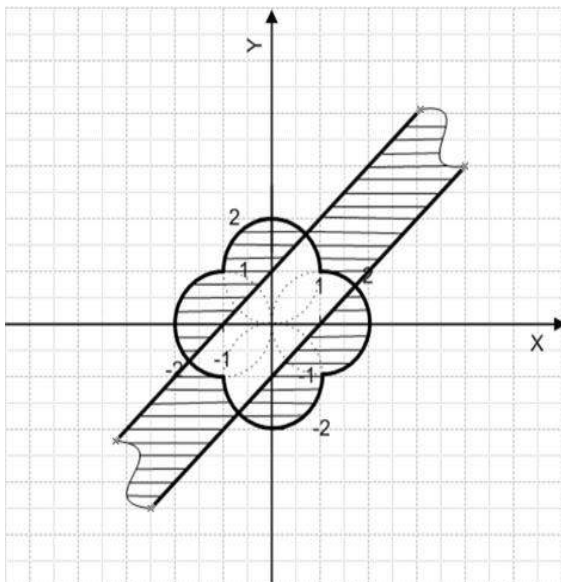


Рис. 5.4

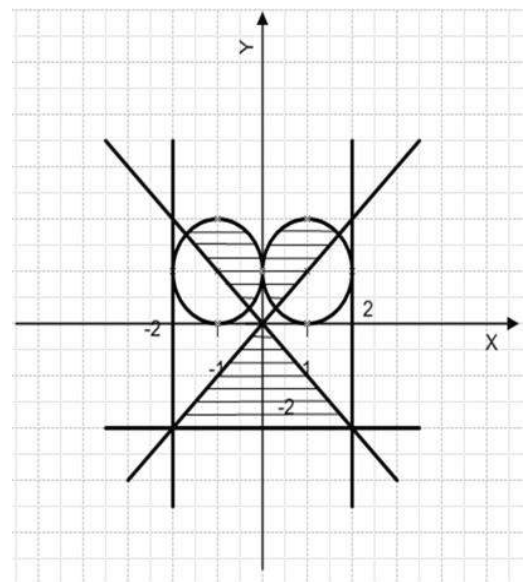


Рис. 5.5

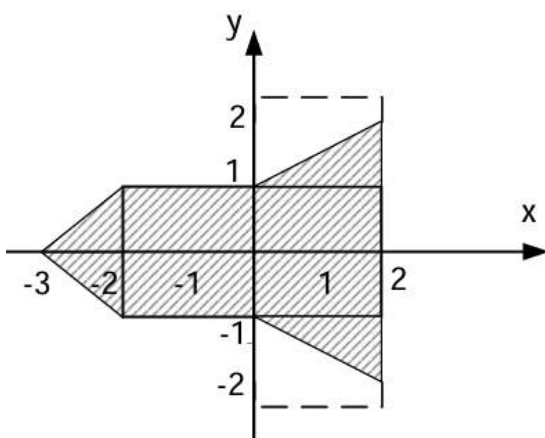


Рис. 5.6

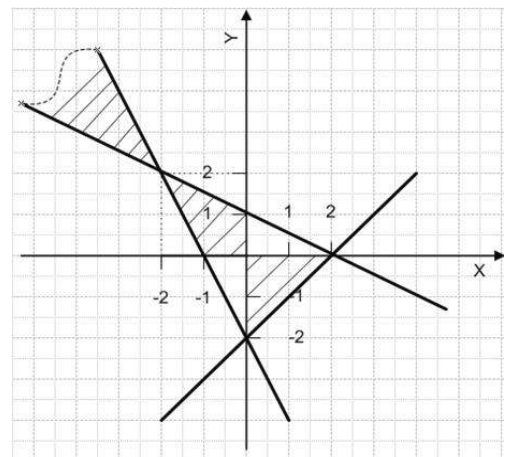


Рис. 5.7

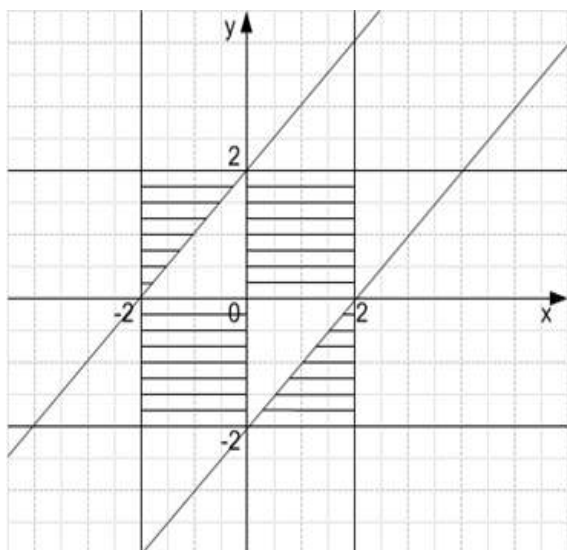


Рис. 5.8

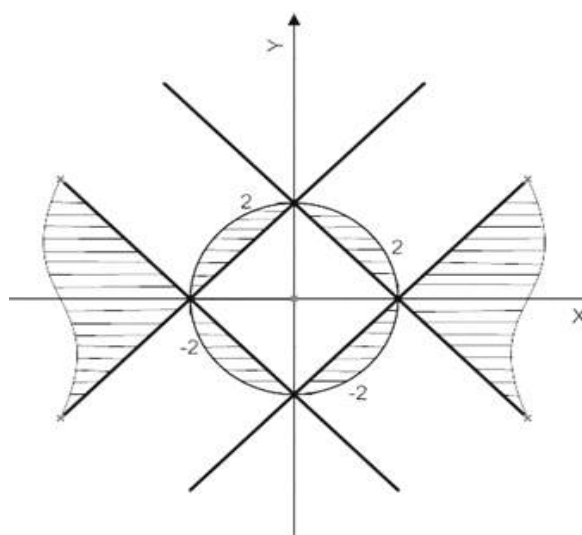


Рис. 5.9

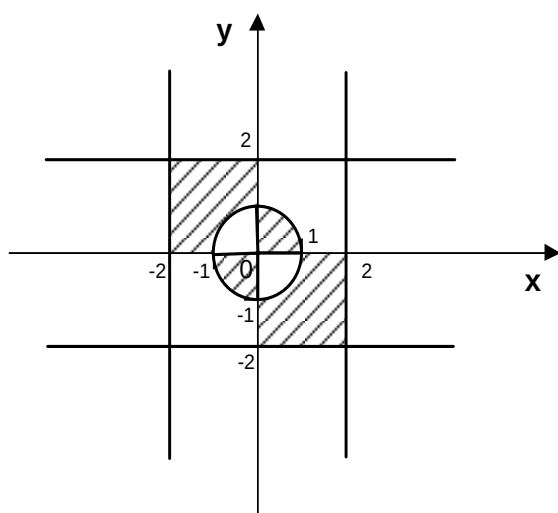


Рис. 5.10

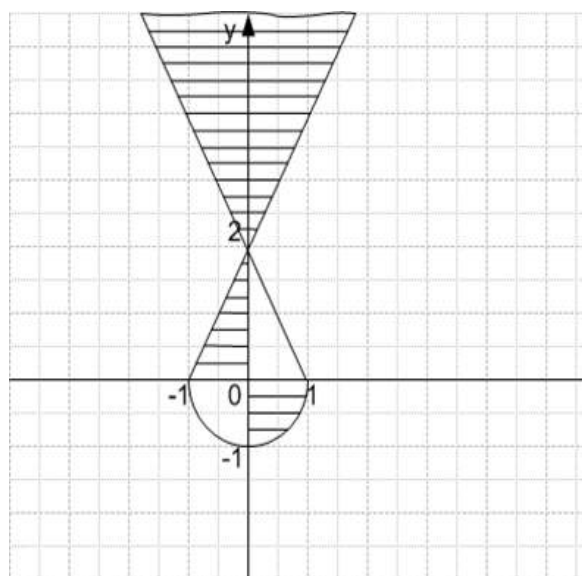


Рис. 5.11

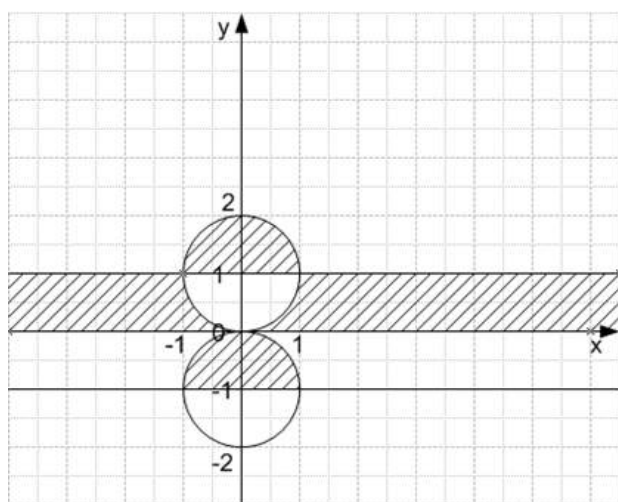


Рис. 5.12

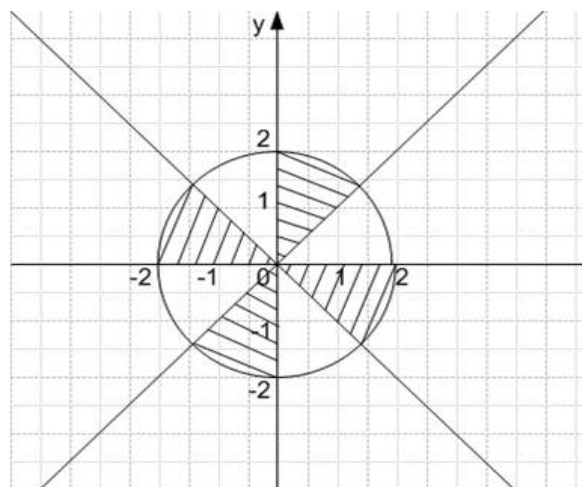


Рис. 5.13

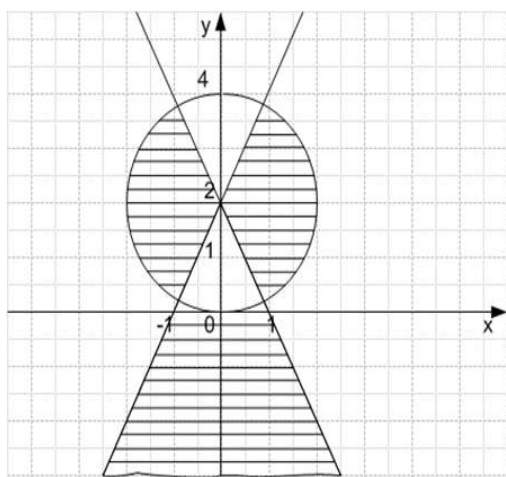


Рис. 5.14

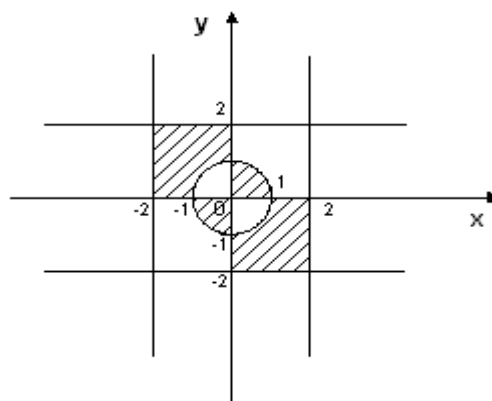


Рис. 5.15

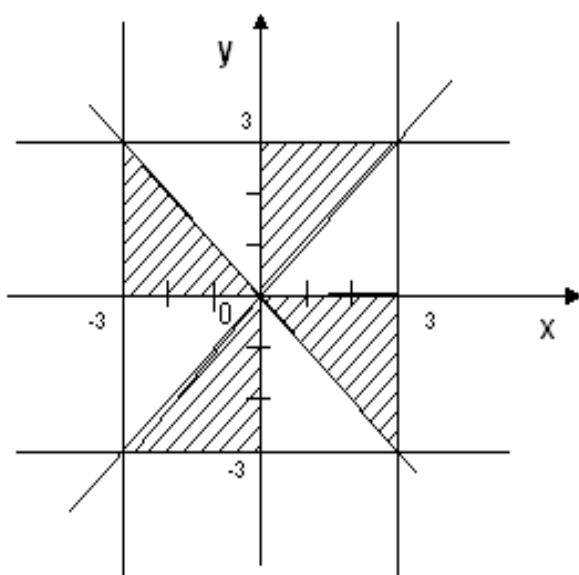


Рис. 5.16

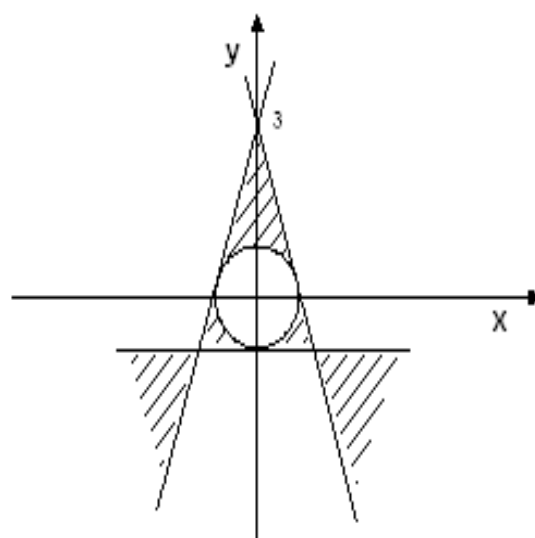


Рис. 5.17

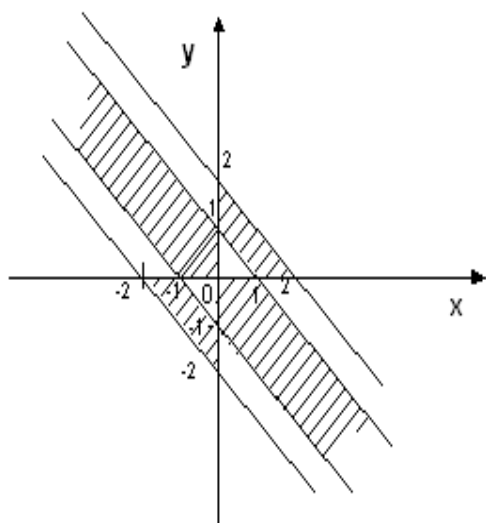


Рис. 5.18

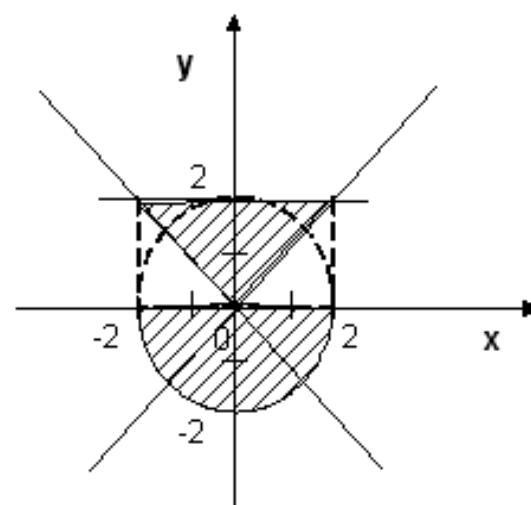


Рис. 5.19

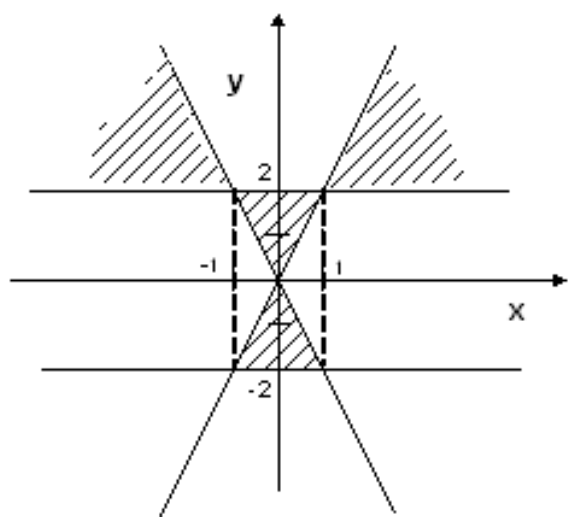


Рис. 5.20

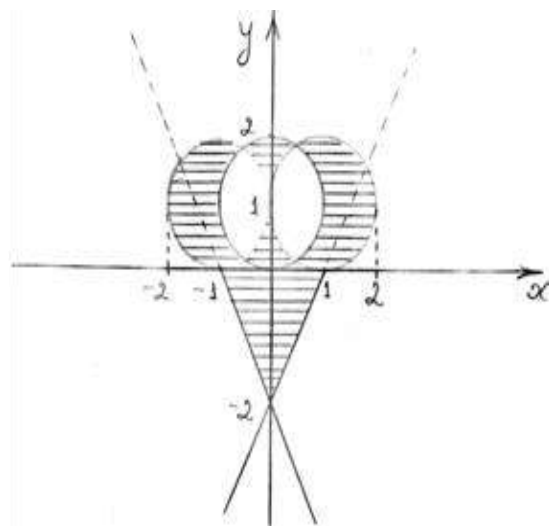


Рис. 5.21

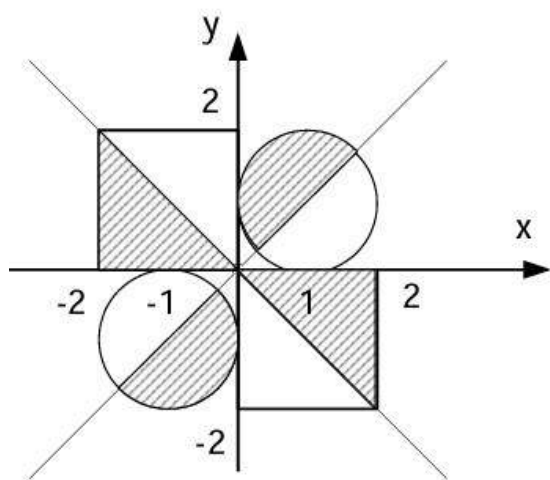


Рис. 5.22

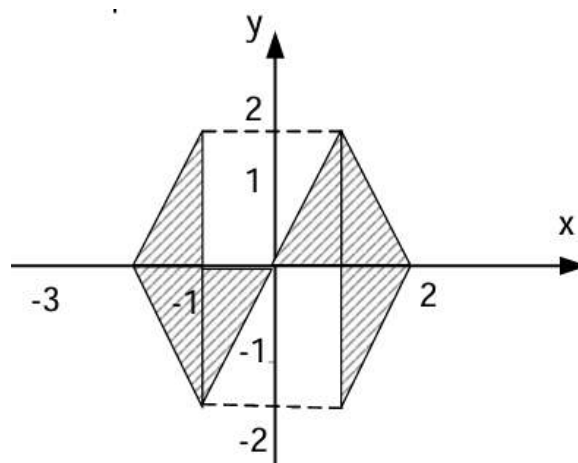


Рис. 5.23

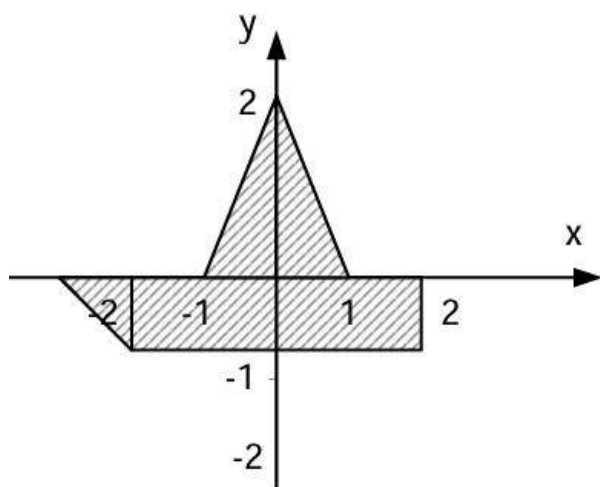


Рис. 5.24

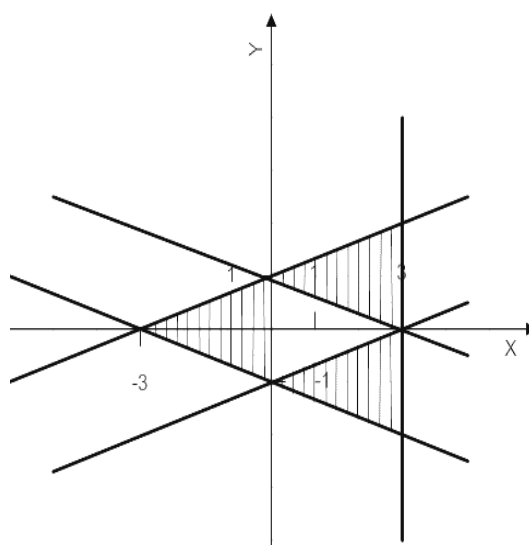


Рис. 5.25

5. Визначити, чи може предикат R бути виконаним, але не тотожно істинним:

- а) $\forall x \forall y (P(x, y) \leftrightarrow \exists y Q(x, y))$;
- б) $\forall x \exists y P(x, y) \rightarrow \exists y \forall x Q(x, y, z)$;
- в) $\exists z \forall u P(x, z) \wedge \forall v Q(v, y)$;
- г) $\forall x \exists y (P(x, y) \wedge Q(y, z))$;
- д) $\forall x \exists y P(x, y) \vee Q(x, z)$;
- е) $\exists x P(x, z) \rightarrow \forall x \exists y Q(y, x)$.

6. Визначити тип і місткість предиката, кожний аргумент якого набуває значень з множини M :

- а) $\forall x (x + y - z > 2), M = \mathbb{Z}$;
- б) $\exists x \forall y (x y = z - x), M = \mathbb{R}$;
- в) $\forall x \exists y (x y z = y z), M = \mathbb{N}$;
- г) $\forall x \exists y (z x = y^2), M = \mathbb{N}$;
- д) $\exists y (y + z > x + 2y), M = \mathbb{Z}$;
- е) $\forall x (x^2 - y^2 > x + y), M = \mathbb{N}$.

7. Записати у предикативній формі такі фрази:

- а) жодна нецивілізована нація не войовнича;
- б) деякі нації не войовничі.

Предикати визначити на множині націй, увести ознаки «бути цивілізованим» і «бути войовничим».

8. Записати у предикативній формі такі фрази:

- а) жодний старанний студент не вчиться погано;
- б) жодний студент не старанний.

Предикати визначити на множині студентів, увести ознаки «бути старанним» і «вчитися добре».

9. Записати у предикативній формі такі фрази:

- а) корисних слив не існує;
- б) деякі сливи корисні.

Предикати визначити на множині фруктів, увести ознаки «бути сливою» і «бути корисним».

10. Записати формулою логіки предикатів таке твердження:

- а) для кожного x $P(x)$ – хибне;
- б) $P(x)$ – хибне для кожного x .

11. Указати вільні та зв'язані змінні у виразі. Для кожної зв'язаної змінної визначити, яким саме квантором її зв'язано:

- а) $P(x) \rightarrow \forall y (Q(y) \vee \exists z P(z) \leftrightarrow \neg P(y))$;
- б) $\exists x(P(y) \rightarrow P(x) \wedge \exists z(Q(z) \leftrightarrow \exists y(Q(y) \vee Q(x))) \wedge \forall x(P(x) \rightarrow \neg (\exists y Q(y))))$;
- в) $\forall x (x^2 y > 0 \rightarrow y > 0)$;
- г) $x > 3 \wedge \forall x \forall y (xy^2 > 0)$;
- д) $\forall x \forall y (x < y \rightarrow \exists z (x < z \wedge z < y))$.

12. Предметна область – це множина $\mathbb{R}$ дійсних чисел. Визначити, є вираз висловлюванням чи пропозиційною формою. У першому випадку вказати, істинним чи хибним є висловлювання, у другому – навісити квантори так, щоб отримати істинне висловлювання:

- а) $\exists y \forall x (xz = xy)$;
- б) $\exists p \forall x (x^2 + px + q > 0)$;
- в) $\exists x \forall y \exists z (xy = z)$;
- г) $\exists y \forall x (xy > 0) \rightarrow \exists z (z^2 < 0)$.

13. Нехай предметною областю є множина $\mathbb{N}$ натуральних чисел. Проаналізувати область дії кванторів і визначити істинність висловлювань:

- а) $\forall x \forall y (\exists z (z > x \wedge z < y) \rightarrow x < y)$;
- б) $\forall x \forall y \exists z ((z > x \wedge z < y) \rightarrow x < y)$.

14. Навести приклади тверджень математичного та нематематичного змісту, що містять кванторні вирази. Для кожного прикладу проаналізувати, як змінюється істинність твердження у разі зміні порядку кванторів.

15. Проаналізувати області дії кванторів і порівняти істинність таких виразів:

$$\forall x (x > 3) \rightarrow (2 > 3) \text{ та } \forall x ((x > 3) \rightarrow (2 > 3)).$$

16. Довести, що формула предикатної логіки

$$\forall x \exists y (P(x) \vee \neg P(y))$$

є логічно загальнозначущою (тобто істинною за будь-якої інтерпретації).

Проаналізувати, чи є логічно загальнозначущою формула

$$\exists y \forall x (P(x) \vee \neg P(y)).$$

17. Перевірити, чи є наведені формули тотожно істинними (тобто істинними за будь-якої інтерпретації):

- а) $\exists x (P(x) \wedge Q(x)) \rightarrow (\exists x P(x) \wedge \exists x Q(x))$;
- б) $(\forall x P(x) \rightarrow \exists x Q(x)) \equiv \exists x (P(x) \rightarrow Q(x))$.

18. Для кожної з наведених формул визначити, чи є вона тотожно істинною (тобто істинною за будь-якої інтерпретації). Довести або спростувати:

- а) $\exists xP(x) \wedge \exists x(P(x) \rightarrow Q(x)) \rightarrow \exists xQ(x)$;
- б) $\exists xP(x) \wedge \forall x(P(x) \rightarrow Q(x)) \rightarrow \exists xQ(x)$;
- в) $\forall xP(x) \wedge \exists x(P(x) \rightarrow Q(x)) \rightarrow \exists xQ(x)$;
- г) $\forall x(P(x) \rightarrow (Q(x) \rightarrow R(x))) \rightarrow \forall x(P(x) \vee Q(x) \rightarrow R(x))$.

19. Довести, що наведені формули логіки предикатів є рівносильними:

- а) $A = \exists x(P(x) \vee Q(x)), B = \exists xP(x) \vee \exists xQ(x)$;
- б) $A = \neg(\forall xP(x)), B = \exists x(\neg P(x))$.

20. Побудувавши відповідну інтерпретацію (контрприклад), довести, що наведені формули логіки предикатів не є рівносильними:

- а) $A = \exists x(P(x) \wedge Q(x)), B = \exists xP(x) \wedge \exists xQ(x)$;
- б) $A = \forall x(P(x) \vee Q(x)), B = \forall xP(x) \vee \forall xQ(x)$;
- в) $A = \exists xP(x) \rightarrow \exists xQ(x), B = \exists x(P(x) \rightarrow Q(x))$.

21. Довести або спростувати твердження про рівносильність наведених формул логіки предикатів:

- а) $A = \forall y\forall x(P(x) \rightarrow Q(y)), B = \forall y(\forall xP(x) \rightarrow Q(y))$;
- б) $A = \forall x(P(x) \rightarrow Q(y)), B = \forall xP(x) \rightarrow Q(y)$;
- в) $A = \exists x(P(x) \rightarrow Q(y)), B = \exists xP(x) \rightarrow Q(y)$.

22. Побудувати префіксну форму для кожної з наведених формул:

- а) $\forall xA(x) \rightarrow \forall y(\exists zB(x, y, z) \rightarrow \neg\forall xA(x) \wedge \exists xC(x, y))$;
- б) $\forall x\neg\exists yA(x, y) \rightarrow \forall xB(x) \rightarrow \neg\exists yA(x, y)$;
- в) $\neg\forall xA(x) \wedge \exists xB(x) \vee \forall x(\forall yC(x, y) \rightarrow A(y))$;
- г) $\forall xA(x) \rightarrow \forall y(\forall zB(x, y, z) \rightarrow \neg\forall xA(x))$.

23. Визначити, які з тверджень 1–5 є логічно еквівалентні:

- а) Неправильно, що всі числа, кратні 4, є точними квадратами.
- б) Усі числа, кратні 4, не є точними квадратами.
- в) Не всі числа, кратні 4, є точними квадратами.
- г) Існує число, кратне 4, яке не є точним квадратом.
- д) Деякі числа, не кратні 4, не є точними квадратами.

У подальших вправах перевірити, чи впливають у логіці предикатів із припущень зроблені з них висновки, тобто чи коректними були здійснені міркування.

Якщо міркування правильне, то побудувати відповідний дедуктивний ланцюжок; якщо ж міркування некоректне, то побудувати відповідний контрприклад.

24. Довести, що:

- а) кожне число, кратне 51, кратне 17 і кратне 3;
- б) кожне число кратне 3 тільки тоді, коли сума цифр його запису в десятковій системі числення кратна 3;
- в) сума цифр запису числа 10 712 не кратна 3. Отже, 10 712 не кратне 51.

25. Довести:

- а) Якщо хтось може розв'язати певну задачу, то знайдеться здібний студент, який зробить це.
- б) Петренко – здібний студент, але не зміг розв'язати цю задачу. Отже, ніхто не здатен розв'язати цю задачу.

5.9. Приклад програмної реалізації вправ

Завдання: написати програму, яка перевіряє, чи належить точка заданій заштрихованій області на рис. 5.26.

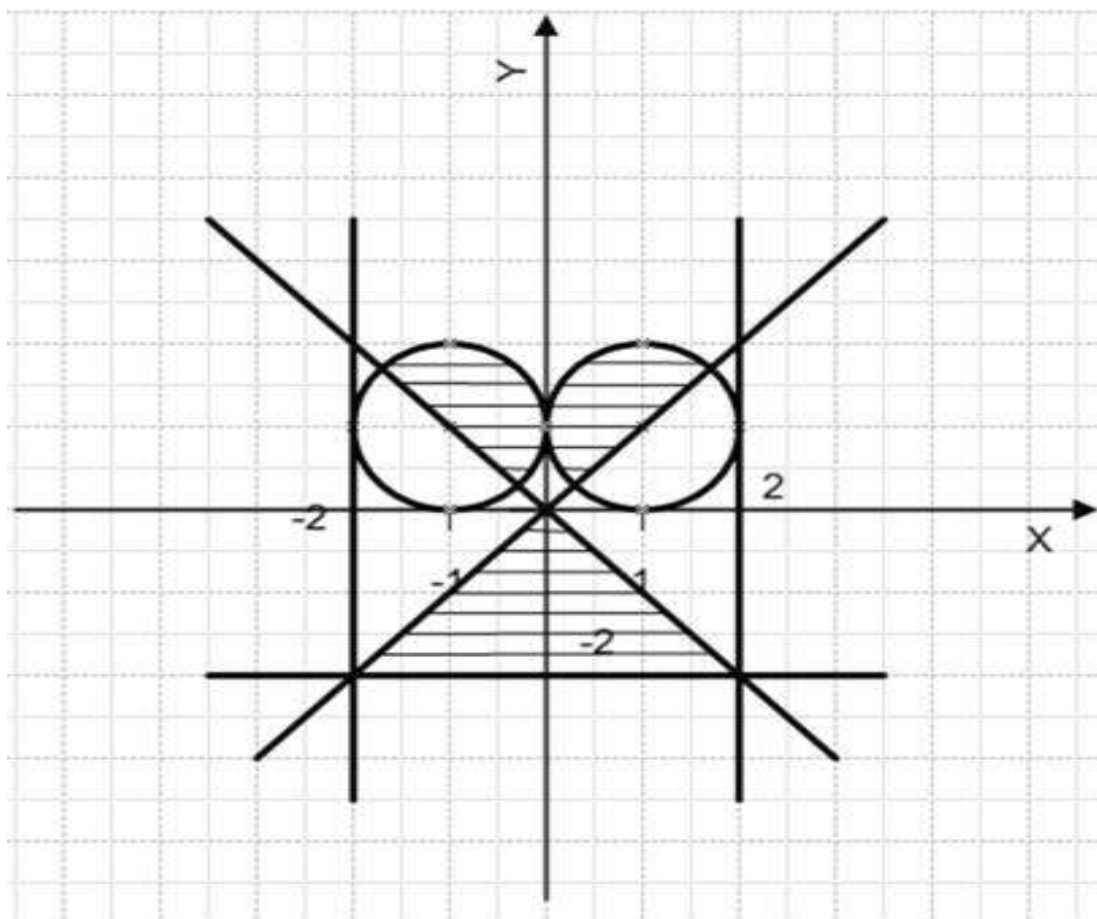


Рис. 5.26. Заштрихована область допустимих значень

У результаті виконання завдання була написана програма, яка виконує інтерактивну перевірку належності точки, заданої координатами (x, y) , складній геометричній області, що складається з кількох фігур. Програма також виконує візуалізацію цієї області, її меж та введеної користувачем точки.

Структура програми

Програма має модульну структуру і використовує зовнішні бібліотеки для математичних обчислень та побудови графіків.

Основні бібліотеки:

- *NumPy*: використовується для виконання математичних операцій, зокрема для роботи з масивами координат при побудові графіка (векторизація обчислень);

- *Matplotlib*: є інструментом для візуалізації, застосовується для зафарбовування областей, побудови графіків функцій, нанесення точок та налаштування зовнішнього вигляду графіка.

Опис функцій

Для розв'язання задачі було написано три ключові функції.

1. *is_in_shaded_region(x, y)*

Призначення: основна логічна функція, яка є математичним ядром програми.

Вона перевіряє, чи задовольняє точка з координатами (x, y) системі нерівностей, що описує заштриховану область.

Принцип роботи: функція приймає координати x та y , усередині неї реалізовано дві умови:

- перевірка належності до *нижнього трикутника*:

$(y \leq -|x|) \text{ and } (y \geq -2);$

- перевірка належності до *верхньої частини заштрихованої області*:

$(y \geq |x|) \text{ and } (((x + 1)^2 + (y - 1)^2 \leq 1) \text{ or } ((x - 1)^2 + (y - 1)^2 \leq 1)).$

Результати цих перевірок об'єднуються логічним оператором АБО ($|$).

Функція повертає: *True*, якщо точка належить хоча б одній з областей, і *False* в іншому випадку.

Вона оптимізована для роботи з масивами *NumPy*, що прискорює побудову графіка.

2. *plot_region_and_point(user_x, user_y)*

Призначення: відповідає за всю візуальну частину, а саме створення графіка, зафарбовування областей, побудову обмежувальних ліній та нанесення точки.

Принцип роботи: створює сітку координат (*meshgrid*) для всієї області графіка; використовуючи функцію *is_in_shaded_region*, обчислює, які точки сітки належать областям, і зафарбовує їх за допомогою методу *contourf*.

Далі послідовно малює всі обмежувальні та допоміжні лінії ($y = |x|, y = -|x|, y = -2$, кола тощо), задаючи для кожної унікальний колір та стиль.

Наприкінці наносить на графік червону точку, що відповідає координатам, які ввів користувач.

3. *main()*

Призначення: головна функція, що керує життєвим циклом програми та організовує діалог з користувачем.

Принцип роботи: запускає нескінченний цикл для запиту координат у користувача. Для захисту від некоректного вводу (наприклад, введення тексту замість числа) використовується блок *try – except*.

Після отримання коректних даних функція викликає *is_in_shaded_region* для перевірки, виводить текстовий результат («Належить» або «Не належить») на екран, друкує опис заштрихованих областей і, наостанок, викликає *plot_region_and_point* для візуалізації.

Лістинг програми

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle

def is_in_shaded_region(x, y):
    """
    Перевіряє, чи належить точка (x, y) до заштрихованої області.
    """
    # Умова для нижньої трикутної області
    in_triangle = (y <= -np.abs(x)) & (y >= -2)
    # Умова для верхньої області
    is_above_v_shape = y >= np.abs(x)
    is_in_left_circle = (x + 1)**2 + (y - 1)**2 <= 1
    is_in_right_circle = (x - 1)**2 + (y - 1)**2 <= 1
    in_lenses = is_above_v_shape & (is_in_left_circle |
is_in_right_circle)
    # Точка належить області, якщо вона в трикутнику АБО в "лінзах"
    return in_triangle | in_lenses

def plot_region_and_point(user_x=None, user_y=None):
    """
    Будує графік з двома заштрихованими областями, лініями та
    червоною точкою.
    """
    # Створення сітки точок
    x_coords = np.linspace(-4, 4, 500)
    y_coords = np.linspace(-4, 4, 500)
    X, Y = np.meshgrid(x_coords, y_coords)
    # --- Умови для заштрихованих областей ---
    # 1. Нижня трикутна область
    Z_triangle_region = (Y <= -np.abs(X)) & (Y >= -2)
    # 2. Верхня область "лінз"
    is_above_v_shape = Y >= np.abs(X)
    is_in_left_circle = (X + 1)**2 + (Y - 1)**2 <= 1
    is_in_right_circle = (X - 1)**2 + (Y - 1)**2 <= 1
    Z_lenses_region = is_above_v_shape & (is_in_left_circle |
is_in_right_circle)
    # Створення графіка
    fig, ax = plt.subplots(figsize=(8, 8))
    # Зафарбовування обох областей
```



```

    ax.contourf(X, Y, Z_triangle_region, levels=[0.5, 1.5],
colors=['#4682B4'], alpha=0.8)
    ax.contourf(X, Y, Z_lenses_region, levels=[0.5, 1.5],
colors=['#4682B4'], alpha=0.8)
    # -- Малюємо обмежуючі та допоміжні лінії різними кольорами ---
    line_x_vals = np.linspace(-4, 4, 400)
    ax.plot(line_x_vals, np.abs(line_x_vals), color='green',
linestyle='--', linewidth=1.5, label='y = |x|')
    ax.plot(line_x_vals, -np.abs(line_x_vals), color='purple',
linestyle='--', linewidth=1.5, label='y = -|x|')
    ax.axhline(-2, color='red', linestyle='--', linewidth=1.5,
label='y = -2')
    ax.axvline(-2, color='blue', linestyle='--', linewidth=1.5,
label='x = -2')
    ax.axvline(2, color='orange', linestyle='--', linewidth=1.5,
label='x = 2')
    circle1 = Circle((-1, 1), 1, color='magenta', fill=False,
linestyle='--', linewidth=1.5)
    circle2 = Circle((1, 1), 1, color='cyan', fill=False,
linestyle='--', linewidth=1.5)
    ax.add_patch(circle1)
    ax.add_patch(circle2)
    # Виправлений рядок, що не викликає попередження
    ax.plot([], [], '--', color='brown', label='(x±1)²+(y-1)² = 1')
    # Налаштування вигляду графіка
    ax.set_aspect('equal', adjustable='box')
    ax.grid(True, linestyle=':', alpha=0.6)
    ax.axhline(0, color='black', linewidth=0.7)
    ax.axvline(0, color='black', linewidth=0.7)
    ax.set_xlim(-4, 4)
    ax.set_ylim(-4, 4)
    ax.set_xlabel("Вісь X", fontsize=12)
    ax.set_ylabel("Вісь Y", fontsize=12)
    ax.set_title("Графічне представлення області та її меж",
fontsize=14)
    # Малюємо точку користувача червоним кольором
    if user_x is not None and user_y is not None:
        ax.plot(user_x, user_y, 'ro', markersize=8, label=f'Ваша
точка ({user_x}, {user_y})')
    ax.legend(loc='upper right', fontsize=9)
    plt.show()
def main():
    """
    Головна функція для взаємодії з користувачем.
    """
    print("--- Перевірка належності точки заштрихованій області ---")
    while True:
        try:
            x_input = float(input("Введіть координату X: "))
            y_input = float(input("Введіть координату Y: "))
            break
        except ValueError:
            print("Помилка! Будь ласка, введіть дійсні числа.")

```

```

# Перевірка належності до фінальної заштрихованої області
if is_in_shaded_region(x_input, y_input):
    print(f"\n Результат: Точка ({x_input}, {y_input}) НАЛЕЖИТЬ
заштрихованій області.")
else:
    print(f"\n Результат: Точка ({x_input}, {y_input}) НЕ
НАЛЕЖИТЬ заштрихованій області.")
    print("\nЗараз буде відображено графік...")
    plot_region_and_point(x_input, y_input)
if __name__ == "__main__":
    main()

```

Тестування працездатності програми

Щоб переконатися у коректності роботи програми, було проведено тестування, що охоплює перевірку основної логіки, обробку межових значень та поведінку в разі некоректного введення даних.

Виконаємо тести, які підтверджують, що програма коректно ідентифікує точки всередині, ззовні та на межах кожної з двох складних областей:

– перевірка, що програма коректно ідентифікує точки, які гарантовано лежать всередині заштрихованих областей: точка (0, -1.5) всередині нижнього трикутника (рис. 5.27);

--- Перевірка належності точки заштрихованій області -

Введіть координату X: 0

Введіть координату Y: -1.5

Результат: Точка (0.0, -1.5) НАЛЕЖИТЬ заштрихованій області.

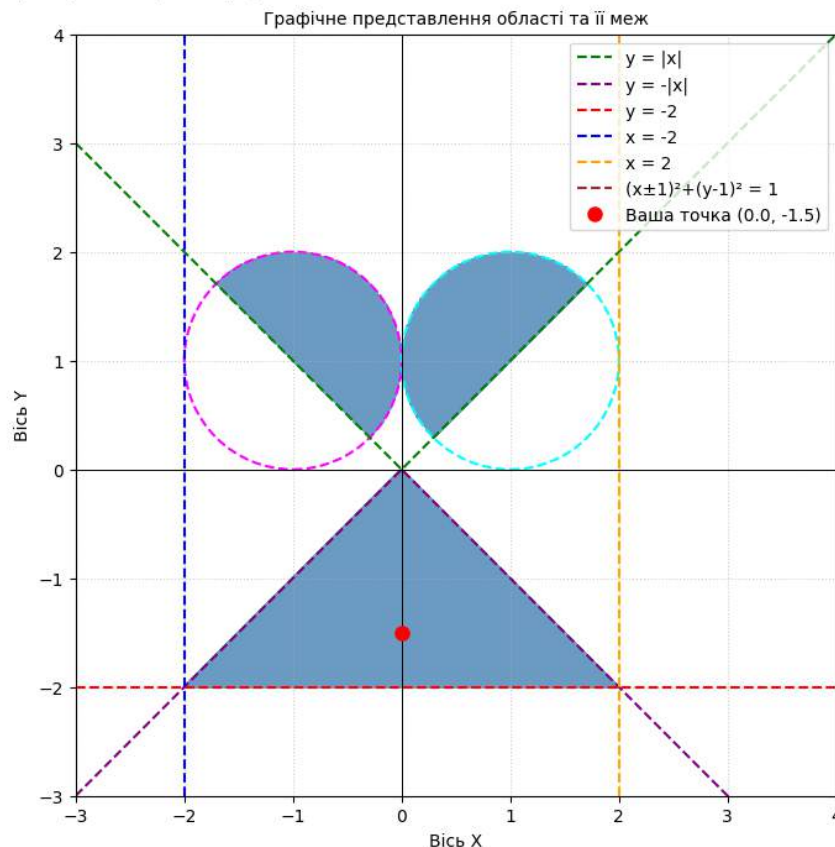


Рис. 5.27. Перевірка внутрішніх точок

– перевірка, що точка (0.5, 1.0), яка гарантовано лежить всередині верхньої заштрихованої області, утвореної криволінійними дугами кругових секторів (рис. 5.28).

--- Перевірка належності точки заштрихованій області -
Введіть координату X: 0.5
Введіть координату Y: 1.0

Результат: Точка (0.5, 1.0) НАЛЕЖИТЬ заштрихованій області.

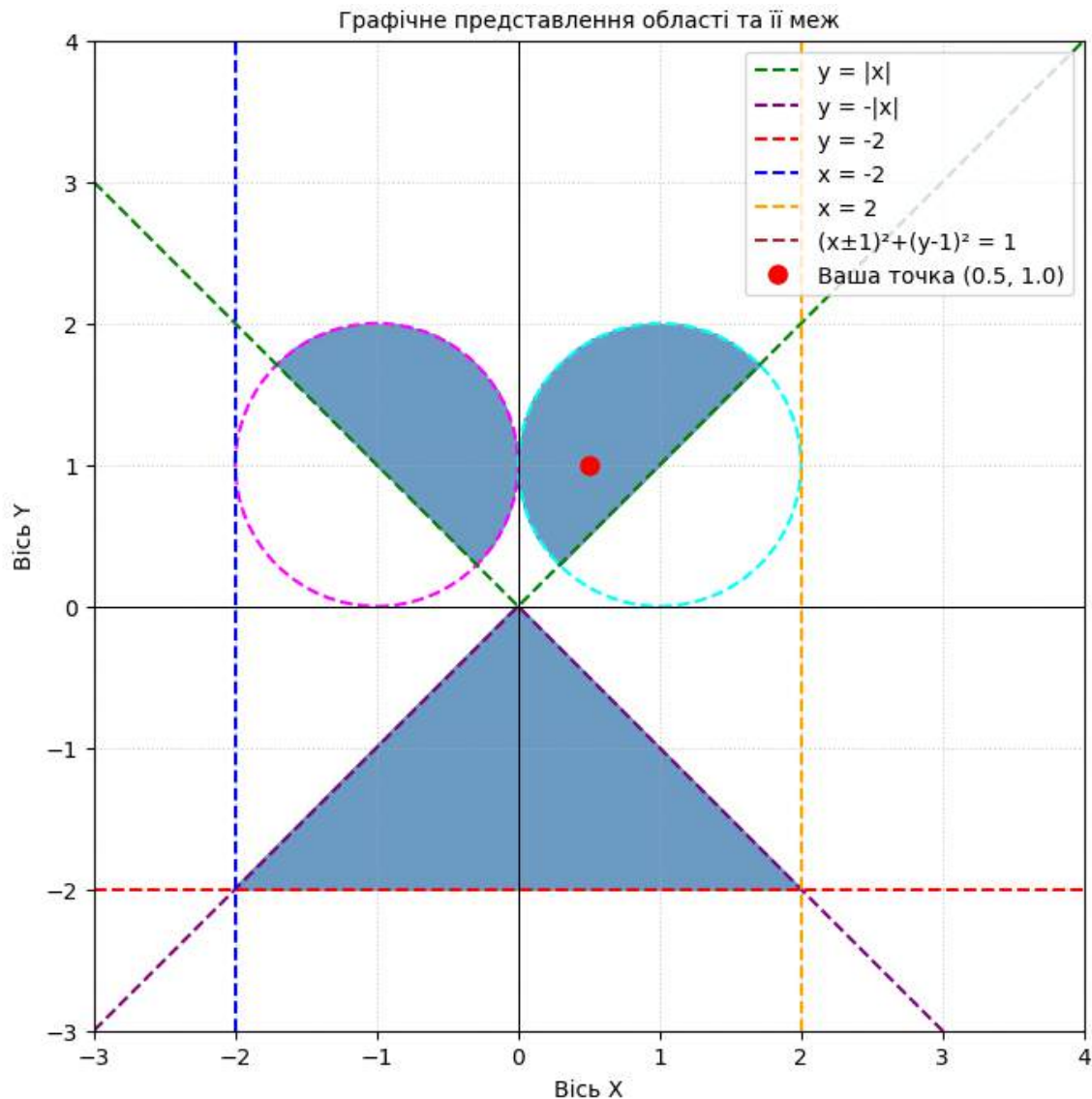


Рис. 5.28. Перевірка внутрішніх точок

Результати тестування на рис. 5.27 та рис. 5.28 підтверджують безпомилкову ідентифікацію точок у межах обох типів геометричних фігур. Алгоритм точно опрацьовує як лінійні межі трикутного сегмента (точка (0.0, -1.5)), так і криволінійні дуги кругових секторів (точка (0.5, 1.0)), що доводить коректність реалізації комбінованої математичної моделі. Відповідність візуального положення червоних маркерів розрахунковим даним свідчить про повну синхронізацію графічного модуля з логічною частиною програми.

– перевірка, що програма правильно класифікує точки, які знаходяться поза межами областей (рис. 5.29-5.30);

--- Перевірка належності точки заштрихованій області -
Введіть координату X: 0
Введіть координату Y: 0.5

Результат: Точка (0.0, 0.5) НЕ НАЛЕЖИТЬ заштрихованій області.

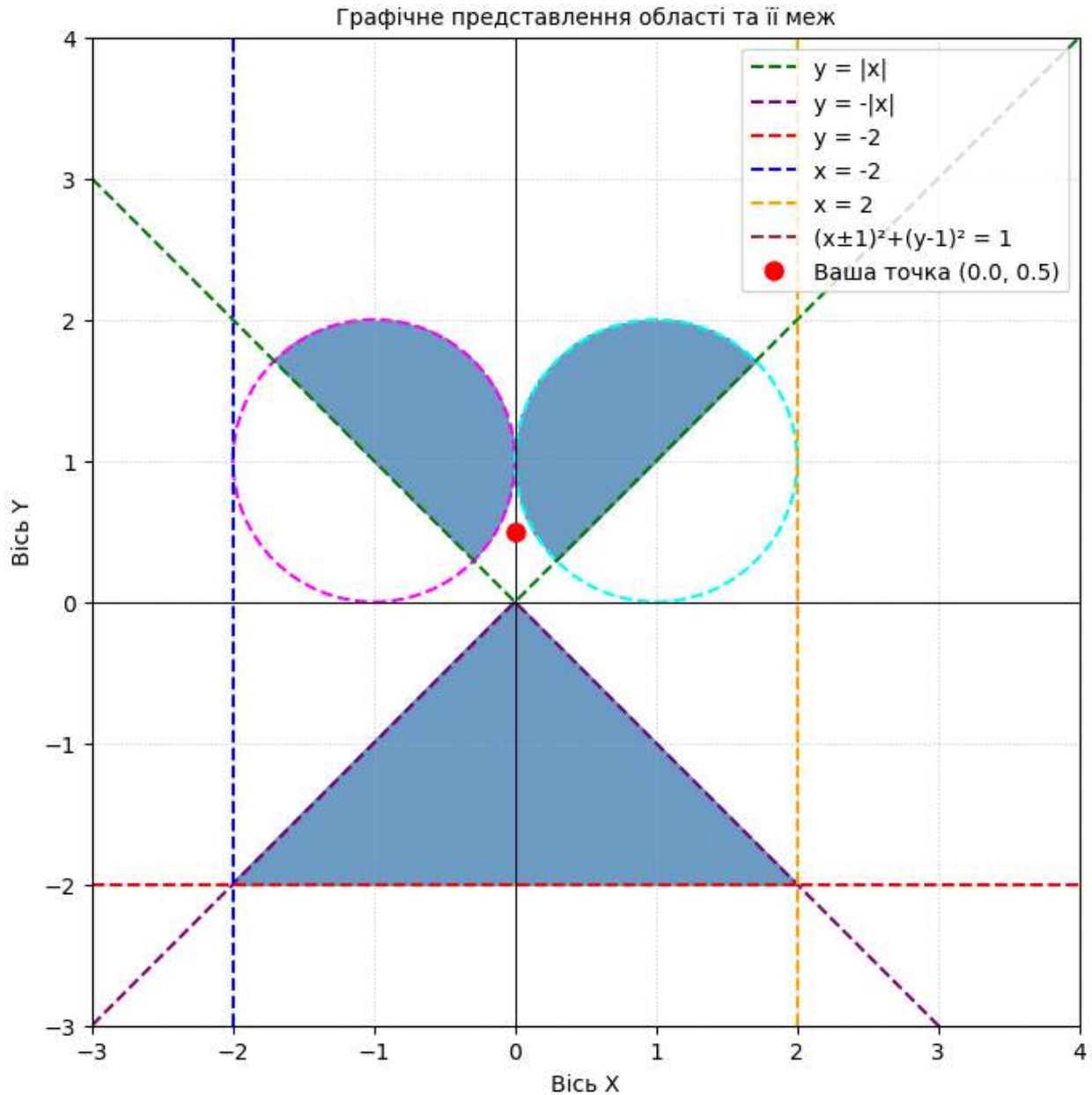


Рис. 5.29. Перевірка зовнішніх точок (між областями)

Візуальний аналіз поданих даних (рис. 5.29) підтверджує, що обрана точка (0.0,0.5) знаходиться у вільному просторі між верхніми сегментами фігури. Це ілюструє високу точність налаштованих логічних умов, які дозволяють програмі чітко розрізняти межі заштрихованих зон навіть за мінімальної відстані від них.

--- Перевірка належності точки заштрихованій області -
Введіть координату X: 10
Введіть координату Y: 10

Результат: Точка (10.0, 10.0) НЕ НАЛЕЖИТЬ заштрихованій області.

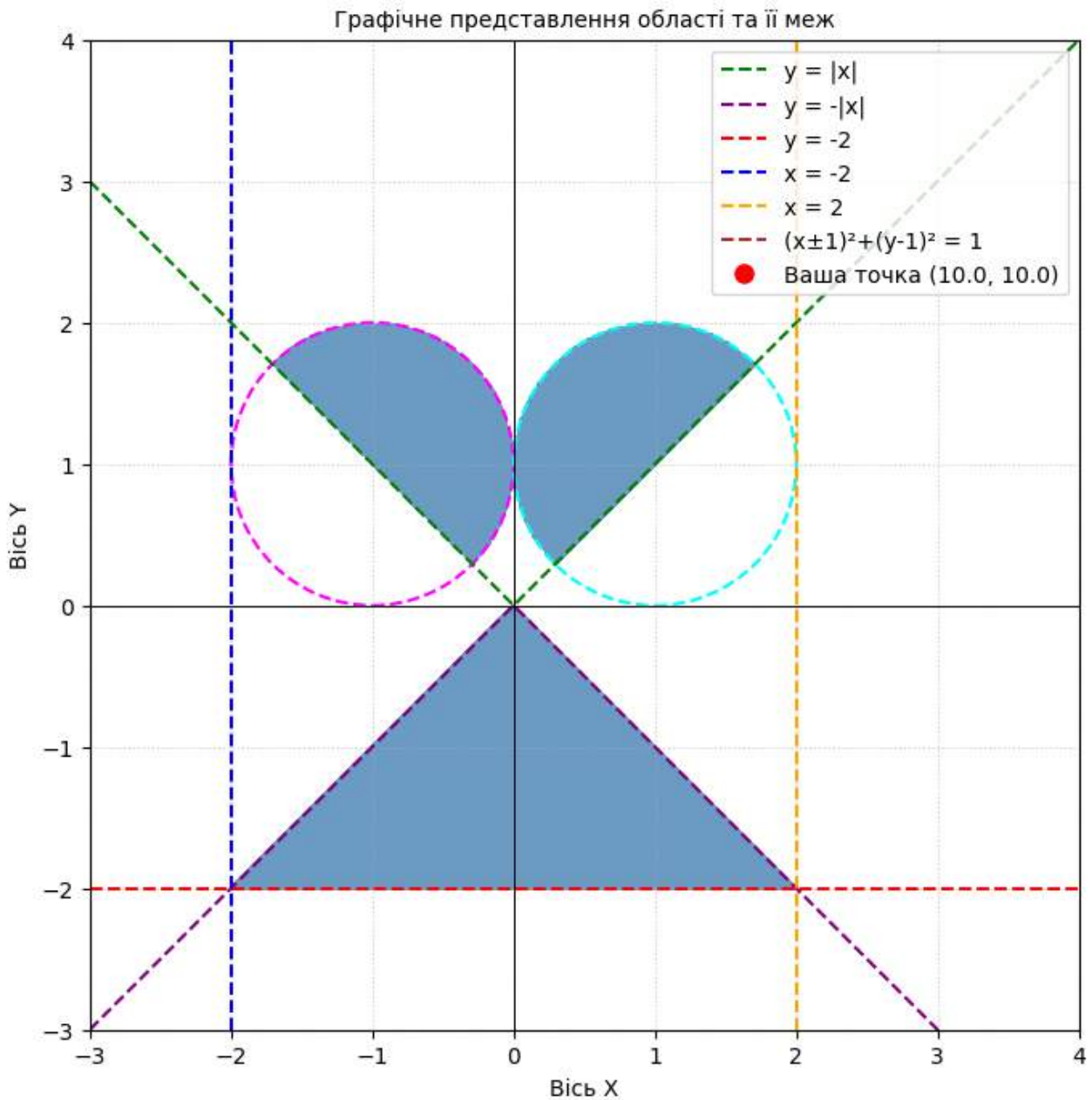


Рис. 5.30. Перевірка зовнішніх точок (далеко за межами фігур)

Графічна інтерпретація результатів розрахунку (рис. 5.30) демонструє поведінку алгоритму під час перевірки координат (10.0,10.0), що значно виходять за межі видимої координатної сітки. Отриманий результат «НЕ НАЛЕЖИТЬ» підтверджує стабільність роботи математичної моделі та відсутність помилок при обробленні великих числових значень.

– тест, що перевіряє коректність обробки нестрогих нерівностей ($\leq$, $\geq$), точки на межі мають вважатися такими, що належать області (рис. 5.31–5.33);

--- Перевірка належності точки заштрихованій області -
Введіть координату X: -1.0
Введіть координату Y: -1.0

Результат: Точка (-1.0, -1.0) НАЛЕЖИТЬ заштрихованій області.

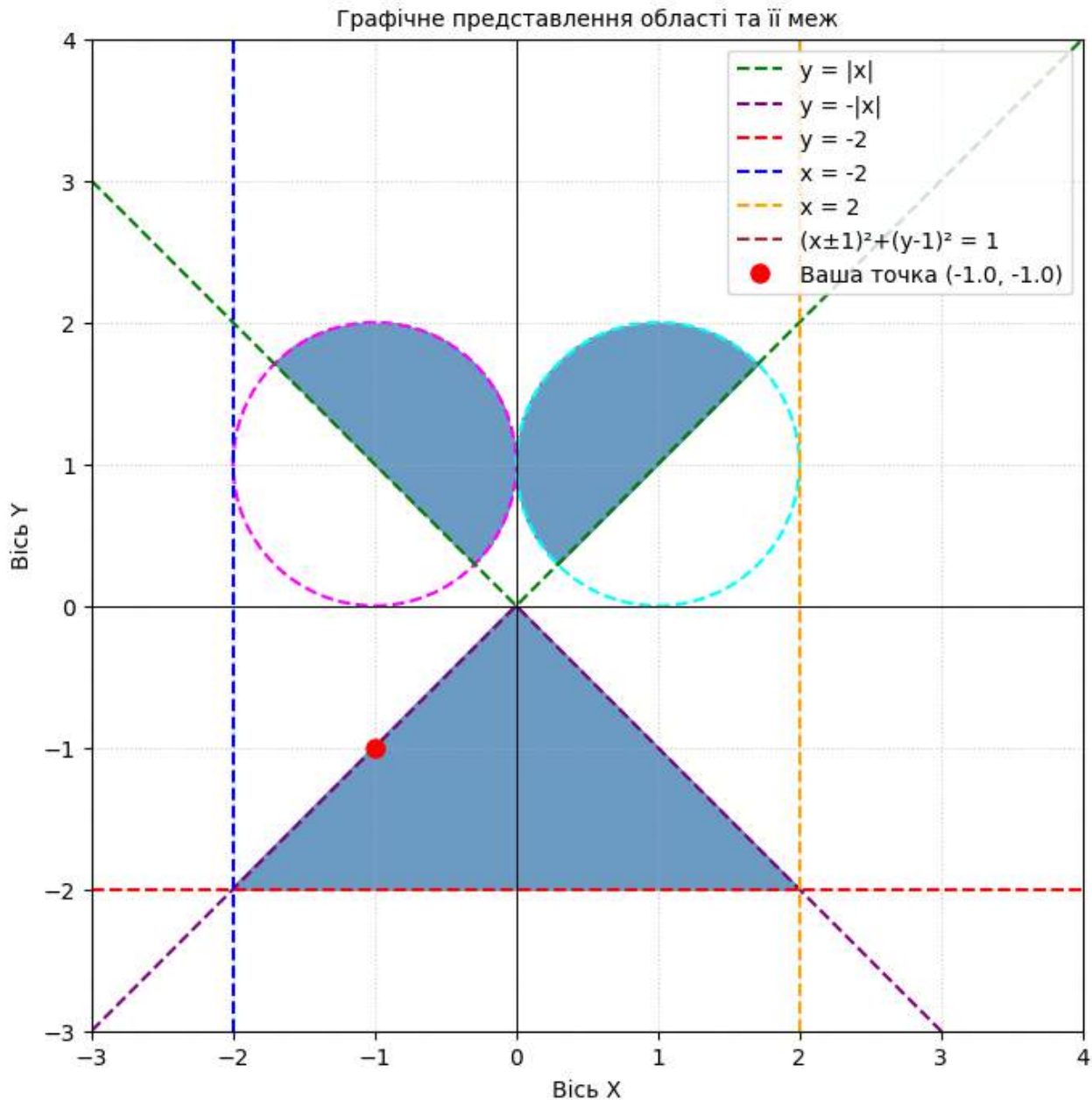


Рис. 5.31. Перевірка межових значень (на межі трикутника, що лежить на прямій $y = -|x|$)

Рис. 5.31 демонструє випадок знаходження точки безпосередньо на межі заштрихованої області, що підтверджує коректність обробки алгоритмом нестрогих нерівностей у математичній моделі.

--- Перевірка належності точки заштрихованій області -
Введіть координату X: 0.0
Введіть координату Y: 1.0

Результат: Точка (0.0, 1.0) НАЛЕЖИТЬ заштрихованій області.

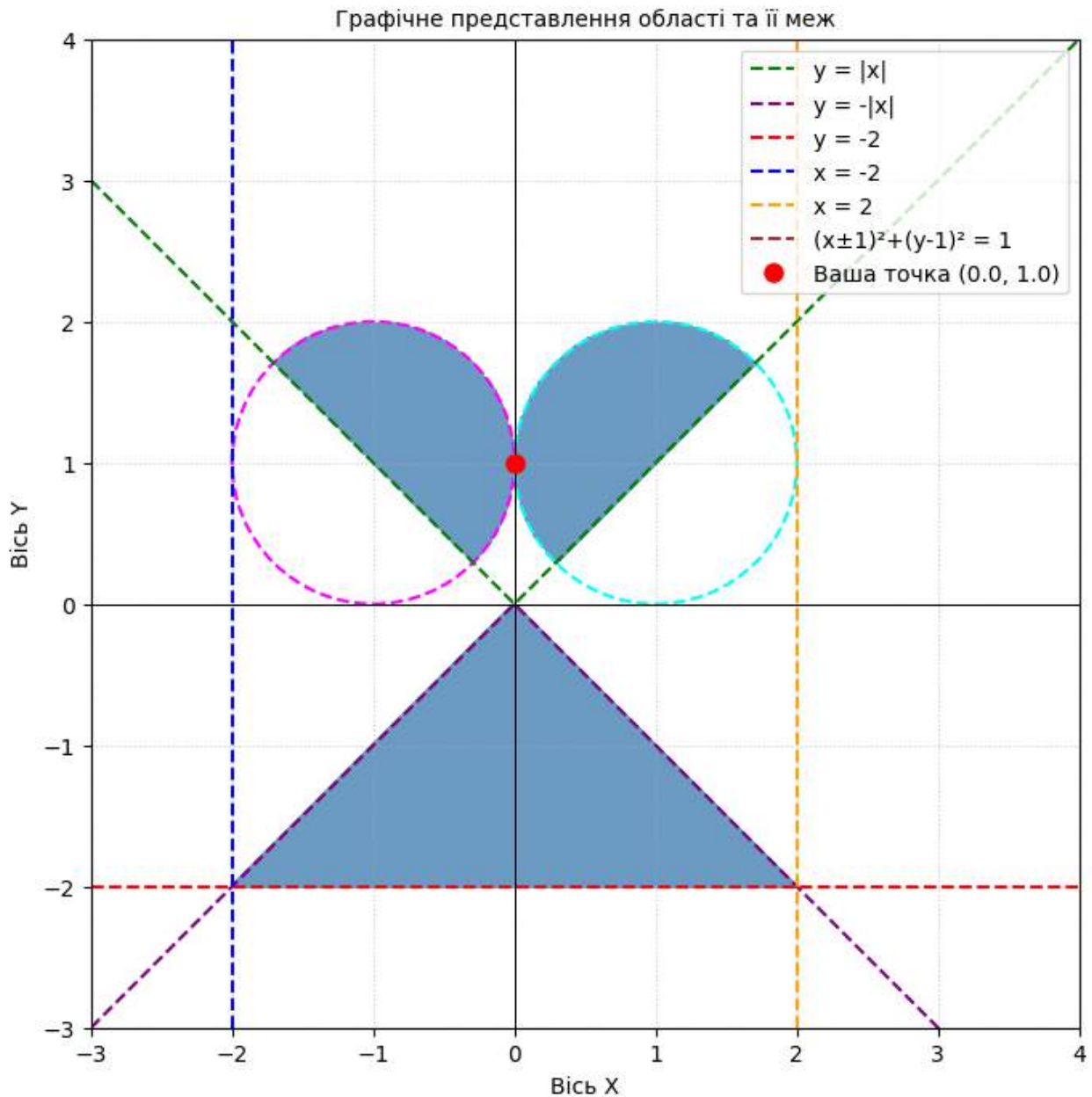


Рис. 5.32. Перевірка межових значень (дотик двох кіл)

Наведений рисунок ілюструє роботу програми під час перевірки точок, що потрапляють у критичні вузли перетину декількох функцій. Отриманий результат підтверджує, що логічні умови у програмному коді правильно розраховують пріоритетність меж (кіл та прямих), забезпечуючи високу точність класифікації об'єктів незалежно від складності геометричної конфігурації вказаного сектора.

--- Перевірка належності точки заштрихованій області -
Введіть координату X: 0.0
Введіть координату Y: -2.0

Результат: Точка (0.0, -2.0) НАЛЕЖИТЬ заштрихованій області.

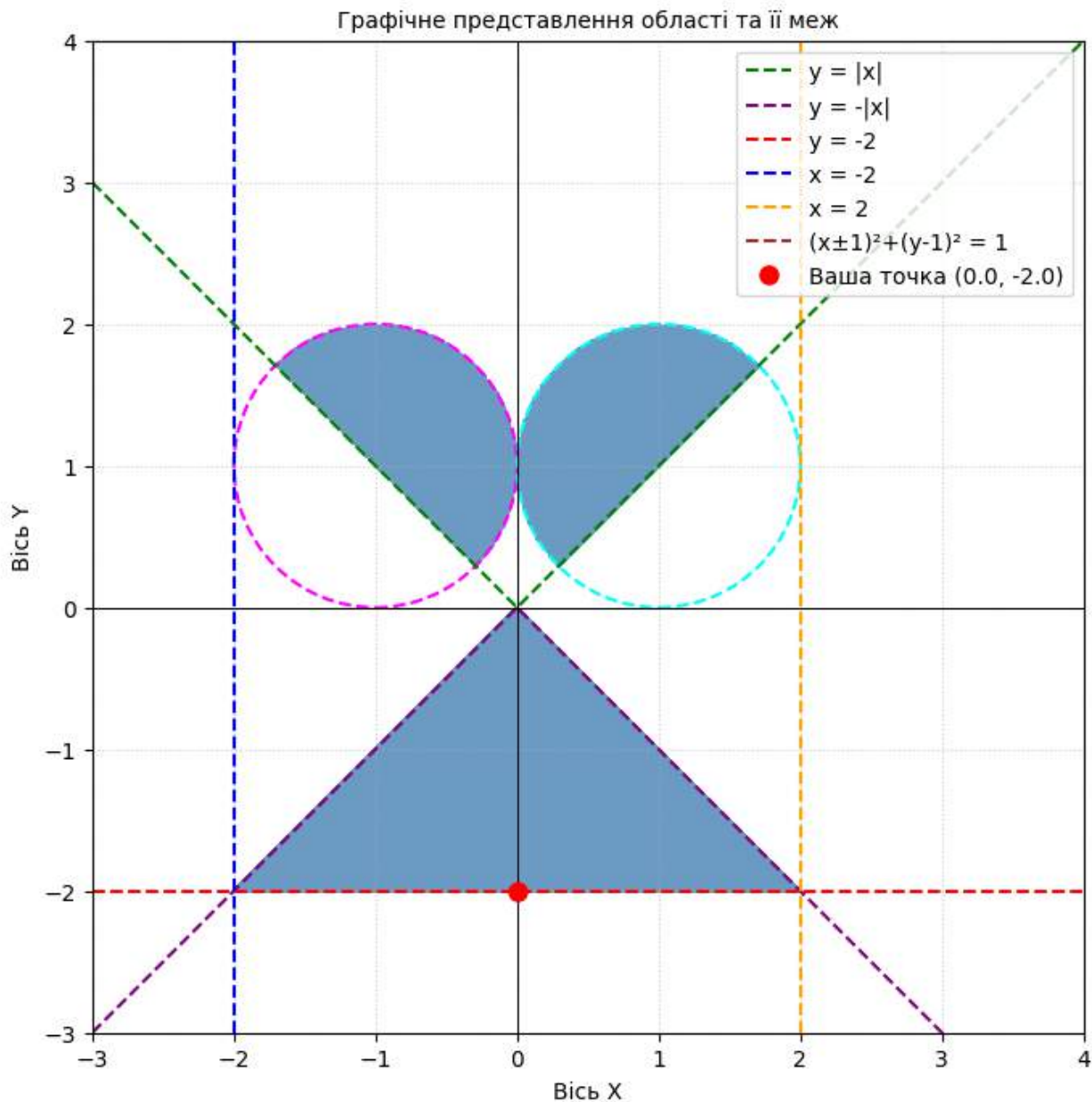


Рис. 5.32. Перевірка межових значень (на межі трикутника, що лежить на прямій $y = -2$)

Візуалізація результату на рис. 5.32 підтверджує коректну обробку межевої точки (0.0, -2.0), розташованої на перетині лінійних меж області. Результат «НАЛЕЖИТЬ» доводить точність розрахунків у критичних вузлах, де межа визначається прямою $y = -2$ та функціями модуля, що гарантує цілісність математичного опису фігури.

6. ОСНОВИ КОМБІНАТОРИКИ

Комбінаторика є розділом математики, що досліджує методи підрахунку кількості можливих варіантів вибору та розташування елементів. Основними об'єктами її вивчення є:

- упорядковані та неупорядковані вибірки;
- розміщення, перестановки та сполучення;
- комбінаторні структури, зокрема графи, дерева та розбиття множин.

Застосування комбінаторики охоплює широкий спектр дисциплін: теорію ймовірностей, інформатику, криптографію, теорію алгоритмів та задачі оптимізації. Вона забезпечує формальний апарат для аналізу складності процесів, оцінки ймовірностей подій та побудови ефективних методів розв'язання прикладних задач.

6.1. Основні поняття і визначення

Набір елементів $a_{i_1}, \dots, a_{i_r}$ з множини $U = \{a_1, \dots, a_n\}$ називається **вибіркою** обсягом r з n елементів, або (n, r) -**вибіркою**.

Вибірка називається *впорядкованою*, якщо порядок наступності елементів у ній заданий. Дві впорядковані вибірки, що різняться лише порядком наступності в них елементів, вважаються різними.

Якщо порядок наступності елементів у вибірці не вважається за істотний (не важливий), то вибірка називається *неупорядкованою*.

У вибірках можуть припускатися або не припускатися повторення елементів. Упорядкована (n, r) -вибірка, у якій елементи можуть повторюватися, називається *розміщенням з повтореннями* з n елементів по r , або (n, r) -*розміщенням з повтореннями*. Число можливих (n, r) -розміщень з повтореннями позначається $\tilde{A}_n^r$.

Якщо елементи впорядкованої (n, r) -вибірки попарно різні, то вона називається (n, r) -*розміщенням без повторень*, або просто (n, r) -*розміщенням*. Число всіх можливих (n, r) -розміщень позначається A_n^r .

Аналогічно для неупорядкованих вибірок. Неупорядкована (n, r) -вибірка, в якій елементи можуть повторюватися, називається *сполученням з повтореннями* з n елементів за r , або (n, r) -*сполученням з повтореннями*. Число можливих (n, r) -сполучень з повтореннями позначається $\tilde{C}_n^r$. Якщо елементи неупорядкованої (n, r) -вибірки попарно різні, то вона називається (n, r) -*сполученням без повторень*, або (n, r) -*сполученням*. Число можливих (n, r) -сполучень позначається C_n^r .

Приклад 6.1. Нехай задано множину $U = \{a, b, c\}$, $r = 2$. Тоді є:

- дев'ять розміщень з повтореннями: $aa, ab, ac, bb, ba, bc, cc, ca, cb$;
- шість розміщень без повторень: ab, ac, ba, bc, ca, cb ;
- шість сполучень з повтореннями: aa, ab, ac, bb, bc, cc ;
- три сполучення без повторень: ab, ac, bc .

Для підрахунку числа можливих комбінацій використовують такі правила.

Правило добутку. Якщо об'єкт A може бути вибраний m способами і після кожного такого вибору об'єкт B може бути вибраний n способами, то вибір об'єктів « A і B » може бути здійснений $m \cdot n$ способами.

Правило суми. Якщо об'єкт A може бути вибраний m способами, а об'єкт B може бути вибраний іншими n способами, то вибір об'єктів « A або B » може бути здійснений $m + n$ способами.

Приклад 6.2. З пункту A в пункт B відправляються 16 потягів, три літаки, вісім автобусів. Підрахуйте, скільки є способів дістатися з пункту A в пункт B ?

Розв'язання. Дістатися можна одним з альтернативних способів: автобусом, потягом або літаком. Виходить, слід використати правило суми:

$$N = N_a + N_n + N_l,$$

де N_a – кількість способів приїхати автобусом;

N_n – кількість способів приїхати потягом;

N_l – кількість способів прилетіти літаком.

Кількість можливих способів дістатися автобусом дорівнює кількості альтернатив-автобусів, тобто $N_a = 8$. Аналогічно, $N_n = 16$ і $N_l = 3$.

Підставляючи набуті значення, маємо:

$$N = N_a + N_n + N_l = 8 + 16 + 3 = 27$$

способів.

Приклад 6.3. У колективі 10 хлопців і п'ять дівчат. Для участі в конкурсі потрібно вибрати одного хлопця і одну дівчину. Скільки є способів вибрати представників?

Розв'язання. Хлопця можна вибрати десятьма способами. Після цього можна вибрати дівчину п'ятьма способами. За правилом добутку маємо:

$$N = 10 \cdot 5 = 50$$

способів.

6.2. Формули для підрахунку числа розміщень і сполучень

Для підрахунку кількості розміщень і сполучень використовують такі формули.

Число можливих (n, r) -розміщень визначають за формулою:

$$A_n^r = \frac{n!}{(n-r)!}; \quad (6.1)$$

у разі, коли $r = n$, маємо:

$$A_n^n = \frac{n!}{(n-n)!} = \frac{n!}{0!} = \frac{n!}{1} = n!.$$

Розміщення n елементів позначають P_n і називають *перестановкою* з n елементів. Таким чином, P_n обчислюють за формулою:

$$P_n = n!. \quad (6.2)$$

Число можливих (n, r) -розміщень з повтореннями отримують за формулою:

$$\tilde{A}_n^r = n^r. \quad (6.3)$$

Число можливих (n, r) -сполучень обчислюють за формулою:

$$C_n^r = \frac{n!}{r! \cdot (n-r)!}. \quad (6.4)$$

Число можливих (n, r) -сполучень з повтореннями визначають, використовуючи формулу:

$$\tilde{C}_n^r = C_{n+r-1}^{n-1}, \quad (6.5)$$

після чого знаходять потрібне число, застосовуючи формулу (6.4).

Підтвердимо результати, отримані у прикладі 6.1, використовуючи формули для підрахунку числа розміщень і сполучень (6.1)–(6.5).

Для множини з трьох елементів (тобто $n = 3$) при $r = 2$ маємо:

– кількість розміщень з повтореннями:

$$\tilde{A}_3^2 = 3^2 = 9;$$

– кількість розміщень без повторень:

$$A_3^2 = \frac{3!}{(3-2)!} = \frac{3!}{1!} = \frac{3 \cdot 2 \cdot 1}{1} = \frac{6}{1} = 6;$$

– кількість сполучень з повтореннями:

$$\tilde{C}_3^2 = C_{3+2-1}^{3-1} = C_4^2 = \frac{4!}{2! \cdot (4-2)!} = \frac{4!}{2! \cdot 2!} = \frac{4 \cdot 3 \cdot 2 \cdot 1}{2 \cdot 1 \cdot 2 \cdot 1} = \frac{24}{4} = 6;$$

– кількість сполучень без повторень:

$$C_3^2 = \frac{3!}{2! \cdot (3-2)!} = \frac{3!}{2! \cdot 1!} = \frac{3 \cdot 2 \cdot 1}{2 \cdot 1 \cdot 1} = \frac{6}{2} = 3.$$

Приклад 6.4. Щоб увійти в систему, необхідно знати пароль, що складається з семи символів. Для задання пароля можна використовувати 12 різних символів. Скільки існує варіантів задання невірного пароля?

Розв'язання.

Оскільки в паролі символи можуть повторюватися, а порядок їх слідування має значення, то для визначення кількості можливих варіантів задання пароля слід використовувати формулу для підрахунку числа розміщень з повтореннями:

$$\tilde{A}_{12}^7 = 12^7 = 35831808$$

варіантів.

Оскільки лише один варіант з цього числа є правильним, то число варіантів задання неправильного пароля дорівнює:

$$\tilde{A}_{12}^7 - 1 = 12^7 - 1 = 35831808 - 1 = 35831807$$

варіантам.

6.3. Біноміальні коефіцієнти

Розглянемо детальніше формулу (6.4) для визначення числа сполучень без повторень і спробуємо її спростити:

$$C_n^r = \frac{n!}{r! \cdot (n-r)!} = \frac{n \cdot (n-1) \cdot \dots \cdot (n-r+1) \cdot (n-r)!}{r! \cdot (n-r)!} = \frac{n \cdot (n-1) \cdot \dots \cdot (n-r+1)}{r!}.$$

Чисельник отриманої формули $n \cdot (n-1) \cdot \dots \cdot (n-r+1)$ позначають $(n)_r$, тобто:

$$C_n^r = \frac{n!}{r! \cdot (n-r)!} = \frac{(n)_r}{r!}. \quad (6.6)$$

Якщо n – натуральне число, то $(n)_n = n!$. Нагадаємо, що $0! = 1$.

Для будь-якого дійсного числа n і цілого невід'ємного r величина $\frac{(n)_r}{r!}$ називається **біноміальним коефіцієнтом**. Назва пояснюється застосуванням таких коефіцієнтів у формулі **бінома Ньютона**:

$$(x_1 + x_2)^n = \sum_{r=0}^n C_n^r \cdot x_1^r \cdot x_2^{n-r}, \quad (6.7)$$

де $x_1, \dots, x_n$ – будь-які числа. Розглянемо більш загальний випадок.

Нехай $r_1, r_2, \dots, r_k$ – цілі невід'ємні числа і $r_1 + r_2 + \dots + r_k = n$, тоді величина

$$\frac{n!}{r_1! \cdot r_2! \cdot \dots \cdot r_k!}$$

називається **поліноміальним коефіцієнтом**.

Справедлива рівність:

$$(x_1 + x_2 + \dots + x_k)^n = \sum \frac{n!}{r_1! \cdot r_2! \cdot \dots \cdot r_k!} \cdot x_1^{r_1} \cdot x_2^{r_2} \cdot \dots \cdot x_k^{r_k}. \quad (6.8)$$

Підсумовування проводиться за всіма цілочисельними невід'ємними розв'язками рівняння $r_1 + \dots + r_k = n$. Кількість доданків у сумі формули (6.8) дорівнює $\tilde{C}_k^n$. Розглянемо застосування формули (6.8) на прикладах.

Приклад 6.5. Обчислити вираз $(1 + x^4 + x^7)^2$, використовуючи формулу (6.7).

Розв'язання. Оскільки в дужках три доданки, то $k = 3$, вираз слід піднести до ступеня $n = 2$. Визначимо всі невід'ємні цілочисельні розв'язки рівняння $r_1 + r_2 + r_3 = 2$ і їхню кількість:

$$\tilde{C}_3^2 = C_4^2 = \frac{4!}{2! \cdot 2!} = \frac{24}{4} = 6$$

рішень.

Запишемо таблицю з можливими розв'язками (табл. 6.1).

Таблиця 6.1

Розподіли показників степеня між трьома доданками

№ розв'язку	r_1	r_2	r_3
1	0	1	1
2	0	0	2
3	0	2	0
4	2	0	0
5	1	0	1
6	1	1	0

Підставимо отриманий результат у формулу (6.8):

$$(1 + x^4 + x^7)^2 = \sum_{\substack{r_1=0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 1 \\ 1}} \sum_{\substack{r_2=1 \\ 0 \\ 2 \\ 0 \\ 0 \\ 0 \\ 1}} \sum_{\substack{r_3=1 \\ 2 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0}} \frac{2!}{r_1! \cdot r_2! \cdot r_3!} \cdot 1^{r_1} \cdot (x^4)^{r_2} \cdot (x^7)^{r_3} =$$

Обчислюємо кожний доданок:

$$(r_1 = 0, r_2 = 1, r_3 = 1):$$

$$\frac{2!}{0! \cdot 1! \cdot 1!} \cdot 1^0 \cdot (x^4)^1 \cdot (x^7)^1 = \frac{2!}{0! \cdot 1! \cdot 1!} \cdot x^4 \cdot x^7 = 2 \cdot x^{11};$$

$$(r_1 = 0, r_2 = 0, r_3 = 2):$$

$$\frac{2!}{0! \cdot 0! \cdot 2!} \cdot 1^0 \cdot (x^4)^0 \cdot (x^7)^2 = \frac{2!}{0! \cdot 0! \cdot 2!} \cdot x^{14} = 1 \cdot x^{14};$$

$$(r_1 = 0, r_2 = 2, r_3 = 0):$$

$$\frac{2!}{0! \cdot 2! \cdot 0!} \cdot 1^0 \cdot (x^4)^2 \cdot (x^7)^0 = \frac{2!}{0! \cdot 2! \cdot 0!} \cdot x^8 = 1 \cdot x^8;$$

$$(r_1 = 2, r_2 = 0, r_3 = 0):$$

$$\frac{2!}{2! \cdot 0! \cdot 0!} \cdot 1^2 \cdot (x^4)^0 \cdot (x^7)^0 = \frac{2!}{2! \cdot 0! \cdot 0!} \cdot 1 = 1;$$

$$(r_1 = 1, r_2 = 0, r_3 = 1):$$

$$\frac{2!}{1! \cdot 0! \cdot 1!} \cdot 1^1 \cdot (x^4)^0 \cdot (x^7)^1 = \frac{2!}{1! \cdot 0! \cdot 1!} \cdot x^7 = 2 \cdot x^7;$$

$$(r_1 = 1, r_2 = 1, r_3 = 0):$$

$$\frac{2!}{1! \cdot 1! \cdot 0!} \cdot 1^1 \cdot (x^4)^1 \cdot (x^7)^0 = \frac{2!}{1! \cdot 1! \cdot 0!} \cdot x^4 = 2 \cdot x^4.$$

Підсумковий результат:

$$(1 + x^4 + x^7)^2 = 1 + 2x^4 + 2x^7 + x^8 + 2x^{11} + x^{14}.$$

Приклад 6.6. Знайти коефіцієнт при x^{15} у виразі $(2 + x^3 + x^4)^5$.

Розв'язання. Тут $k = 3$, $n = 5$. Виділимо ненульові ступені x у сумі: це x^3 і x^4 . Завдання зводиться до знаходження розв'язків рівняння:

$$(x^3)^{r_2} \cdot (x^4)^{r_3} = x^{15},$$

тобто

$$3 \cdot r_2 + 4 \cdot r_3 = 15.$$

Розв'язками цього рівняння є:

$$r_2 = 1, r_3 = 3 \Rightarrow r_1 = 5 - 3 - 1 = 1 \text{ і } r_2 = 5, r_3 = 0 \Rightarrow r_1 = 5 - 5 - 0 = 0.$$

Використовуючи перший розв'язок для підстановки у формулу (6.8), отримуємо результат:

$$C_{2^1 \cdot (x^3)^1 \cdot (x^4)^3}^1 = \frac{5!}{1! \cdot 1! \cdot 3!} \cdot 2^1 \cdot (x^3)^1 \cdot (x^4)^3 = \frac{5 \cdot 4}{1} \cdot 2 \cdot x^{15} = 40 \cdot x^{15}.$$

Підставивши у формулу (6.8) другий розв'язок, отримуємо:

$$C_{2^0 \cdot (x^3)^5 \cdot (x^4)^0}^2 = \frac{5!}{0! \cdot 5! \cdot 0!} \cdot 2^0 \cdot (x^3)^5 \cdot (x^4)^0 = \frac{1}{1} \cdot 1 \cdot x^{15} = x^{15}.$$

Склавши коефіцієнти при першому і другому входженні x^{15} , отримаємо підсумкове значення коефіцієнта при x^{15} у виразі $(2 + x^3 + x^4)^5$: **41**.

6.4. Формула включення-виключення

Нехай дано N об'єктів і їхні властивості $a_1, a_2, \dots, a_n$. Форма запису $N_{a_i a_j \bar{a}_k}$ означає, що об'єкт має властивості a_i і a_j , не має властивості a_k , решта властивостей може бути як присутньою, так і відсутньою. Тоді:

$$N_{\bar{a}_1 \bar{a}_2 \dots \bar{a}_n} = N - \sum_i N_{a_i} + \sum_{i < j} N_{a_i a_j} - \sum_{i < j < k} N_{a_i a_j a_k} + \dots + (-1)^n \cdot N_{a_1 a_2 \dots a_n}. \quad (6.9)$$

Пояснимо формулу на прикладі.

Приклад 6.7. У групі навчаються 20 студентів.

Із них:

десять – уміють розв'язувати завдання на обчислення коефіцієнтів (k);

вісім – уміють розв'язувати завдання на розміщення (p);

сім – уміють розв'язувати завдання на сполучення (c);

чотири – уміють розв'язувати завдання на обчислення коефіцієнтів і розміщення (k, p);

три – уміють розв'язувати завдання на обчислення коефіцієнтів і сполучення (k, c);

два – уміють розв'язувати завдання на розміщення і сполучення (p, c);

один – може розв'язувати всі перераховані види комбінаторних завдань. Скільки студентів взагалі не вміють розв'язувати зазначені завдання?

Розв'язання.

$$N = 20; \quad N(k) = 10; \quad N(p) = 8; \quad N(c) = 7; \quad N(k, p) = 4; \quad N(k, c) = 3; \\ N(p, c) = 2; \quad N(k, p, c) = 1.$$

Потрібно знайти $N(\bar{k}, \bar{p}, \bar{c})$.

Підставивши перераховані значення у формулу (6.9), отримуємо:

$$N_{\bar{k} \bar{p} \bar{c}} = N - N_k - N_p - N_c + N_{k, p} + N_{k, c} + N_{p, c} - N_{k, p, c} = \\ = 20 - 10 - 8 - 7 + 4 + 3 + 2 - 1 = 3 \text{ студента.}$$

6.5. Вправи до розділу

1. Із Києва до Чернігова можна дістатися потягом, пароплавом, автобусом, літаком; з Чернігова до Новгород-Сіверського – пароплавом або автобусом. Скільки є способів здійснити подорож за маршрутом Київ – Чернігів – Новгород-Сіверський?
2. Визначити, скільки є способів вибрати в слові «комбінаторика»:
 - а) голосну і приголосну;
 - б) дві приголосні;
 - в) три голосні;
 - г) дві голосні та дві приголосні?
3. Обчислити, скільки є способів у грі «Спорт-лото»:
 - а) вибрати п'ять куль з 36;
 - б) вгадати чотири номери (усього 36 куль);
 - в) вгадати три номери (усього 36 куль);
 - г) вибрати шість куль з 45;
 - д) вгадати п'ять номерів (усього 45 куль);
 - е) вгадати чотири номери (усього 45 куль);
 - ж) вгадати три номери (усього 45 куль)?
4. Скільки різних телефонних номерів можна сформувати, якщо номер має бути:
 - а) п'ятизначним;
 - б) шестизначним;
 - в) семизначним?
5. У групі навчаються 20 студентів. Скільки існує способів сформувати команду для участі в олімпіаді, якщо команда повинна складатися:
 - а) з трьох студентів;
 - б) з чотирьох студентів;
 - в) з п'яти студентів?
6. У виразі $(1 + x^3 + x^5)^8$ визначити:
 - а) кількість доданків поліноміальної формули;
 - б) усі цілочисельні невід'ємні розв'язки рівняння $r_1 + \dots + r_k = n$;
 - в) коефіцієнт при x^{13} ;
 - г) коефіцієнт при x^{19} ;
 - д) коефіцієнт при x^{15} .
7. У виразі $(3 + x^2 + x^7)^5$ визначити:
 - а) кількість доданків поліноміальної формули;
 - б) усі цілочисельні невід'ємні розв'язки рівняння $r_1 + \dots + r_k = n$;
 - в) коефіцієнт при x^9 ;

- г) коефіцієнт при x^{20} ;
- д) коефіцієнт при x^{14} .

8. У виразі $(x^3 + x^4 + x^7)^{11}$ визначити:

- а) кількість доданків поліноміальної формули;
- б) усі цілочисельні невід'ємні розв'язки рівняння $r_1 + \dots + r_k = n$;
- в) коефіцієнт при x^{14} ;
- г) коефіцієнт при x^{17} ;
- д) коефіцієнт при x^8 .

9. У відділі працює 15 програмістів. Із них п'ять програмістів використовують у своїй роботі C#, шість програмістів – Oracle, сім програмістів – My SQL, троє – C# і My SQL, двоє – Oracle і C#, ще двоє – Oracle і My SQL. Один програміст використовує всі вказані засоби. Скільки чоловік не використовують жодної з цих систем?

10. Текстовий редактор, не знайшовши у своєму словнику введеного слова, подає можливі варіанти написання. Скільки існує варіантів неправильного написання слова, якщо:

- а) мова – українська (33 літери), слово з шести букв, допустима помилка – одна буква;
- б) мова – українська (33 літери), слово з восьми букв, допустима помилка – дві букви;
- в) мова – англійська (26 літер), слово з п'яти букв, допустима помилка – одна буква;
- г) мова – англійська (26 літер), слово з семи букв, допустима помилка – дві букви.

11. Для задання пароля використовується алфавіт, що складається з 26 символів. Скільки існує способів введення неправильного пароля, якщо:

- а) пароль складається з семи символів;
- б) пароль складається не більше ніж з восьми символів;
- в) пароль складається з шести символів;
- г) пароль складається з п'яти символів;
- д) пароль складається з чотирьох символів.

12. Порахувати:

а) У чоловіка є п'ять піджаків, вісім сорочок і сім краваток. Скільки різних костюмів можна скласти з цих предметів?

б) У жінки в шафі висить шість суконь, п'ять спідниць і три блузи. Скільки різних нарядів вона може скласти зі свого одягу?

в) У холодильнику є морозиво шести різних найменувань. На десерт можна узяти одну, дві або навіть три порції морозива одразу. Скільки можливостей є у вас для різних десертів?

13. Порахувати:

а) Перевертень – це багатозначне число, яке не змінить свого значення, якщо всі його цифри записати в оберненому порядку. Скільки існує шестизначних перевертнів? А скільки семизначних?

б) Скільки чотиризначних чисел, що не перевершують 6000, можна скласти, використовуючи тільки непарні цифри?

в) Пароль, що відкриває доступ до комп'ютера, складається з шести символів. Перші два з них – малі букви латинського алфавіту (всього 26 букв), а чотири символи, що залишилися, можуть бути як цифрами, так і малими буквами. Скільки можна придумати різних паролів?

14. Нехай S – множина чотиризначних чисел, у десятковому записі котрих беруть участь цифри 0, 1, 2, 3 і 6, причому 0 на першому місці стояти не може.

а) Яка потужність множини S ?

б) Скільки чисел з S у своєму десятковому записі не мають цифр, що повторюються?

в) Як багато парних чисел серед чисел пункту б)?

г) Скільки чисел з пункту б) виявляться більшими ніж 4000?

15. Обчисліть такі величини:

д) $P(7, 2)$, $P(8, 5)$, $P(6, 4)$ і $P(n, n - 1)$;

е) $C(10, 7)$, $C(9, 2)$, $C(8, 6)$ і $C(n, n - 1)$.

Переконайтеся, що $C(n, k) = C(n, n - k)$.

16. Порахувати:

а) Скільки існує можливостей для присудження першого, другого і третього місць сімнадцяти учасникам змагань з ікебани?

б) Комітет з 20 членів обирає голову і секретаря. Скількома способами це можна зробити?

в) Пароль, що відкриває доступ до комп'ютера, складається за правилами вправи 6.2 (в) навчального посібника. Скільки різних паролів можна написати зі символів, що не повторюються?

17. Порахувати:

а) Хокейна команда налічує 18 гравців. Одинадцять з них входять до основного складу. Підрахуйте кількість можливих основних складів.

б) Судейська бригада з 5 жінок і 7 чоловіків повинна бути створена із кандидатів, серед яких вісім жінок і 11 чоловіків. Скільки можна вибрати різних судейських бригад?

в) Належить вибрати команду з чотирьох гравців у гольф з п'яти професійних гравців і п'яти любителів. Скільки різних команд може складатися з трьох професіоналів і одного любителя? Скільки команд складається тільки з професіоналів або тільки з любителів?

18. В один із комітетів парламенту потрібно відібрати трьох членів, причому вибирати треба з п'яти консерваторів, трьох лейбористів і чотирьох ліберал-демократів. Визначте:

- а) скільки різних комітетів можна скласти?
- б) скільки різних комітетів можна скласти, якщо до комітету повинен входити принаймні один ліберал-демократ?
- в) скільки різних комітетів можна скласти, якщо лейбористи і консерватори не можуть бути його членами одночасно?
- г) скільки різних комітетів можна скласти, якщо до комітету повинен увійти принаймні один консерватор і хоч би один лейборист?

19. У невеликій фірмі вісім чоловік працюють на виробництві, п'ятеро – у відділі збуту і троє – в бухгалтерії. Для обговорення нової продукції було вирішено запросити на нараду шістьох працівників. Скількома способами це можна зробити, якщо:

- а) необхідно запросити по два представники від кожного відділу;
- б) необхідно запросити принаймні двох представників виробництва;
- в) необхідні представники кожного з трьох відділів?

20. Порахувати:

- а) Ресторан у своєму меню пропонує п'ять різних головних блюд. Кожний з компанії в шість чоловік замовляє своє головне блюдо. Скільки різних замовлень може одержати офіціант?
- б) Квіткарка продає троянди чотирьох різних сортів. Скільки різних букетів можна скласти з дюжини троянд?

21. Ви купуєте п'ять різдвяних листівок у магазині, який може запропонувати чотири різних типи листівок, що сподобалися вам.

- а) Як багато наборів з п'яти листівок можна купити?
- б) Скільки наборів можна скласти, якщо обмежитися тільки двома типами листівок з чотирьох, але купити все одно п'ять листівок?

22. Ось восьмий рядок трикутника Паскаля:

1 7 21 35 35 21 7 1

- а) 3
- б) знайти дев'ятий і десятий його рядки.
- в) Перевірити, що якщо a , b і c – три послідовні числа у восьмому рядку трикутника Паскаля, то одне з чисел десятого рядка можна одержати як суму $a + 2b + c$.
- г) Довести рівність, користуючись формулою Паскаля:

$$C(n, k) + 2C(n, k + 1) + C(n, k + 2) = C(n + 2, k + 2)$$

при $0 < k < n - 2$.

23. Порахувати:

- а) Скільки різних «слів» можна одержати із слова «АБРАКАДАБРА»?
- б) Скільки з них починаються з літери «К»?
- в) У скількох з них обидві літери «Б» стоять поряд?

24. Поклавши в біномі Н'ютона $a = b = 1$,

- а) показати, що для будь-якого $n = 0, 1, 2, \dots$ справедлива формула

$$C(n, 0) + C(n, 1) + \dots + C(n, n) = 2^n.$$

Довести, що в множині S з n елементів міститься рівно $2n$ різних підмножин (вказівка: визначити спочатку, скільки підмножин потужності k міститься в S).

- б) показати, що

$$C(n, 0) - C(n, 1) + C(n, 2) - \dots + (-1)^n C(n, n) = 0.$$

25. Порахувати:

- а) Знайти коефіцієнт при $a^3 b^5$ після розкриття дужок у виразі $(a + b)^8$.
- б) Знайти коефіцієнт при $xy^3 z^4$ після розкриття дужок у виразі $(x + y + z)^8$.
- в) Знайти коефіцієнт при $xy^2 z$ після розкриття дужок у виразі $(x + 2y + z - 1)^5$.

26. Скількома способами можна розкласти 17 однакових куль по шестити різних пакетах, якщо жодний пакет не має бути порожнім?

27. Скільки є способів розміщення k пасажирів у n вагонах поїзда, за яких рівно m вагонів виявиться порожніми?

28. Скількома способами можна розкласти:

- а) 19 однакових куль по семи урнах так, щоб виявилось не більше двох порожніх урн;
- б) 20 однакових куль по п'яти урнах так, щоб у кожній урні виявилось не менше двох куль?

29. Довести, що кількість розв'язків рівняння $x_1 + x_2 + \dots + x_m = n$ у невід'ємних цілих числах дорівнює кількості розв'язків рівняння

$$x_1 + x_2 + \dots + x_m = n + m$$

у натуральних числах.

30. Скількома способами п'ять червоних, сім білих і дев'ять синіх куль можна розкласти в шість різних пакетів (деякі пакети можуть бути порожніми)?

31. Скількома способами можна розмістити n білих, m червоних і k синіх куль по p урнах? Скільки є таких розміщень, де в першій урні міститься n білих, m червоних і k синіх куль?

32. У $2n$ урнах розподілено n білих і n чорних однакових куль, причому в кожній урні є принаймні одна куля. Скількома різними способами можна це зробити?

33. Скількома способами можна укласти букет з 11 квіток, якщо квітковий магазин пропонує 25 різних сортів квітів? Скільки існує таких способів, щоб у кожному букеті обов'язково було щонайменше три троянди?

34. Скількома способами можна вибрати k тістечок у кондитерській, де є n різних сортів тістечок, так, щоб серед обраних k тістечок обов'язково були тістечка кожного із n сортів ($k \geq n$)?

35. Скількома способами можна вибрати п'ять літер із набору?

а) а, а, а, а, а, б, б, б, б, б;

б) а, а, а, а, а, б, б, б, б, б, в, в, в, в, в;

в) а, а, а, а, а, б, б, в, в, в.

36. Скількома способами можна розподілити k екзаменаційних білетів між n студентами?

37. Скількома способами з шістьох інженерів і 14 робітників можна створити бригаду, яка складалася б із двох інженерів і п'яти робітників?

38. Визначити, скількома способами можна вибрати з натуральних чисел від 1 до 100 три натуральні числа так, щоб їх сума:

а) була непарною;

б) ділилася на три.

39. Скількома способами можна вибрати з n чоловік групу людей для роботи, якщо група має складатися не менше ніж із p чоловік?

40. Скільки є перестановок цифр 0, 1, 2, ..., 9, у яких цифра 2 займає друге місце, а цифра 5 – п'яте або шосте?

41. Нехай дано n символів a , m символів b . Визначити кількість різних слів,

а) які можна скласти зі всіх цих символів;

б) які складаються зі всіх цих символів і у яких жодні два символи b не стоять поруч.

42. Скільки різних слів можна утворити, переставляючи літери в слові?

а) комбінаторика;

б) філологія;

в) мінімум;

г) абракадабра?

43. Скількома способами можна усадити за круглий стіл n чоловіків і m жінок так, щоб жодні дві жінки не сиділи поруч?

44. Скількома способами можна роздати 28 кісток доміно чотирьом гравцям так, щоб кожний одержав сім кісток?

45. Скількома способами можна переставити літери слова «кавоварка» так, щоб голосні та приголосні чергувалися? Розв'язати ту саму задачу для слова «палеонтолог».

46. Визначити, скількома способами можна переставити літери наведеного слова так, щоб приголосні йшли за абеткою:

- а) абракадабра;
- б) монолог;
- в) амальгама.

47. Скільки слів довжиною 5 можна утворити з літер a, b, c , якщо:

- а) кожна із літер зустрічається у слові без обмежень;
- б) літеру a можна використати у слові лише один раз;
- в) літеру a можна використати у слові не більше двох разів?

48. У скількох перестановках літер $a, a, a, a, b, b, b, c, c$ не зустрічається жодне з підслів $aaaa, bbb, cc$?

49. У скількох перестановках літер a, a, b, b, c, c жодна літера не збігається з літерою, що стоїть на такому самому місці у слові $aabbcc$?

50. Нехай M – скінченна множина і $A, B \subseteq M$. Розташуйте в порядку неспадання такі величини:

- а) $|B|, |A \cup B|, |\emptyset|, |A \cap B|, |M|$;
- б) $|A \setminus B|, |A| + |B|, |A \Delta B|, |\emptyset|, |A \cup B|$.

51. Довести нерівності:

- а) $|A \cup B| \leq |A| + |B|$; д) $|A \Delta B| \leq |A \cup B|$;
- б) $|A \setminus B| \leq |A|$; е) $|A| < |B(A)|$;
- в) $|A \Delta B| \leq |A| + |B|$; ж) $|A \cap B| \leq |A|$;
- г) $|A \setminus B| \geq |A| - |B|$; и) $|A \cap B| \leq |A \cup B|$.

52. Нехай A – скінченна множина, a – елемент, а B – підмножина множини A . Яких підмножин множини A більше:

- а) тих, що містять елемент a , чи тих, що не містять елемента a ;
- б) тих, що містять множину B , чи тих, що не перетинаються із множиною B ;
- в) тих, що містять множину B , чи тих, що не містять множину B ?

53. У групі 27 студентів. Із них 16 відвідують семінар А, 12 – семінар Б, а 7 студентів не відвідують жодного семінару.

- а) Скільки студентів відвідують семінари А та Б?

б) Скільки студентів відвідують лише семінар А?

54. Знайти кількість і суму чотиризначних натуральних чисел, що не діляться на жодне з таких чисел:

а) 2, 5, 11;

б) 6, 10, 18.

55. Комбінаторика мережевих адрес (IPv4). Адреса протоколу IPv4 складається з 32 біт.

а) Скільки всього унікальних адрес можна сформувати в цьому просторі?

б) У мережах класу С перші 24 біти відведено під адресу мережі, а останні 8 біт – під адреси пристроїв (хостів). Скільки пристроїв можна підключити до такої мережі, якщо адреси, що складаються з усіх нулів та усіх одиниць в останньому октеті, зарезервовані для службових цілей?

56. Оптимізація бази даних

а) Програміст створює SQL-запит для вибору даних із таблиці Users. У блоці WHERE він планує використати чотири різні умови (наприклад, по місту, віку, імені та даті реєстрації), поєднані логічним оператором AND.

б) Скільки існує способів записати послідовність цих умов у запиті?

в) Якщо система керування базами даних (СКБД) підтримує 3 типи індексів для цієї таблиці, скількома способами можна призначити по одному індексу для кожної з чотирьох умов?

57. Шляхи в дискретній сітці (Grid Walking). Уявіть двовимірну сітку розміром $M \times N$. Робот знаходиться в точці $(0,0)$ і повинен дістатися точки (M,N) , рухаючись лише на один крок праворуч або на один крок угору за раз.

а) Обчисліть загальну кількість найкоротших шляхів для сітки 8×8 .

б) Скільки таких шляхів проходитиме через «вузол» із координатами $(3,4)$?

58. Формування складних паролів. Пароль до інформаційної системи повинен містити рівно 8 символів. Дозволено використовувати цифри (0–9) та малі літери латинського алфавіту (26 символів).

а) Скільки всього паролів можна створити, якщо символи можуть повторюватися?

б) Скільки існує варіантів, якщо пароль обов'язково має містити принаймні одну цифру?

в) Скільки варіантів пароля можна скласти, якщо цифри не можуть стояти на першому та останньому місцях?

59. Розміщення ресурсів у кластері

а) Системному адміністратору потрібно розподілити 10 ідентичних мікросервісів (контейнерів) по 4 різних серверах.

б) Скількома способами це можна зробити, якщо деякі сервери можуть залишитися порожніми?

в) Скільки існує способів, якщо на кожному сервері має бути запущено хоча б один мікросервіс?

60. Комбінаторика кодових послідовностей. Код повідомлення складається з послідовності бітів довжиною 12.

а) Скільки існує кодів, що містять рівно п'ять одиниць і сім нулів?

б) У скількох із цих кодів усі п'ять одиниць стоять поруч одна за одною?

61. Дослідження читацьких смаків студентів показало, що 60 % студентів читають журнал А, 50 % – журнал Б, 50 % – журнал В, 30 % – журнали А та Б, 20 % – журнали Б і В, 40 % – журнали А та В, 10 % – журнали А, Б і В. Скільки відсотків студентів:

а) читають принаймні один журнал;

б) не читають жодного з журналів;

в) читають точно два журнали;

г) читають не менше двох журналів?

62. Під час екзаменаційної сесії з чотирьох іспитів не менше 70 % студентів склали іспит з дискретної математики, не менше 75 % – з математичного аналізу, не менше 80 % – з алгебри й не менше 85 % – з програмування. Яка мінімальна кількість студентів, що склали одночасно всі чотири іспити?

63. Номер автомашини складається із трьох літер українського алфавіту (що містить 33 літери) і чотирьох цифр. Скільки можна скласти різних номерів автомашин?

64. Скількома способами можна розмістити на шахівниці розміром $m \times n$ дві різнокольорові тури так, щоб вони не били одна одну?

65. Скільки існує n -значних десяткових чисел,

а) які починаються із двох однакових цифр;

б) у яких сусідні цифри різні;

в) усі цифри яких непарні;

г) у запису яких є принаймні одна непарна цифра;

д) у запису яких немає цифри 9;

е) у запису яких обов'язково є цифра 3?

66. Визначити, скільки різних натуральних дільників має число:

а) $32015 \cdot 52016 \cdot 112017$;

б) 24482124;

в) 31244004;

г) 92162331.

67. Скількома способами у множині A з n елементів можна вибрати дві підмножини, що не перетинаються?

68. Нехай множина A містить n елементів, а множина B – m елементів. Визначити кількість:

- а) сюр'єктивних відповідностей;
- б) ін'єктивних відповідностей;
- в) бієктивних відповідностей між множинами A та B .

69. На одній прямій дано n точок, а на іншій, паралельній першій, – m точок. Скільки існує трикутників, вершинами яких є ці точки?

70. Нехай множина M містить n елементів. Визначити кількість:

- а) неантирефлексивних відношень;
- б) несиметричних відношень;
- в) неантисиметричних відношень;
- г) рефлексивних і симетричних відношень;
- д) рефлексивних і антисиметричних відношень;
- е) антирефлексивних і симетричних відношень;
- ж) антирефлексивних і несиметричних відношень;
- з) антирефлексивних і антисиметричних відношень на множині M .

71. Формування логічних імен (ідентифікаторів). В одній із навчальних мов програмування ідентифікатор змінної може складатися лише з двох символів. Перший символ обов'язково має бути літерою латинського алфавіту (26 варіантів), а другий – або літерою, або цифрою (10 варіантів).

- а) Скільки всього існує таких ідентифікаторів?
- б) Скільки з них закінчуються цифрою?

72. Налаштування палітри кольорів (RGB). Для оформлення інтерфейсу програми дизайнер використовує спрощену палітру, де кожен колір (червоний, зелений, синій) може мати один із 4 рівнів інтенсивності.

- а) Скільки всього різних кольорів можна створити таким чином?
- б) Скільки існує кольорів, де інтенсивність червоного каналу максимальна?

73. Розподіл завдань у команді (Agile). У студентській команді з 5 осіб потрібно обрати одного «тімліда» (керівника) та одного «тестувальника».

- а) Скількома способами це можна зробити, якщо одна людина не може обіймати дві посади одночасно?
- б) Скількома способами можна обрати двох просто «розробників» з цієї команди для виконання конкретного завдання (порядок вибору не має значення)?

6.6. Використання основ мови Python у комбінаториці

У цьому розділі розглядаються підходи до програмної реалізації базових комбінаторних обчислень. З метою формування глибокого розуміння обчислювальних процесів матеріал подається через порівняльний аналіз двох методів:

1) використання стандартних бібліотек (застосування оптимізованих функцій модуля *math*, що є стандартом у професійній розробці);

2) алгоритмічна реалізація (побудова обчислень «з нуля» за допомогою ітеративних конструкцій (циклів), що дозволяє дослідити внутрішню логіку математичних формул).

Особливості ітераційних процесів (функція *range*)

У реалізації математичних алгоритмів необхідно враховувати специфіку генерації числових послідовностей у Python.

У математиці інтервал «від 1 до n » охоплює обидві межі. У Python функція *range(start, stop)*, яка використовується для створення циклів, генерує значення, починаючи зі *start* і завершуючи значенням, що передує *stop*, тобто верхня межа не входить у набір.

Для коректної реалізації математичного циклу до n включно програмний діапазон необхідно задавати зі зміщенням на одиницю: *range(1, $n + 1$)*.

Перестановки (факторіал)

Кількість перестановок $P(n)$ для множини з n елементів визначається як факторіал числа n :

$$P_n = n! = 1 \cdot 2 \cdot \dots \cdot n.$$

Стандартним інструментарієм є модуль *math*, що містить функцію *math.factorial(n)*, яка виконує обчислення максимально швидко.

Алгоритмічний підхід полягає в тому, що задача зводиться до накопичення добутку. Використовується змінна-акумулятор, яка ініціалізується одиницею та в циклі послідовно множиться на кожне число натурального ряду.

Програмний код, що реалізує обчислення факторіала двома способами (стандартним та алгоритмічним) і порівнює отримані результати:

```
import math
n = 6
# --- Стандартна функція ---
perm_lib = math.factorial(n)
# --- Алгоритмічна реалізація (цикл) ---
perm_manual = 1
for i in range(1, n + 1):
    perm_manual *= i
```

```
# --- Результати ---
print(f"Вхідні дані: n = {n}")
print(f"Стандартна функція: {perm_lib}")
print(f"Ручний алгоритм: {perm_manual}")
```

Результат програмної реалізації обчислення кількості перестановок:

```
Вхідні дані: n = 6
Стандартна функція: 720
Ручний алгоритм: 720
```

Як видно з результатів виконання програми, обидва методи дають ідентичний результат. Це підтверджує коректність алгоритмічного підходу та демонструє, що математична операція факторіала $n!$ програмно реалізується через цикл накопичення добутку. Хоча у прикладних задачах доцільніше використовувати оптимізовану функцію *math.factorial*, ручна реалізація є необхідною для розуміння природи ітераційних обчислень.

Розміщення (Arrangements)

Розміщеннями з n елементів по k (A_n^k) називаються впорядковані підмножини, де порядок елементів має суттєве значення. Формула для обчислення має вигляд:

$$A_n^k = \frac{n!}{(n-k)!}$$

Стандартним інструментарієм у сучасних версіях Python є функція *math.perm(n, k)*. Вона обчислює кількість розміщень, оптимізуючи процес так, щоб уникнути обчислення надзвичайно великих чисел, які виникають при знаходженні повних факторіалів.

Алгоритмічний підхід у разі ручної реалізації стає двоскладовим. Необхідно окремо обчислити факторіал чисельника ($n!$) та факторіал знаменника ($(n-k)!$) за допомогою циклів, після чого виконати операцію цілочисельного ділення.

Програмний код, що ілюструє різницю між використанням вбудованої функції та покроковим алгоритмом:

```
import math
n = 5
k = 2
# --- Стандартна функція ---
arrangements_lib = math.perm(n, k)
# --- Алгоритмічна реалізація ---
# Чисельник n!
fact_n = 1
for i in range(1, n + 1):
    fact_n *= i
# Знаменник (n-k)!
limit_nk = n - k
fact_nk = 1
for i in range(1, limit_nk + 1):
```

```

fact_nk *= i
# Ділення (оператор // для цілих чисел)
arrangements_manual = fact_n // fact_nk
# --- Результати ---
print(f"Розміщення A({n},{k}) (стандартна функція): {arrangements_lib}")
print(f"Розміщення A({n},{k}) (ручний алгоритм): {arrangements_manual}")

```

Результат програмної реалізації обчислення кількості розміщень:

```

Розміщення A(5,2) (стандартна функція): 20
Розміщення A(5,2) (ручний алгоритм): 20

```

Аналіз отриманих результатів показує, що алгоритмічний метод, хоч і є більш громіздким, дозволяє чітко простежити виконання формули: спочатку накопичення значень (множення), а потім їх зменшення (ділення). Це є проміжною ланкою до вивчення сполучень.

Сполучення (Combinations)

Сполученнями C_n^k називаються невпорядковані підмножини. Порядок елементів не впливає на результат, тому їх кількість завжди менша за кількість розміщень. Формула для обчислення:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Стандартним інструментарієм є функція `math.comb(n, k)`, яка вважається найбільш ефективним способом обчислення біноміального коефіцієнта в Python.

Алгоритмічний підхід у цьому випадку є найбільш трудомістким, оскільки вимагає обчислення трьох окремих факторіалів: для n , для k та для різниці $n - k$. Це наочно демонструє зростання обчислювальної складності алгоритму при переході від перестановок до сполучень.

Код програми для порівняння стандартного та ручного методів:

```

import math
n = 5
k = 2
# --- Стандартна функція ---
combinations_lib = math.comb(n, k)
# --- Алгоритмічна реалізація ---
# 1. Факторіал n
fact_n = 1
for i in range(1, n + 1):
    fact_n *= i
# 2. Факторіал k
fact_k = 1
for i in range(1, k + 1):
    fact_k *= i
# 3. Факторіал (n-k)
fact_nk = 1
for i in range(1, n - k + 1):

```

```

    fact_nk *= i
# 4. Формула
combinations_manual = fact_n // (fact_k * fact_nk)
# --- Результати ---
print(f"Сполучення          C({n},{k})          (стандартна      функція):
{combinations_lib}")
print(f"Сполучення          C({n},{k})          (ручний        алгоритм):
{combinations_manual}")

```

Результат програмної реалізації обчислення кількості сполучень:

```

Сполучення C(5,2) (стандартна функція): 10
Сполучення C(5,2) (ручний алгоритм):    10

```

Порівняння результатів підтверджує правильність реалізації формули. Варто зазначити, що за алгоритмічного підходу критично важливо використовувати оператор цілочисельного ділення `//`, щоб отримати коректний тип даних на виході.

Програмна реалізація формули включення-виключення

Формула включення-виключення є фундаментальним принципом комбінаторики для знаходження потужності об'єднання множин, що перетинаються. Для випадку двох множин формула має вигляд:

$$|A \cup B| = |A| + |B| - |A \cap B|.$$

У разі простого додавання потужностей $|A| + |B|$ елементи, що належать їхньому перетину $A \cap B$, враховуються двічі. Для отримання коректної кількості унікальних елементів необхідно відняти кількість елементів перетину.

У цьому випадку стандартні бібліотечні функції не використовуються, оскільки задача вирішується базовими арифметичними операціями, проте програмна реалізація демонструє застосування теоретико-множинних принципів у коді.

```

# Вхідні дані: потужності множин
cardinality_A = 15      # |A|
cardinality_B = 10      # |B|
intersection = 4         # |A ∩ B|
# --- Помилковий шлях (проста сума) ---
naive_sum = cardinality_A + cardinality_B
# --- Коректний шлях (формула включення-виключення) ---
union_size = cardinality_A + cardinality_B - intersection
# --- Результати ---
print(f"Проста сума (з дублюванням): {naive_sum}")
print(f"Коректне об'єднання |A ∪ B|: {union_size}")

```

Результат програмної реалізації формули включення-виключення:

```

Проста сума (з дублюванням): 25
Коректне об'єднання |A ∪ B|: 21

```

Цей приклад ілюструє, що програмний код у комбінаториці виступає інструментом формалізації математичної моделі: комп'ютер виконує арифметичні дії, але правильність результату залежить від закладеної логіки.

Перевірка коректності введення даних

Важливим етапом створення надійної програми є забезпечення її стійкості до помилок користувача. Якщо під час запиту числових даних для розрахунків (наприклад, n або k) буде введено текст або неприпустимі символи, виконання програми перерветься системною помилкою *ValueError*.

Для запобігання цьому використовується конструкція обробки винятків *try...except* у поєднанні з циклом *while*.

Алгоритм працює таким чином.

1. Здійснюється спроба перетворити введені дані на ціле число.
2. У разі успіху цикл переривається.
3. У разі помилки виводиться повідомлення і запит повторюється.

Фрагмент коду, що реалізує цей механізм:

```
print("Введення даних для розрахунку")
while True:
    # Запит даних
    user_input = input("Введіть ціле число n: ")
    try:
        # Спроба перетворити введений текст у число
        n = int(user_input)
        # Якщо перетворення успішне, вихід з циклу
        break
    except ValueError:
        # Цей блок спрацьовує, якщо введено не число
        print("Помилка! Будь ласка, введіть ціле число.")
# Виведення результату після успішної перевірки
print(f"Прийнято коректне значення: n = {n}")
```

Результат роботи алгоритму перевірки коректності вхідних даних:

```
Введення даних для розрахунку
Введіть ціле число n: а
Помилка! Будь ласка, введіть ціле число.
Введіть ціле число n: 1.1
Помилка! Будь ласка, введіть ціле число.
Введіть ціле число n: 1,1
Помилка! Будь ласка, введіть ціле число.
Введіть ціле число n: 1
Прийнято коректне значення: n = 1
```

Такий підхід дозволяє створити програму, яка ігнорує некоректні дані та продовжує роботу до моменту отримання правильного числового значення, що є обов'язковою вимогою для якісного програмного продукту.

7. ДОДАТКОВІ ПОЛОЖЕННЯ

7.1. Метод зваженої суми для розв'язування багатокритеріальних задач

Метод зваженої суми (Weighted Sum Model, WSM) – один із найпростіших і найпоширеніших методів прийняття багатокритеріальних і багатоальтернативних рішень. Кожна альтернатива оцінюється за кількома критеріями, кожному критерію призначається вага, а підсумковий бал обчислюється як зважена сума.

Нехай:

- A_i – альтернатива;
- C_j – критерій ($j = 1..m$);
- w_j – вага критерію ($\sum w_j = 1$);
- x_{ij} – оцінка альтернативи A_i за критерієм C_j ,

тоді узагальнена оцінка альтернативи:

$$S_i = \sum w_j * x_{ij}.$$

Кращою вважається альтернатива з найбільшим значенням S_i .

Приклад

Вибір ноутбука за критеріями:

ціна (C_1)

швидкодія (C_2)

автономність (C_3)

Ваги:

$$w_1 = 0.4$$

$$w_2 = 0.35$$

$$w_3 = 0.25$$

Оцінки (нормалізовані):

Альтернатива	Ціна, (C_1)	Швидкодія, (C_2)	Автономність, (C_3)
A_1	0.9	0.7	0.6
A_2	0.6	0.9	0.8
A_3	0.7	0.6	0.9

Розрахунки для:

$$A_1: S_1 = (0.4 \cdot 0.9) + (0.35 \cdot 0.7) + (0.25 \cdot 0.6) = 0.36 + 0.245 + 0.15 = 0.755.$$

$$A_2: S_2 = (0.4 \cdot 0.6) + (0.35 \cdot 0.9) + (0.25 \cdot 0.8) = 0.24 + 0.315 + 0.20 = 0.755.$$

$$A_3: S_3 = (0.4 \cdot 0.7) + (0.35 \cdot 0.6) + (0.25 \cdot 0.9) = 0.28 + 0.21 + 0.225 = 0.715.$$

Отже, найкращими альтернативами є A_1 та A_2 , бо вони мають найбільші значення узагальненої оцінки.

Алгоритм:

- визначити набір альтернатив та критеріїв;
- задати ваги для критеріїв (сума ваг = 1);
- нормалізувати оцінки;
- обчислити зважену суму для кожної альтернативи;
- обрати альтернативу з найбільшим значенням.

7.1.1. Реалізація алгоритму мовою Python

```
import numpy as np # Матриця оцінок (альтернативи × критерії)
X = np.array([
    [0.9, 0.7, 0.6], # A1
    [0.6, 0.9, 0.8], # A2
    [0.7, 0.6, 0.9] ]) # A3
w = np.array([0.4, 0.35, 0.25]) # Ваги критеріїв
scores = X.dot(w) # Обчислення зважених сум
for i, s in enumerate(scores, start=1): # Вивід результатів
    print(f"A{i}: {s:.3f}")
print("Найкращі альтернативи:", np.where(np.isclose(scores,
scores.max()))[0] + 1)
```

Тестування працездатності програми

Щоб переконатися, що наведений Python-код працює коректно, проведемо кілька тестів: від простих (тривіальних) до складніших (нетривіальних), які перевіряють роботу в крайніх або нетипових умовах.

Базові перевірки, які показують, що програма видає очікуваний результат на стандартних даних:

– результат роботи програми збігається з *ручним розрахунком*, наведеним у прикладі:

Вхідні дані: матриця оцінок X та ваги w з прикладу.

```
import numpy as np
# Матриця оцінок (Альтернативи A1-A3)
X = np.array([
    [0.5, 0.6, 0.4], # A1
    [0.9, 0.9, 0.8], # A2 - Явний лідер
    [0.7, 0.5, 0.6]  # A3
])
# Ваги критеріїв
w = np.array([0.4, 0.35, 0.25])
# Обчислення зважених сум
scores = X.dot(w)
# Вивід результатів
for i, s in enumerate(scores, start=1):
    print(f"A{i}: {s:.3f}")
# Пошук найкращої альтернативи
print("Найкращі альтернативи:", np.where(np.isclose(scores,
scores.max()))[0] + 1)
```

Перевірка на даних з основного прикладу:

```
A1: 0.510
A2: 0.875
A3: 0.605
Найкращі альтернативи: [2]
```

Результат повністю відповідає розрахункам з прикладу:

$S_1 = 0.755$, $S_2 = 0.755$, $S_3 = 0.715$.

Програма правильно визначила, що альтернативи A_1 та A_2 є однаково найкращими.

– програма правильно визначає єдиного лідера, коли *одна з альтернатив значно краща за інші*:

Вхідні дані: Альтернатива A_2 має найвищі оцінки за всіма критеріями.

```
import numpy as np
# Матриця оцінок (Альтернативи A1-A3)
X = np.array([
    [0.5, 0.6, 0.4], # A1
    [0.9, 0.9, 0.8], # A2 - Явний лідер
    [0.7, 0.5, 0.6]  # A3
])
# Ваги критеріїв
w = np.array([0.4, 0.35, 0.25])
# Обчислення зважених сум
scores = X.dot(w)
# Вивід результатів
for i, s in enumerate(scores, start=1):
    print(f"A{i}: {s:.3f}")
# Пошук найкращої альтернативи
print("Найкращі альтернативи:", np.where(np.isclose(scores,
scores.max()))[0] + 1)
```

Результат оброблення випадку з очевидним переможцем:

```
A1: 0.510
A2: 0.875
A3: 0.605
Найкращі альтернативи: [2]
```

Програма безпомилково визначила альтернативу A_2 як найкращу, що підтверджує коректність логіки ранжування.

Виконаємо тести, що перевіряють, як програма поводить себе в нестандартних ситуаціях:

– робота алгоритму, коли *є лише один критерій*, у такому випадку вага цього критерію дорівнює 1, а найкращою має стати альтернатива з найвищою оцінкою:

Вхідні дані: матриця X має лише один стовпець.

```
import numpy as np
# Матриця оцінок (A1-A3, лише один критерій)
X = np.array([
    [0.8], # A1
    [0.6], # A2
    [0.9]  # A3 - Найвища оцінка
])
w = np.array([1.0]) # Вага єдиного критерію (повна значущість)
scores = X.dot(w) # Обчислення зважених сум
for i, s in enumerate(scores, start=1): # Вивід результатів
    print(f"A{i}: {s:.3f}")
print("Найкращі альтернативи:", np.where(np.isclose(scores,
scores.max()))[0] + 1)
```

Результат обробки випадку з лише одним критерієм для оцінки:

```
A1: 0.800
A2: 0.600
A3: 0.900
Найкращі альтернативи: [3]
```

Програма правильно впоралася із крайнім випадком, обравши альтернативу A_3 , яка має максимальну оцінку за єдиним критерієм.

– перевірка *стійкості алгоритму* (хоча за методологією сума ваг має дорівнювати 1, математично програма має працювати, рейтинг альтернатив не повинен змінитися):

Вхідні дані: сума ваг w дорівнює $0.5 + 0.6 + 0.7 = 1.8$.

```
import numpy as np
# Матриця оцінок (Альтернативи A1-A2)
X = np.array([
    [0.9, 0.7, 0.6], # A1
    [0.6, 0.9, 0.8] # A2
])
w = np.array([0.5, 0.6, 0.7]) # Неправильні ваги (сума > 1)
scores = X.dot(w) # Обчислення зважених сум
for i, s in enumerate(scores, start=1): # Вивід результатів
    print(f"A{i}: {s:.3f}")
# Пошук найкращої альтернативи
print("Найкращі альтернативи:", np.where(np.isclose(scores,
scores.max()))[0] + 1)
```

Результат оброблення випадку, коли ваги критеріїв не сумуються до 1:

```
A1: 1.290
A2: 1.400
Найкращі альтернативи: [2]
```

Хоча абсолютні значення оцінок змінилися (вони більше не є нормалізованими), програма все одно коректно визначила відносний рейтинг. Альтернатива A_2 є кращою. Це показує, що сам механізм множення матриць працює надійно.

– робота алгоритму з оцінками за межами стандартного діапазону $[0, 1]$, наприклад, якщо критерій означає «втрати», а не «вигоди»:

Вхідні дані: матриця X містить від'ємні значення.

```
import numpy as np
# Матриця оцінок (A1-A2) з від'ємними значеннями
X = np.array([
    [0.9, -0.8, 0.5], # A1
    [0.7, 0.2, -0.4] # A2
])
w = np.array([0.5, 0.3, 0.2]) # Ваги критеріїв
scores = X.dot(w) # Обчислення зважених сум
for i, s in enumerate(scores, start=1): # Вивід результатів
    print(f"A{i}: {s:.3f}")
print("Найкращі альтернативи:", np.where(np.isclose(scores,
scores.max()))[0] + 1)
```

Результат обробки випадку використання від'ємних оцінок:

A1: 0.310

A2: 0.330

Найкращі альтернативи: [2]

Програма коректно виконує математичні операції навіть з нетиповими даними. Алгоритм є універсальним і не залежить від діапазону оцінок.

Ці тести демонструють, що наведений код не лише правильно реалізує метод зваженої суми для стандартних завдань, але й є достатньо надійним для оброблення крайніх та нестандартних випадків.

7.1.2. Практичні завдання для самостійної роботи

Наведені нижче завдання розроблено для закріплення теоретичних знань про метод зваженої суми (WSM) та набуття практичних навичок прийняття багатокритеріальних рішень. Виконання цих вправ передбачає проходження повного циклу аналізу: від формалізації критеріїв та нормування даних до програмної реалізації алгоритму та порівняння отриманих результатів з альтернативними методами.

Завдання 1. Вибір автомобіля за методом WSM

Необхідно провести розрахунок та обрати найкращий варіант автомобіля з трьох альтернатив.

Критерії оцінювання:

- 1) ціна (мінімізація);
- 2) витрата палива (мінімізація);
- 3) рівень безпеки (максимізація).

Сформуувати матрицю рішень, визначити вагові коефіцієнти для кожного критерію та розрахувати інтегральну оцінку для кожної альтернативи.

Завдання 2. Моделювання вибору закладу вищої освіти

Сформулювати задачу прийняття рішень для вибору університету.

1. Визначити не менше 4-х критеріїв (наприклад: вартість контракту, рейтинг викладачів, наявність гуртожитку, локація).
2. Встановити власні вагові коефіцієнти, що відображають особисті пріоритети.
3. Оцінити 3-4 реальні або гіпотетичні університети та визначити переможця за методом WSM.

Завдання 3. Програмна реалізація методу

Розробити скрипт мовою Python для автоматизації розрахунків.

Вимоги до функції:

- 1) вхідні дані: матриця оцінок альтернатив (нормалізована) та вектор вагових коефіцієнтів;
- 2) логіка: обчислення суми добутків оцінок на ваги для кожного рядка матриці;
- 3) вихідні дані: індекс або назва найкращої альтернативи.

Завдання 4. Порівняльний аналіз методів

Провести порівняння результатів, отриманих методом зваженої суми (WSM), із результатами методу найкращої та найгіршої альтернативи (Best-Worst Method – BWM).

Визначити, чи змінюється ранжування альтернатив у разі зміни математичного апарату оцінювання, та проаналізувати чутливість результатів.

Структурна схема методу

Для кращого розуміння потоків даних у методі WSM наводиться графічна схема процесу (рис. 7.1). Вона демонструє, як вагові коефіцієнти критеріїв (w) впливають на формування оцінки кожної альтернативи (A).

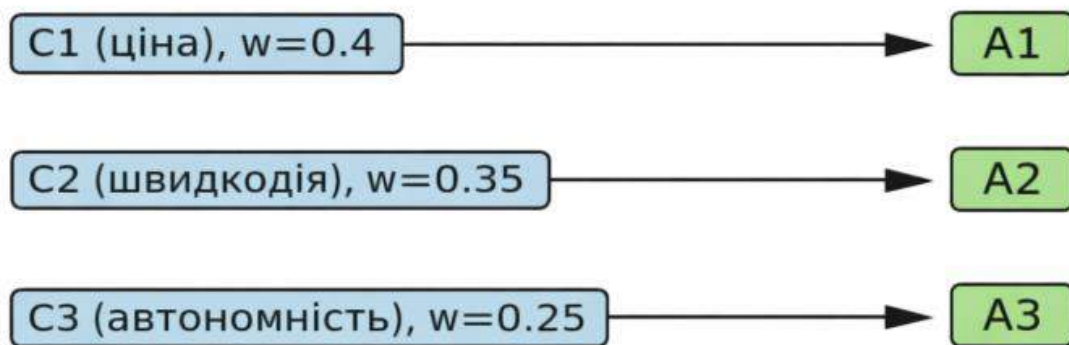


Рис. 7.1. Структурна схема методу зваженої суми (WSM)

На схемі відображено пряму залежність: вхідні параметри (критерії C_1, C_2, C_3 з відповідними вагами) трансформуються в інтегральні оцінки альтернатив (A_1, A_2, A_3) шляхом лінійної згортки.

7.2. Зворотний польський запис

Зворотний польський запис (ЗПЗ), або постфіксна нотація (Reverse Polish Notation, RPN), – це форма запису математичних виразів, у якій знаки операцій розташовані після своїх операндів.

Ключовою особливістю ЗПЗ є відсутність необхідності у використанні дужок для визначення пріоритету операцій. Порядок обчислень однозначно задається порядком слідування елементів.

Порівняння нотацій

Для виразу, що у звичній формі записується як $(3 + 4) \times 5$, існують такі варіанти запису.

1. **Інфіксний (традиційний):** $(3 + 4) \times 5$.

Оператор знаходиться між операндами. Порядок дій регулюється дужками та пріоритетом операцій.

2. **Префіксний (польський):** $\times + 3 4 5$.

Оператор записується перед операндами.

3. **Постфіксний (зворотний польський):** $3 4 + 5 \times$.

Оператор записується після операндів.

Алгоритм обчислення

Обчислення виразів у ЗПЗ найефективніше реалізується з використанням структури даних **стек**. Алгоритм оброблення рядка виглядає так:

1. Програма читає вираз зліва направо.
2. Якщо зустрічається **число (операнд)**, воно поміщається на вершину стека (операція *push*).
3. Якщо зустрічається **оператор** (наприклад, +, -, ×, /):
 - зі стека вилучаються два верхніх числа (операція *pop*);
 - над ними виконується відповідна математична дія;
 - результат обчислення поміщається назад у стек.
4. Після завершення перегляду виразу у стеку залишається одне число – результат.

Приклад обчислення виразу $3\ 4\ +\ 5\ \times$

1. Читаємо 3: стек [3].
2. Читаємо 4: стек [3, 4].
3. Читаємо +: беремо 4 і 3, додаємо ($3 + 4 = 7$), результат у стек: [7].
4. Читаємо 5: стек [7, 5].
5. Читаємо ×: беремо 5 і 7, множимо ($7 \times 5 = 35$), результат у стек: [35].

Відповідь: 35.

Приклади перетворення виразів

Для кращого розуміння відмінностей між інфіксним записом та ЗПЗ розглянемо декілька прикладів різної складності (табл. 7.1).

Таблиця 7.1.

Порівняння інфіксного запису та ЗПЗ

Складність	Інфіксний запис (звичний)	Зворотний польський запис (ЗПЗ)	Коментар
Простий	$2 + 3$	$2\ 3\ +$	Оператор ставиться після операндів.
3 пріоритетом	$(2 + 3) \times 4$	$2\ 3\ +\ 4\ \times$	Спочатку виконується додавання, результат множиться на 4.
Змішаний	$5 + (6 \times 7)$	$5\ 6\ 7\ \times\ +$	Спочатку множення (6 і 7), потім додавання 5 до результату.
Складний	$(3 + 4) \times (5 - 2)$	$3\ 4\ +\ 5\ 2\ -\ \times$	Дві незалежні дії в дужках виконуються, потім їх результати перемножуються.

Алгоритм «сортувальної станції» (Shunting Yard)

Для автоматичного перетворення виразів з інфіксного запису на постфіксний (ЗПЗ) використовується *алгоритм «сортувальної станції»*, розроблений Едсгером Дейкстрою. Назва походить від аналогії із залізничною станцією, де потяги (токени) розподіляються по різних коліях (стек та вихідна черга).

Принцип роботи алгоритму

1. Підготовка. Використовуються дві структури даних:

- *вихідний рядок (черга)* – сюди записується результат (ЗПЗ);
- *стек операторів* – для тимчасового зберігання знаків операцій

(+, −, *, (тощо).

2. Покрокове оброблення. Вираз читається зліва направо, по одному елементу (токену):

- якщо токен – це число: одразу додається у вихідний рядок;
- якщо токен – це відкривальна дужка (: кладеться в стек;
- якщо токен – це закривальна дужка): зі стека виштовхуються всі оператори у вихідний рядок доти, доки не зустрінеться відкривальна дужка (. Самі дужки у вихідний рядок не потрапляють;
- якщо токен – це оператор (+, −, *, /):

- перевіряється пріоритет поточного оператора з тим, що знаходиться на вершині стека;

- якщо на вершині стека лежить оператор з *вищим або рівним* пріоритетом (наприклад, * проти +), він переміщується зі стека у вихідний рядок;

- поточний оператор кладеться в стек.

3. Завершення. Коли вхідний рядок закінчився, всі оператори, що залишилися у стеку, послідовно переносяться у вихідний рядок.

Цей алгоритм є фундаментальним для роботи компіляторів та інтерпретаторів математичних виразів.

7.2.1. Програмна реалізація на Python

Код програми реалізує функції обчислення простої контрольної суми, доповнення до модуля та алгоритму CRC-8, а також демонструє порівняння результатів цих методів.

```
def infix_to_rpn(expression):  
    """
```

```
    Перетворює математичний вираз з інфіксної форми у ЗПЗ,  
    використовуючи алгоритм сортувальної станції.  
    """
```

```
    precedence = {'+': 1, '-': 1, '*': 2, '/': 2}  
    output = []  
    stack = []  
    tokens = expression.split()  
    for token in tokens:
```

```
        if token.isnumeric():
```



```

        output.append(token)
    elif token in precedence:
        while stack and stack[-1] in precedence and
precedence[stack[-1]] >= precedence[token]:
            output.append(stack.pop())
            stack.append(token)
    elif token == '(':
        stack.append(token)
    elif token == ')':
        while stack and stack[-1] != '(':
            output.append(stack.pop())
        # Видаляємо відповідну відкриваючу дужку зі стека
        if stack and stack[-1] == '(':
            stack.pop()
        else:
            # Якщо дужка не знайдена, вираз некоректний
            raise ValueError("Помилка: невідповідність дужок")
# В кінці виштовхуємо всі оператори, що залишилися в стеку
while stack:
    if stack[-1] == '(':
        # Якщо в стеку залишилися дужки, вираз некоректний
        raise ValueError("Помилка: невідповідність дужок")
    output.append(stack.pop())
return output
def evaluate_rpn(tokens):
    """
    Обчислює вираз, поданий у зворотному польському записі,
    використовуючи стек.
    """
    stack = []
    for token in tokens:
        if token.isnumeric():
            stack.append(int(token))
        else:
            if len(stack) < 2:
                raise ValueError("Помилка: недостатньо операндів
для оператора")
            b = stack.pop()
            a = stack.pop()
            if token == '+':
                stack.append(a + b)
            elif token == '-':
                stack.append(a - b)
            elif token == '*':
                stack.append(a * b)
            elif token == '/':
                if b == 0:
                    raise ZeroDivisionError("Помилка: ділення на
нуль")
                stack.append(a / b)
    if len(stack) != 1:
        raise ValueError("Помилка: некоректний вираз")
    return stack[0]

```

```

if __name__ == "__main__":
    print("Простий калькулятор. Для виходу введіть 'вихід'.")
    print("Вводьте вираз, розділяючи числа та оператори пробілами  
(напр. 5 * ( 2 + 3 ) )")
    # Головний цикл, який дозволяє вводити вирази послідовно
    while True:
        expr = input("\nВведіть вираз > ")
        # Перевірка умови виходу з програми
        if expr.lower() in ['вихід', 'exit', 'quit']:
            print("Завершення роботи.")
            break
        # Блок обробки помилок для захисту програми від "падіння"
        try:
            print(f"Вхідний вираз: \"{expr}\"")
            rpn_tokens = infix_to_rpn(expr)
            print(f"Зворотний польський запис: {rpn_tokens}")
            result = evaluate_rpn(rpn_tokens)
            print(f"Результат обчислення: {result}")
        except (ValueError, IndexError, ZeroDivisionError) as e:
            # Виведення повідомлення у разі відомої помилки
            print(f"Не вдалося обчислити вираз. {e}")
        except Exception as e:
            # Виведення повідомлення для будь-яких інших
            # непередбачених помилок
            print(f"Сталася невідома помилка: {e}")
    finally:
        print("-" * 30)

```

Тестування працездатності програми

Щоб переконатися, що обидві частини програми (перетворення на ЗПЗ та обчислення) працюють коректно, проведемо серію тестів.

Виконаємо тривіальні тести, які підтверджують, що найголовніші правила математики – пріоритет операцій та оброблення дужок – працюють:

– перевірка базового пріоритету операцій (алгоритм має «розуміти», що множення важливіше за додавання, і відкласти операцію додавання на потім) наведено на рис. 7.2.

```

Простий калькулятор. Для виходу введіть 'вихід'.
Вводьте вираз, розділяючи числа та оператори пробілами (напр. 5 * ( 2 + 3 ) )

Введіть вираз > 5 + 2 * 3
Вхідний вираз: "5 + 2 * 3"
Зворотний польський запис: ['5', '2', '3', '*', '+']
Результат обчислення: 11
-----

Введіть вираз > вихід
Завершення роботи.

```

Рис. 7.2. Простий вираз без дужок

Програма правильно дотримується пріоритету операцій. Оператор «+» чекав у стеку, поки не завершилася операція з вищим пріоритетом «*».

– перевірка, що дужки примусово змінюють порядок операцій, дужка (діє як тимчасова межа у стеку, змушуючи всі операції всередині неї виконатися першими (рис. 7.3).

```
Простий калькулятор. Для виходу введіть 'вихід'.
Вводьте вираз, розділяючи числа та оператори пробілами (напр. 5 * ( 2 + 3 ) )

Введіть вираз > ( 5 + 2 ) * 3
Вхідний вираз: "( 5 + 2 ) * 3"
Зворотний польський запис: ['5', '2', '+', '3', '*']
Результат обчислення: 21
-----

Введіть вираз > вихід
Завершення роботи.
```

Рис. 7.3. Простий вираз з дужками

Дужки успішно ізолювали операцію «+» і змусили її виконатися раніше за «*». Закриваюча дужка «)» спрацювала як сигнал для виштовхування всіх операторів до відкриваючої дужки «(».

Перевіримо, як алгоритм поводить себе зі складнішими структурами та нетиповими вхідними даними, що є ключем до підтвердження його надійності:

– як програма керує стеком для оброблення довгого ланцюжка операцій зліва направо, постійно порівнюючи пріоритети (рис. 7.4).

```
Простий калькулятор. Для виходу введіть 'вихід'.
Вводьте вираз, розділяючи числа та оператори пробілами (напр. 5 * ( 2 + 3 ) )

Введіть вираз > 10 - 2 * 3 + 8 / 4
Вхідний вираз: "10 - 2 * 3 + 8 / 4"
Зворотний польський запис: ['10', '2', '3', '*', '-', '8', '4', '/', '+']
Результат обчислення: 6.0
-----

Введіть вираз > вихід
Завершення роботи.
```

Рис. 7.4. Довгий вираз із різними пріоритетами

Програма коректно обробляє безперервний потік токенів, правильно виштовхуючи оператори зі стека, коли надходить новий оператор з нижчим або таким самим пріоритетом.

– головний стрес-тест для логіки оброблення дужок. Алгоритм має правильно обробити внутрішні дужки, перш ніж перейти до зовнішніх, що демонструє принцип «*Last – In, First – Out*» (LIFO) роботи стека (рис. 7.5).

```

Простий калькулятор. Для виходу введіть 'вихід'.
Вводьте вираз, розділяючи числа та оператори пробілами (напр. 5 * ( 2 + 3 ) )

Введіть вираз > 5 * ( ( 2 + 3 ) * 4 - 10 )
Вхідний вираз: "5 * ( ( 2 + 3 ) * 4 - 10 )"
Зворотний польський запис: ['5', '2', '3', '+', '4', '*', '10', '-', '*']
Результат обчислення: 50
-----

Введіть вираз > вихід
Завершення роботи.

```

Рис. 7.5. Вираз із вкладеними дужками

Вкладені дужки обробляються бездоганно. Стек накопичує оператори та дужки, а потім коректно «розмотує» їх у правильному порядку.

– перевірка крайнього випадку, коли весь вираз є однією великою групою (це гарантує, що алгоритм не видасть помилку, якщо (є першим токеном, а) – останнім) показано на рис. 7.6.

```

Простий калькулятор. Для виходу введіть 'вихід'.
Вводьте вираз, розділяючи числа та оператори пробілами (напр. 5 * ( 2 + 3 ) )

Введіть вираз > ( 5 * 3 - 3 )
Вхідний вираз: "( 5 * 3 - 3 )"
Зворотний польський запис: ['5', '3', '*', '3', '-']
Результат обчислення: 12
-----

Введіть вираз > вихід
Завершення роботи.

```

Рис. 7.6. Вираз, що починається і закінчується дужками

Алгоритм коректно обробляє вираз і в кінці повністю очищує стек, не залишаючи в ньому дужок.

– переконання, що програма є гнучкою і не залежить від довжини чисел, оскільки ми розбиваємо рядок на токени за допомогою `.split()`, програма має однаково сприймати 3 і 300 (рис. 7.7).

```

Простий калькулятор. Для виходу введіть 'вихід'.
Вводьте вираз, розділяючи числа та оператори пробілами (напр. 5 * ( 2 + 3 ) )

Введіть вираз > 100 + 200 / 10 - 5 * 2
Вхідний вираз: "100 + 200 / 10 - 5 * 2"
Зворотний польський запис: ['100', '200', '10', '/', '+', '5', '2', '*', '-']
Результат обчислення: 110.0
-----

Введіть вираз > вихід
Завершення роботи.

```

Рис. 7.7. Вираз із одноцифровими та багатоцифровими числами

Програма надійно працює з числами будь-якої довжини, що підтверджує правильність підходу з токенізацією.

Ці тести показують, що реалізація алгоритму для перетворення та обчислювача ЗПЗ є надійною і правильно працює як на простих, так і на складніших математичних виразах.

7.2.2. Практичні завдання для самостійної роботи

Наведені нижче завдання розроблено для закріплення навичок роботи зі стековими структурами даних та алгоритмами парсингу. Виконання цих вправ дозволить на практиці опанувати принципи перетворення арифметичних виразів на постфіксну нотацію (ЗПЗ) та реалізувати логіку стекового калькулятора, що є базою для побудови компіляторів та інтерпретаторів.

Завдання 1. Перетворення інфіксних виразів на ЗПЗ

Необхідно перевести подані арифметичні вирази зі звичної (інфіксної) форми у зворотний польський запис (постфіксну нотацію), дотримуючись пріоритету операцій.

Вираз А: $7 - (2 + 3) \times 4$

Вираз Б: $(8 / (4 - 2)) + 5$

Використати алгоритм «сортувальної станції» або метод логічного розбору пріоритетів (дужки $\rightarrow$ множення / ділення $\rightarrow$ додавання / віднімання).

Завдання 2. Обчислення значень у постфіксній нотації

Провести ручне обчислення виразу, записаного у ЗПЗ, моделюючи роботу стека.

Вхідний вираз: $6\ 2\ /\ 3\ +$

Послідовно обробити токени зліва направо, виконуючи операції над останніми двома числами у стеку. Зафіксувати проміжні стани стека.

Завдання 3. Програмна реалізація стекового калькулятора

Розробити скрипт мовою Python, що реалізує обчислення арифметичних виразів, поданих у зворотному польському записі.

Вимоги до функції:

- Вхідні дані: рядок або список токенів у форматі ЗПЗ.
- Логіка: використання списку як стека (*append*, *pop*).
- Підтримка операцій: $+$, $-$, $*$, $/$.
- Вихідні дані: числовий результат обчислення.

Графічна інтерпретація алгоритмів

Для візуалізації процесів, що відбуваються під час оброблення виразів, наводяться схеми роботи стека та алгоритму перетворення.

1. Обчислення виразу у ЗПЗ

У табл. 7.2 продемонстровано покрокову зміну стану стека під час обчислення виразу $3\ 4\ +\ 5\ \times$.

Таблиця 7.2

Робота стеку при обчисленні виразу $3\ 4\ +\ 5\ \times$ у постфіксній нотації

Крок (Дія)	Опис операції	Стан стеку
Читаємо 3	Операнд додається у стек	['3']
Читаємо 4	Операнд додається у стек	['3', '4']
Читаємо +	Вилучаємо 3 та 4, виконуємо $3 + 4 = 7$, результат у стек	['7']
Читаємо 5	Операнд додається у стек	['7', '5']
Читаємо ×	Вилучаємо 7 та 5, виконуємо $7 \times 5 = 35$, результат у стек	['35']

2. Перетворення інфіксного виразу на 3ПЗ

У табл. 7.3 наведено трасування алгоритму «сортувальної станції» для перетворення виразу $(3 + 4) * 5$ на постфіксну форму. Схема відображає розподіл токенів між вихідною чергою та стеком операторів.

Таблиця 7.3

Трасування алгоритму Дейкстри («сортувальна станція»)

Крок (Дія)	Опис операції	Поточний вихід (Output)	Стан стека (Stack)
Читаємо 3	Операнд йде на вихід	[3]	[]
Читаємо +	Оператор йде у стек	[3]	[+]
Читаємо 4	Операнд йде на вихід	[3, 4]	[+]
Кінець дужки	Виштовхуємо оператор зі стека	[3, 4, +]	[]
Читаємо *	Оператор йде у стек	[3, 4, +]	[*]
Читаємо 5	Операнд йде на вихід	[3, 4, +, 5]	[*]
Кінець виразу	Виштовхуємо все зі стека	[3, 4, +, 5, *]	[]

7.3. Математична індукція

Математична індукція (Mathematical Induction) – це фундаментальний метод доведення математичних тверджень, істинність яких залежить від натурального параметра n . Цей метод дозволяє логічно перейти від перевірки окремого початкового випадку до загального висновку, справедливого для нескінченної множини чисел.

Інтуїтивно принцип математичної індукції можна порівняти з «ефектом доміно»:

1. Якщо ми штовхнемо першу кісточку (база індукції), вона впаде.
2. Якщо падіння довільної кісточки k гарантує падіння наступної кісточки $k + 1$ (індукційний перехід), то...
3. ...неминуче впадуть абсолютно всі кісточки в ряду.

Схема доведення

Для доведення твердження $P(n)$, яке має бути істинним для всіх натуральних $n \geq n_0$, необхідно виконати такі кроки.

1. База індукції (Base case)

Необхідно перевірити істинність твердження для початкового значення $n = n_0$ (найчастіше $n = 1$). Це «поштовх першої кісточки доміно».

2. Індукційне припущення (Inductive hypothesis)

Припускаємо, що твердження $P(k)$ є істинним для деякого довільного натурального числа $k \geq n_0$.

3. Індукційний перехід (Inductive step)

На основі зробленого припущення необхідно довести, що твердження є істинним і для наступного значення $n = k + 1$, тобто доводимо імплікацію $P(k) \Rightarrow P(k + 1)$.

4. Висновок (Conclusion)

Якщо база істинна і перехід доведений, то на основі принципу математичної індукції твердження $P(n)$ є істинним для всіх $n \geq n_0$.

Приклади застосування

Приклад 1. Сума арифметичної прогресії. Довести формулу суми перших n натуральних чисел:

$$1 + 2 + \dots + n = \frac{n(n+1)}{2}.$$

Доведення

1. База індукції ($n = 1$):

Ліва частина: 1.

Права частина: $\frac{1(1+1)}{2} = 1$.

$1 = 1$ – істинно.

2. Індукційне припущення ($n = k$):

Припустимо, що формула вірна для k :

$$1 + 2 + \dots + k = \frac{k(k+1)}{2}.$$

3. Індукційний перехід ($n = k + 1$):

Потрібно довести, що сума для $k + 1$ дорівнює $\frac{(k+1)(k+2)}{2}$.

Розглянемо суму для $k + 1$ доданків:

$$S_{k+1} = \underbrace{(1 + 2 + \dots + k)}_{\text{За припущенням}} + (k + 1),$$

замінімо вираз у дужках на формулу з кроку 2:

$$S_{k+1} = \frac{k(k+1)}{2} + (k+1),$$

звівши до спільного знаменника, отримуємо:

$$\frac{k(k+1) + 2(k+1)}{2} = \frac{(k+1)(k+2)}{2}.$$

Отримали вираз, що відповідає правій частині формули $n = k + 1$.

4. Висновок: формула вірна для будь-якого натурального n .

Приклад 2. Доведення нерівності. Довести, що

$$2^n \geq n + 1$$

для всіх $n \geq 1$.

Доведення

1. База індукції ($n = 1$):

$$2^1 \geq 1 + 1 \Rightarrow 2 \geq 2$$

істинно.

2. Індукційне припущення ($n = k$):

Нехай $2^k \geq k + 1$.

3. Індукційний перехід ($n = k + 1$).

Потрібно довести, що $2^{k+1} \geq k + 2$. Помножимо ліву частину припущення на 2:

$$2^{k+1} = 2 \cdot 2^k \geq 2(k+1) = 2k + 2.$$

Оскільки $k \geq 1$, то $2k + 2 > k + 2$.

4. Висновок: нерівність виконується.

7.3.1. Практичні завдання для самостійної роботи

Наведені нижче завдання розроблено для опанування методу математичної індукції – потужного інструменту доведення тверджень, що залежать від натурального параметра.

Метою роботи є формування навичок побудови індуктивних переходів, роботи з рядами та нерівностями, а також програмної верифікації математичних гіпотез.

Завдання 1. Сума непарних чисел. Використовуючи метод математичної індукції, доведіть, що:

$$1 + 3 + 5 + \dots + (2n - 1) = n^2.$$

Завдання 2. Факторіал та степінь. Доведіть нерівність для всіх $n \geq 1$:

$$n! \geq 2^{n-1}.$$

Завдання 3. Сума квадратів. Доведіть формулу:

$$1^2 + 2^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}.$$

Завдання 4. Програмування. Реалізуйте на Python програму для перевірки формули:

$$1 + 2 + \dots + n = \frac{n(n+1)}{2}.$$

7.3.2. Програмна реалізація на Python

Програмний код не є формальним доведенням, але дозволяє швидко перевірити гіпотезу на великій кількості значень.

```
def check_formula(N):  
    """  
    Перевіряє формулу суми перших n чисел для всіх n від 1 до N.  
    """  
    # Перевіряємо, чи є N цілим числом  
    if not isinstance(N, int):  
        print("Помилка: N має бути цілим числом.")  
        return  
    print(f"\n--- Запуск перевірки для N = {N} ---")  
    if N < 1:  
        print("Цикл не буде виконано, оскільки N < 1.")  
        # Цикл, що перебирає всі числа від 1 до N включно  
        for n in range(1, N + 1):  
            # 1. Ліва частина: пряме обчислення суми  
            left_side = sum(range(1, n + 1))  
            # 2. Права частина: обчислення за формулою Гаусса  
            right_side = n * (n + 1) // 2  
            # 3. Вивід результатів порівняння  
            print(f"n={n}: сума={left_side}, формула={right_side},  
збір={left_side == right_side}")  
            print("-" * (25 + len(str(N))))
```



```

if __name__ == "__main__":
    print("Програма для перевірки формули суми  $1+2+...+n = n*(n+1)/2$  .")
    print("Для виходу з програми введіть 'вихід'.")
    while True:
        # Запитуємо ввід від користувача
        user_input = input("\nВведіть ціле число N для перевірки > ")
        # Перевіряємо, чи хоче користувач вийти
        if user_input.lower() in ['вихід', 'exit', 'quit']:
            print("Завершення роботи.")
            break
        # Обробляємо можливі помилки вводу
        try:
            # Намагаємось перетворити ввід у ціле число
            number_N = int(user_input)
            # Викликаємо нашу функцію для перевірки
            check_formula(number_N)
        except ValueError:
            # Якщо користувач ввів не число, повідомляємо про помилку
            print(f"Помилка: '{user_input}' не є цілим числом. Спробуйте ще раз.")

```

Тестування працездатності програми

Перевірка роботи програми допоможе переконатися, що програма працює коректно для різних значень n .

Розглянемо прості випадки, які підтверджують, що основна логіка програми працює правильно для стандартних позитивних чисел:

- перевірка коректної роботи програми для невеликого набору послідовних чисел, де результат легко перевірити вручну (рис. 7.8).

```

Програма для перевірки формули суми  $1+2+...+n = n*(n+1)/2$ .
Для виходу з програми введіть 'вихід'.

Введіть ціле число N для перевірки > 5

--- Запуск перевірки для N = 5 ---
n=1: сума=1, формула=1, збіг=True
-----
n=2: сума=3, формула=3, збіг=True
-----
n=3: сума=6, формула=6, збіг=True
-----
n=4: сума=10, формула=10, збіг=True
-----
n=5: сума=15, формула=15, збіг=True
-----

Введіть ціле число N для перевірки > вихід
Завершення роботи.

```

Рис. 7.8. Стандартна перевірка ($N = 5$)

Програма успішно підтвердила, що формула є правильною для всіх чисел від 1 до 5.

– перевірка граничного випадку – найменше можливе значення n , для якого формула має сенс.

Сума першого одного числа – це саме це число (рис. 7.9).

```
Програма для перевірки формули суми  $1+2+\dots+n = n*(n+1)/2$ .  
Для виходу з програми введіть 'вихід'.  
  
Введіть ціле число N для перевірки > 1  
  
--- Запуск перевірки для N = 1 ---  
n=1: сума=1, формула=1, збіг=True  
-----  
  
Введіть ціле число N для перевірки > вихід  
Завершення роботи.
```

Рис. 7.9. Перевірка для єдиного значення ($N = 1$)

Програма коректно обробляє базовий випадок, підтверджуючи формулу для $N = 1$.

Тести для перевірки роботи програми з незвичайними вхідними даними, що допомагає визначити межі її застосування:

– перевірка поведінки програми, коли верхня межа циклу не дозволяє йому виконатися жодного разу.

Цикл $range(1, 1)$ є порожнім (рис. 7.10).

```
Програма для перевірки формули суми  $1+2+\dots+n = n*(n+1)/2$ .  
Для виходу з програми введіть 'вихід'.  
  
Введіть ціле число N для перевірки > 0  
  
--- Запуск перевірки для N = 0 ---  
Цикл не буде виконано, оскільки  $N < 1$ .  
  
Введіть ціле число N для перевірки > вихід  
Завершення роботи.
```

Рис. 7.10. Перевірка для нульового значення ($N = 0$)

Програма поводить ся передбачувано. Оскільки цикл $for n in range(1, 1)$ не виконується, жодного рядка з обчисленнями не виводиться. Це коректна поведінка.

– перевірка, що формула залишається правильною для великих значень n і що в Python немає проблем із обчисленням великих цілих чисел (рис. 7.11).

```
➡ Програма для перевірки формули суми  $1+2+\dots+n = n*(n+1)/2$ .
   Для виходу з програми введіть 'вихід'.

   Введіть ціле число N для перевірки > 1000

   --- Запуск перевірки для N = 1000 ---
   n=1: сума=1, формула=1, збіг=True
   n=2: сума=3, формула=3, збіг=True
   ...
   n=1000: сума=500500, формула=500500, збіг=True
   -----

   Введіть ціле число N для перевірки > вихід
   Завершення роботи.
```

Рис. 7.11. Перевірка для великого числа ($N = 1000$)

Програма підтверджує, що формула працює і для великих чисел.

– перевірка, як програма реагує на логічно некоректні вхідні дані (сума перших -5 чисел не визначена в контексті цієї формули) показана на рис. 7.12.

```
Програма для перевірки формули суми  $1+2+\dots+n = n*(n+1)/2$ .
Для виходу з програми введіть 'вихід'.

Введіть ціле число N для перевірки > -5

--- Запуск перевірки для N = -5 ---
Цикл не буде виконано, оскільки  $N < 1$ .

Введіть ціле число N для перевірки > вихід
Завершення роботи.
```

Рис. 7.12. Перевірка для від'ємного значення ($N = -5$)

Як і у випадку з $N = 0$, програма не «ламається», а просто не входить у цикл, оскільки $\text{range}(1, -4)$ є порожнім. Це безпечна та передбачувана поведінка.

Виконані тести підтверджують, що програма є надійною: вона коректно перевіряє формулу для будь-яких додатних чисел і стабільно працює з нетиповими вхідними даними, як-от нуль чи від'ємні числа.

7.4. Відстань Хеммінга

Відстань Хеммінга (Hamming distance) – це метрика, яка дозволяє оцінити ступінь відмінності між двома об'єктами (рядками, векторами, послідовностями) **однакової довжини**. Числове значення цієї відстані дорівнює кількості позицій, у яких відповідні символи двох рядків є різними.

Формально для двох рядків s_1 та s_2 довжини n відстань d_H визначається як сума фактів неспівпадіння:

$$d_H(s_1, s_2) = \sum_{i=1}^n [s_1[i] \neq s_2[i]],$$

де вираз у квадратних дужках дорівнює 1, якщо символи різні, і 0, якщо вони збігаються.

Приклади та застосування

Для кращого розуміння розглянемо конкретні пари рядків:

– 1011101 та 1001001: розбіжності у 3-й, 4-й та 6-й позиціях.

Відстань $d_H = 3$;

– *karolin* та *kathrin*: символи відрізняються у 3-й, 4-й та 5-й позиціях ($r - t, o - h, l - r$). Відстань $d_H = 3$.

Сфери практичного застосування

1. Теорія кодування (використовується для контролю та виправлення помилок під час передавання даних (коди Хеммінга). Дозволяє визначити, скільки бітів було спотворено шумом у каналі зв'язку).

2. Біоінформатика та генетика (порівняння нуклеотидних послідовностей ДНК для оцінки генетичної спорідненості або мутацій).

3. Машинне навчання (використовується як міра подібності для бінарних векторів ознак або для класифікації текстових даних).

Алгоритм обчислення

Процес знаходження відстані Хеммінга складається з трьох кроків:

1) валідація (перевірити, чи мають вхідні рядки однакову довжину: якщо довжини різні, обчислення метрики Хеммінга неможливе);

2) ітерація (пройти циклом по всіх позиціях від 0 до $n - 1$;

3) акумуляція (для кожної позиції порівняти символи: якщо вони не рівні, збільшити лічильник результату на 1).

Геометрична інтерпретація

Відстань Хеммінга має чіткий геометричний зміст. Якщо розглядати бінарні вектори як координати вершин одиничного n -вимірного куба, то ця відстань дорівнює мінімальній кількості ребер, які необхідно пройти, щоб дістатися від однієї вершини до іншої.

Наприклад, у тривимірному просторі:

– Відстань між $(0,0,0)$ та $(0,0,1)$ дорівнює 1 (суміжні вершини).

– Відстань між протилежними вершинами куба $(0,0,0)$ та $(1,1,1)$ дорівнює 3.

Це наочно демонструє, що чим більша відстань Хеммінга, тим сильніше рознесені дані у просторі станів, що прямо впливає на здатність коду протистояти завадам.

7.4.1. Програмна реалізація на Python

Наведено код програми мовою Python, що реалізує функцію обчислення відстані та інтерфейс взаємодії з користувачем.

```
def hamming_distance(s1, s2):
    """
    Обчислює відстань Хеммінга між двома рядками.
    """
    # Перевірка на однакову довжину. Якщо довжини різні,
    # функція генерує виняток ValueError.

    if len(s1) != len(s2):
        raise ValueError("Помилка: Рядки повинні мати однакову
довжину.")
    # Посимвольне порівняння та підсумовування розбіжностей.
    return sum(c1 != c2 for c1, c2 in zip(s1, s2))
if __name__ == "__main__":
    print("Програма для обчислення відстані Хеммінга.")
    print("Для виходу з програми введіть 'вихід'.")
    while True:
        # Запитуємо ввід від користувача
        string1 = input("\nВведіть перший рядок > ")
        # Перевіряємо, чи хоче користувач вийти
        if string1.lower() in ['вихід', 'exit', 'quit']:
            print("Завершення роботи.")
            break
        string2 = input("Введіть другий рядок > ")
        if string2.lower() in ['вихід', 'exit', 'quit']:
            print("Завершення роботи.")
            break
        # Блок обробки помилок для захисту програми від "падіння"
        try:
            # Викликаємо нашу функцію для обчислення
            distance = hamming_distance(string1, string2)
            print(f"-> Відстань Хеммінга: {distance}")
        except ValueError as e:
            # Якщо функція згенерувала помилку (різна довжина), виводимо
            print(f"-> {e}")
        finally:
            print("-" * 30)
```



Тестування працездатності програми

Щоб переконатися, що програма для обчислення відстані Хеммінга працює коректно, проведемо серію тестів. Ми перевіримо як основну логіку обчислення, так і вбудований захист від даних різної довжини.

Виконаємо тривіальні тести, які підтверджують, що найголовніші правила обчислення відстані, тобто правильний підрахунок розбіжностей для ідентичних та абсолютно різних рядків, працюють:

- перевірка, що для двох однакових рядків відстань дорівнює 0. Це базовий випадок (рис. 7.13).

```
Програма для обчислення відстані Хеммінга.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > identychni  
Введіть другий рядок > identychni  
-> Відстань Хеммінга: 0  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.13. Повністю ідентичні рядки

Програма правильно визначає, що розбіжностей між однаковими рядками немає.

– перевірка, що для двох повністю різних рядків (однакової довжини) відстань дорівнює їхній довжині (рис. 7.14).

```
Програма для обчислення відстані Хеммінга.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > ABCDE  
Введіть другий рядок > FGHIJ  
-> Відстань Хеммінга: 5  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.14. Повністю різні рядки

Програма коректно підраховувала, що всі п'ять символів у рядках відрізняються.

Тести, що перевіряють, як програма поводить себе з незвичайними даними та чи правильно працює її система захисту (обробка помилок):

– перевірка, що порівняння є чутливим до регістру символів (велика літера не дорівнює малій) показана на рис. 7.15.

```
Програма для обчислення відстані Хеммінга.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > Python  
Введіть другий рядок > python  
-> Відстань Хеммінга: 1  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.15. Рядки, чутливі до регістру

Програма правильно враховує регістр, вважаючи «P» та «p» різними символами.

– перевірка, що функція працює не лише з літерами, а з будь-якими символами: цифрами, пробілами, знаками пунктуації (рис. 7.16).

```
Програма для обчислення відстані Хеммінга.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > test_v1.0 !  
Введіть другий рядок > test-v2.0 ?  
-> Відстань Хеммінга: 3  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.16. Рядки з різними типами символів

Алгоритм універсальний і коректно порівнює будь-які пари символів. Розбіжності: «_» і «-», «1» і «2», «!» і «?».

– перевірка граничного випадку з порожніми рядками: відстань між ними має дорівнювати 0 (рис. 7.17).

```
Програма для обчислення відстані Хеммінга.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок >  
Введіть другий рядок >  
-> Відстань Хеммінга: 0  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.17. Порожні рядки

Програма коректно обробляє порожні рядки, не викликаючи помилок.

– перевірка, що вбудований захист працює і програма генерує очікувану помилку *ValueError*, якщо довжини рядків не збігаються (рис. 7.18).

```
Програма для обчислення відстані Хеммінга.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > short  
Введіть другий рядок > longer  
-> Помилка: Рядки повинні мати однакову довжину.  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.18. Рядки різної довжини (перевірка захисту)

Механізм оброблення помилок працює. Програма не намагається обчислити відстань для некоректних даних, а натомість повідомляє про проблему.

Проведені тести показують, що програма правильно обчислює відстань для широкого діапазону даних (разом із різними символами, порожніми рядками та з урахуванням регістру) і, що важливо, має вбудований захист від некоректних вхідних даних, що робить її безпечною для використання.

7.4.2. Практичні завдання для самостійної роботи

Наведені нижче завдання розроблено для закріплення теоретичних знань про метричні простори та набуття практичних навичок обчислення відстані Хеммінга.

Виконання цих вправ дозволить опанувати методи оцінювання відмінностей між знаковими послідовностями, що є критично важливим для теорії кодування, виявлення помилок та біоінформатики.

Завдання 1. Бінарні обчислення. Обчисліть вручну відстань Хеммінга між бітовими послідовностями 11001001 та 10010011. Перевірте результат програмно.

Завдання 2. Лінгвістичне порівняння. Яка відстань Хеммінга між словами машина і малина? Визначте позиції, що відрізняються.

Завдання 3. Пошук найближчої пари. Реалізуйте функцію, яка приймає на вхід список бінарних кодів і знаходить серед них пару (два елементи), відстань Хеммінга між якими є мінімальною.

Завдання 4. Проєктування завадостійкого коду. Уявіть, що ви розробляєте протокол зв'язку для керування роботом. Вам потрібно обрати два 8-бітні коди для команд «ВПЕРЕД» та «НАЗАД».

– Підберіть коди так, щоб відстань Хеммінга між ними була максимальною ($d = 8$). Обґрунтуйте свій вибір.

– Чому використання кодів із великою відстанню Хеммінга (наприклад, 00000000 та 11111111) є надійнішим за використання схожих кодів (наприклад, 00000000 та 00000001) в умовах цифрових перешкод?

Графічна ілюстрація

На схемі (рис. 7.19) зображено процес порівняння рядків 1011101 та 1001001. Червоним кольором виділено комірки, де значення бітів не збігаються (індекси 2, 3, 6 при нумерації з нуля), що в сумі дає відстань 3. Зеленим позначено позиції, де символи ідентичні.

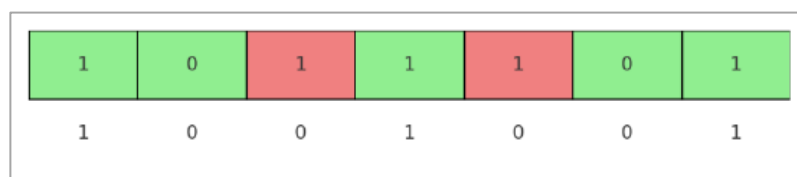


Рис. 7.19. Схема посимвольного порівняння рядків

7.5. Відстань Левенштейна

Відстань Левенштейна (Levenshtein distance) – це метрика редагування рядків, яка визначає мінімальну кількість односимвольних операцій, необхідних для перетворення одного рядка на інший.

Дозволеними операціями є:

- вставка (Insertion) символу.
- видалення (Deletion) символу.
- заміна (Substitution) одного символу на інший.

Ця метрика була запропонована радянським математиком Володимиром Левенштейном у 1965 році.

Сфери застосування

Завдяки своїй універсальності алгоритм широко використовується у:

- обробленні природної мови (NLP) (перевірка правопису (spell-checkers) та автокорекція);
- пошукових системах (реалізація функції «Можливо, ви мали на увазі...» (fuzzy search));
- біоінформатиці (порівняння послідовностей ДНК та білків для виявлення мутацій);
- розпізнаванні мовлення (оцінка точності транскрибування аудіо в текст).

Приклади обчислення

Розглянемо класичні приклади, щоб зрозуміти суть метрики до вивчення формул.

Приклад 1. Перетворення слова 'kitten' на 'sitting'.

Відстань Левенштейна = 3.

Послідовність змін:

- *kitten* → *sitten* (Заміна *k* → *s*)
- *sitten* → *sittin* (Заміна *e* → *i*)
- *sittin* → *sitting* (Вставка *g* в кінець)

Приклад 2. Перетворення слова 'робот' на 'робота'.

Відстань Левенштейна = 1.

Послідовність змін:

- *робот* → *робота* (Вставка *а* в кінець)

Алгоритм обчислення (метод динамічного програмування)

Для знаходження відстані Левенштейна використовується алгоритм Вагнера-Фішера. Суть методу полягає у розбитті складної задачі на простіші: щоб знайти відстань між повними рядками, ми поступово обчислюємо відстані між їхніми частинами (префіксами).

Для цього будується матриця D розміром $(n + 1) \times (m + 1)$, де n та m – довжини першого (s_1) та другого (s_2) рядків.

Елемент матриці $D[i][j]$ зберігає значення відстані Левенштейна між першими i символами рядка s_1 та першими j символами рядка s_2 .

Математична модель алгоритму описується системою рекурентних співвідношень.

1. База індукції (граничні умови).

Ці умови заповнюють перший рядок і перший стовпець матриці. Вони описують ситуацію, коли один із рядків порожній:

– $D[i][0] = i$ – відстань між рядком довжиною i та порожнім рядком дорівнює i . Єдиний спосіб перетворити слово на «ніщо» – це *видалити* всі його символи;

– $D[0][j] = j$ – відстань між порожнім рядком та рядком довжиною j дорівнює j . Щоб отримати слово з «нічого», потрібно *вставити* всі його символи.

2. Крок рекурсії (основна логіка)

Для обчислення значення $D[i][j]$ (де $i > 0, j > 0$) ми порівнюємо поточні символи $s1[i]$ та $s2[j]$:

– випадок, коли символи збігаються ($s1[i] == s2[j]$).

Якщо символи однакові, вартість редагування не збільшується. Ми просто копіюємо результат для попередніх префіксів:

$$D[i][j] = D[i-1][j-1];$$

– випадок, коли символи різні ($s1[i] \neq s2[j]$):).

Якщо символи відрізняються, ми повинні виконати одну дію редагування. Ми обираємо ту операцію, яка дасть мінімальну сумарну відстань, і додаємо 1 (штраф за операцію):

$$D[i][j] = 1 + \min \begin{cases} D[i-1][j] & \text{(видалення)} \\ D[i][j-1] & \text{(вставка)} \\ D[i-1][j-1] & \text{(заміна)} \end{cases}$$

Інтерпретація операцій у матриці

Щоб зрозуміти принцип роботи формули мінімуму, розглянемо, звідки алгоритм бере дані в матриці:

– Видалення (Deletion) – значення береться з комірки зверху ($D[i-1][j]$).

«Видаляємо» зайвий символ із $s1$ і повертаємося до стану, де цей рядок був коротшим.

– Вставка (Insertion) – значення береться з комірки зліва ($D[i][j-1]$).

«Вставляємо» потрібний символ у $s1$, щоб він зрівнявся з $s2$, і дивимось на вартість для попереднього стану $s2$.

– Заміна (Substitution) – значення береться з комірки по діагоналі ($D[i-1][j-1]$).

«Замінюємо» поточний символ $s1$ на символ $s2$. Після цього вони стають однаковими, і ми спираємось на результат для префіксів без цих символів.

7.5.1. Програмна реалізація на Python

Нижче наведено код програми мовою Python, що реалізує функцію обчислення відстані за алгоритмом Вагнера-Фішера та простий інтерфейс взаємодії з користувачем.

```
def levenshtein(s1, s2):
    """
    Обчислює відстань Левенштейна між двома рядками s1 та s2,
    використовуючи алгоритм Вагнера-Фішера.
    """
    n, m = len(s1), len(s2)
    # Створення матриці D для динамічного програмування
    D = [[0] * (m + 1) for _ in range(n + 1)]
    # Ініціалізація першого стовпця та першого рядка
    for i in range(n + 1):
        D[i][0] = i # Вартість видалення всіх символів s1
    for j in range(m + 1):
        D[0][j] = j # Вартість вставки всіх символів s2
    # Заповнення решти матриці
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            # Вартість заміни: 1, якщо символи різні, 0 - якщо однакові
            cost = 0 if s1[i-1] == s2[j-1] else 1
            # Вибір мінімальної вартості з трьох можливих операцій
            D[i][j] = min(
                D[i-1][j] + 1,          # Операція видалення
                D[i][j-1] + 1,          # Операція вставки
                D[i-1][j-1] + cost      # Операція заміни (або символи
збігаються)
            )
    # Результат знаходиться в останній клітинці матриці
    return D[n][m]
if __name__ == "__main__":
    print("Програма для обчислення відстані Левенштейна.")
    print("Для виходу з програми введіть 'вихід'.")
    while True:
        # Запитуємо ввід від користувача
        string1 = input("\nВведіть перший рядок > ")
        # Перевіряємо, чи хоче користувач вийти
        if string1.lower() in ['вихід', 'exit', 'quit']:
            print("Завершення роботи.")
            break
        string2 = input("Введіть другий рядок > ")
        if string2.lower() in ['вихід', 'exit', 'quit']:
            print("Завершення роботи.")
            break
        # Викликаємо функцію та виводимо результат
        try:
            distance = levenshtein(string1, string2)
            print(f"-> Відстань Левенштейна: {distance}")
        except Exception as e:
            print(f"Сталася непередбачена помилка: {e}")
        finally:
            print("-" * 30)
```



Тестування працездатності програми

Щоб переконатися у коректності роботи програми, проведемо серію тестів для перевірки знаходження мінімальної кількості редагувань, реалізуючи три ключові операції: вставку, видалення та заміну символів.

Виконаємо тривіальні тести, які підтверджують, що найголовніші правила обчислення відстані, а саме коректна «вартість» кожної базової операції та правильне оброблення ідентичних рядків, працюють:

– перевірка, що для однакових рядків відстань дорівнює 0, оскільки жодних змін не потрібно (рис. 7.20)).

```
Програма для обчислення відстані Левенштейна.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > тестування  
Введіть другий рядок > тестування  
-> Відстань Левенштейна: 0  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.20. Ідентичні рядки

Програма правильно реалізує базову логіку, повертаючи нульову відстань для ідентичних рядків, оскільки збіг символів коректно обробляється з нульовою вартістю.

– перевірка, що додавання одного символу в кінець рядка дає відстань 1 (рис. 7.21).

```
Програма для обчислення відстані Левенштейна.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > слон  
Введіть другий рядок > слони  
-> Відстань Левенштейна: 1  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.21. Одна вставка

Програма правильно розраховує вартість операції вставки, повертаючи відстань 1, оскільки для перетворення «слон» на «слони» потрібна лише одна елементарна дія – додавання символу.

– перевірка, що видалення одного символу дає відстань 1 (рис. 7.22).

```

Програма для обчислення відстані Левенштейна.
Для виходу з програми введіть 'вихід'.

Введіть перший рядок > документ
Введіть другий рядок > доумент
-> Відстань Левенштейна: 1
-----

Введіть перший рядок > вихід
Завершення роботи.

```

Рис. 7.22. Одне видалення

Алгоритм коректно обчислює операцію видалення, оскільки результат 1 точно відображає необхідність однієї дії для перетворення «документ» на «доумент» шляхом видалення символу «к».

– перевірка, що заміна одного символу на інший дає відстань 1 (рис. 7.23).

```

Програма для обчислення відстані Левенштейна.
Для виходу з програми введіть 'вихід'.

Введіть перший рядок > книга
Введіть другий рядок > кпига
-> Відстань Левенштейна: 1
-----

Введіть перший рядок > вихід
Завершення роботи.

```

Рис. 7.23. Одна заміна

Програма правильно обробляє операцію заміни, оскільки результат 1 відповідає одній дії, де символ «н» замінюється на «п» з вартістю $cost = 1$.

Тести для перевірки, як алгоритм справляється з порожніми рядками, різним регістром та складними комбінаціями операцій:

– перевірка граничного випадку, коли один рядок порожній, відстань має дорівнювати довжині іншого рядка (кількість вставок / видалень) показана на рис. 7.24.

```

Програма для обчислення відстані Левенштейна.
Для виходу з програми введіть 'вихід'.

Введіть перший рядок > Слово
Введіть другий рядок >
-> Відстань Левенштейна: 5
-----

Введіть перший рядок > вихід
Завершення роботи.

```

Рис. 7.24. Один з рядків порожній

Програма працює коректно, адже для перетворення порожнього рядка на слово потрібно рівно п'ять операцій вставки, що точно дорівнює довжині непорожнього рядка.

– перевірка, що алгоритм є чутливим до регістру, тобто, наприклад, «С» і «с» вважаються різними символами (рис. 7.25).

```
Програма для обчислення відстані Левенштейна.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > Київ  
Введіть другий рядок > київ  
-> Відстань Левенштейна: 1  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.25. Чутливість до регістру

Коректно виконується посимвольне порівняння, яке є чутливим до регістру, оскільки програма правильно визначає, що для перетворення «Київ» на «київ» потрібна одна операція заміни («К» на «к») з вартістю 1.

– перевірка, як програма знаходить мінімальну кількість операцій для двох суттєво різних слів (рис. 7.26).

Наприклад, для перетворення «Saturday» на «Sunday» потрібно три операції:

1. **Saturday** -> **Sunturday** (заміна «a» на «u» та «a» на «n»)
2. **Sunturday** -> **Sunday** (видалення «t» та «r»)

```
Програма для обчислення відстані Левенштейна.  
Для виходу з програми введіть 'вихід'.  
  
Введіть перший рядок > Saturday  
Введіть другий рядок > Sunday  
-> Відстань Левенштейна: 3  
-----  
  
Введіть перший рядок > вихід  
Завершення роботи.
```

Рис. 7.26. Складна трансформація

Цей тест підкреслює ефективність динамічного програмування, оскільки алгоритм успішно знаходить оптимальний шлях редагування з мінімальною вартістю 3, коректно комбінуючи операції заміни та видалення.

Проведені тести показують, що реалізація алгоритму для обчислення відстані Левенштейна є надійною та повністю коректною.

Програма правильно обробляє як базові операції (вставку, видалення, заміну), так і складніші випадки, разом із порожніми рядками та чутливістю до регістру, що робить її придатним інструментом для задач порівняння текстових даних.

7.5.2. Практичні завдання для самостійної роботи

Наведені нижче завдання розроблено для закріплення теоретичних знань про метрики схожості рядків та алгоритми динамічного програмування.

Метою роботи є опанування методу обчислення відстані Левенштейна, аналіз вартості редагування тексту та реалізація прикладних інструментів для автоматичного виправлення помилок.

Завдання 1. Ручний розрахунок. Побудуйте матрицю переходів та обчисліть відстань Левенштейна між словами 'мама' і 'тато'.

Завдання 2. Аналіз перетворень. Визначте, скільки операцій потрібно для перетворення слова 'distance' на 'instance'. Які саме це операції?

Завдання 3. Програмна реалізація. Реалізуйте функцію, яка виводить не лише числову відстань, але й **послідовність редагувань** (наприклад: "замінити 'd' на 'i'").

Завдання 4. Пошук помилок. Використайте функцію відстані Левенштейна для пошуку найбільш схожого прізвища у списку студентів, якщо користувач увів його з друкарською помилкою (реалізація найпростішого *spell-checker*).

Завдання 5. Модифікація вагових коефіцієнтів. У класичній відстані Левенштейна вартість кожної операції (вставка, видалення, заміна) дорівнює 1.

– Теоретично обґрунтуйте, у яких випадках операція заміни має коштувати більше, ніж операція вставки (наприклад, \$cost=2\$). Як це вплине на фінальний результат розрахунку для пари слів «кіт» та «скат»?

– Як зміна вартості операцій вплине на алгоритм автоматичного виправлення помилок, якщо ми знаємо, що користувач найчастіше пропускає літери, а не вводить зайві?

Ілюстрація до прикладу

На рис. 7.27 наведено правильно заповнену матрицю для класичної задачі перетворення 'kitten' → 'sitting'. Кінцевий результат (відстань 3) знаходиться у правому нижньому куті таблиці. Ви можете використовувати цю схему для самоперевірки розуміння алгоритму, прослідкувавши, як утворюється кожне число.

Матриця обчислення відстані Левенштейна

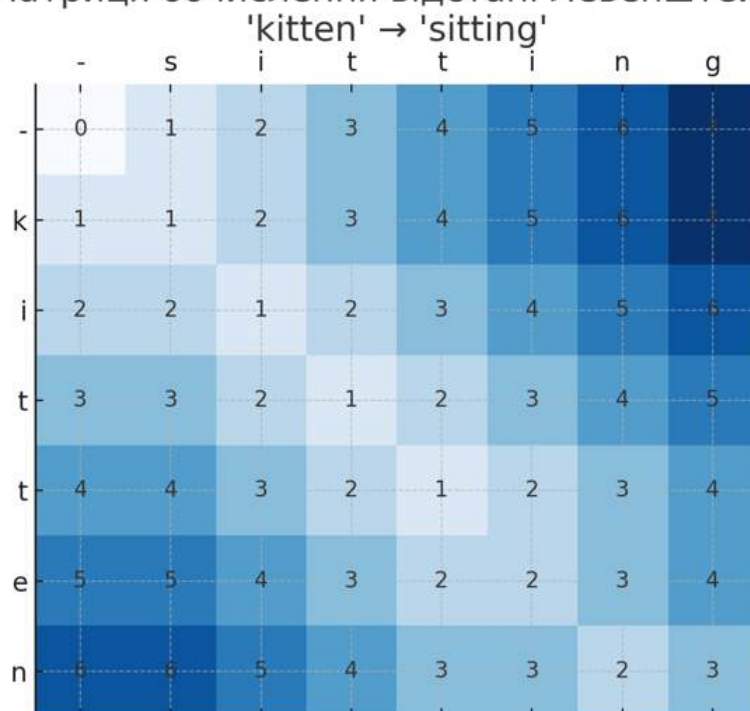


Рис. 7.27. Візуалізація матриці обчислення відстані для слів «kitten» та «sitting»

7.6. Методи контрольного сумування

Контрольна сума (Checksum) – це спеціальне числове значення, яке обчислюється на основі передаваних чи збережених даних і використовується для перевірки їхньої цілісності. Основний принцип роботи:

- відправник обчислює суму для даних і додає її до повідомлення;
- отримувач зчитує дані, за тим самим алгоритмом обчислює нову суму;
- якщо обчислена сума не збігається з отриманою – це означає, що сталася помилка (спотворення даних).

Класифікація методів обчислення

На практиці використовують кілька підходів до формування контрольної суми. Вибір конкретного методу залежить від балансу між швидкістю обчислення та необхідною надійністю виявлення помилок. Розглянемо основні з них у порядку зростання їхньої ефективності.

1. Метод простого сумування (Additive Checksum)

Це найпростіший алгоритм, який часто використовується в мікроконтролерах з обмеженими ресурсами. Його суть полягає у додаванні всіх байтів повідомлення як звичайних чисел. Оскільки результат додавання може перевищити розмір байта, зазвичай використовують лише молодшу частину суми (операція за модулем, наприклад, 256).

Хоча цей метод є надзвичайно швидким, він має суттєвий недолік – низьку завадостійкість. Наприклад, якщо під час передавання два сусідні байти поміняються місцями (помилка перестановки), сума не зміниться, і система помилково вважатиме дані коректними.

2. Метод доповнення до модуля (Modulo-N)

Це вдосконалена версія сумування. Контрольне число підбирається таким чином, щоб сума всіх байтів даних разом із цим числом дорівнювала нулю (або певному фіксованому значенню) за модулем N .

Цей підхід спрощує перевірку на стороні приймача: достатньо просто додати всі отримані байти. Якщо результат не дорівнює нулю – сталася помилка.

3. Циклічний надлишковий код (CRC – Cyclic Redundancy Check)

На сьогодні це стандарт де-факто для телекомунікаційних мереж (Ethernet, Wi-Fi) та систем зберігання даних. В основі методу лежить складна математика: дані розглядаються не як набір чисел, а як коефіцієнти одного великого многочлена.

Цей многочлен ділиться на спеціальний «генераторний поліном», а остача від ділення стає контрольною сумою.

Головна перевага CRC полягає в тому, що він реагує не лише на зміну значення біта, а й на його позицію. Це дозволяє гарантовано виявляти перестановки бітів, а також пакетні помилки (коли пошкоджено групу бітів поспіль), що недоступно для простого сумування.

4. Криптографічні хеш-функції

Для завдань, де критично важлива не лише випадкова помилка, а й захист від навмисної підробки, використовують хеш-функції (наприклад, MD5, SHA-256). Вони створюють унікальний «цифровий відбиток» даних.

На відміну від CRC, імовірність того, що два різні файли матимуть однаковий хеш, є мізерною. Однак через високу обчислювальну складність цей метод рідко застосовують для перевірки даних у реальному часі (наприклад, у потоковому відео), використовуючи його переважно для перевірки цілісності файлів після завантаження.

Критерії ефективності контрольної суми

Для того, щоб метод контрольного сумування вважався надійним, він має відповідати кільком критеріям, які визначають його здатність протидіяти помилкам у каналі зв'язку:

- **Швидкість обчислення** – алгоритм не повинен створювати затримок при передачі поточкових даних.

- **Мінімальний обсяг (Overhead)** – контрольна сума має бути значно меншою за саме повідомлення, щоб не перевантажувати канал зв'язку зайвою службовою інформацією.

- **Ефект лавини (Avalanche effect)** – це критична властивість надійних алгоритмів (особливо CRC та хеш-функцій). Вона полягає в тому, що зміна навіть одного біта у вхідних даних має призводити до зміни принаймні половини бітів у значенні контрольної суми.

Приклади практичного розрахунку

Для розуміння відмінностей між методами розглянемо, як кожний із них обробляє дані на практиці.

1. Приклад простого сумування (Additive Checksum)

Це базовий метод. Його логіка є такою: «додати все і відкинути зайве». Нехай потрібно передати пакет із трьох байтів: [12, 34, 56].

– Виконуємо арифметичне додавання: $12 + 34 + 56 = 102$.

– Результат перевіряється на входження в діапазон байта (0...255).

Оскільки $102 < 256$, переповнення немає.

– Результат: контрольна сума = 102 (або 0x66).

Якщо сума перевищила б 255 (наприклад, дорівнювала б 300), ми взяли б залишок від ділення на 256 ($300 \bmod 256 = 44$).

2. Приклад розрахунку CRC-8 (покроковий трейсинг)

Алгоритм CRC імітує ділення многочленів. Для ручного розрахунку використовують алгоритм зсуву з операцією XOR.

– Вхідний байт: 0xAB (двійковий код 10101011).

– Поліном: 0x07 (стандарт CRC-8, двійковий вигляд 111 для молодших бітів).

На кожному кроці зсуваємо регістр вліво на один розряд. Якщо старший біт, що виходить за межі (MSB), дорівнює 1, над результатом зсуву та поліномом 0x07 виконується операція «виключне АБО» (XOR). Детальний процес перетворення наведено в табл. 7.4.

Таблиця 7.4.

Покроковий розрахунок контрольної суми CRC-8 для байта 0xAB

Крок	Поточне значення (bin)	MSB	Дія (Зсув + XOR?)	Результат (hex)
Старт	10101011	1	Зсув → XOR 0x07	0x51
1	01010001	0	Тільки зсув	0xA2
2	10100010	1	Зсув → XOR 0x07	0x43
3	01000011	0	Тільки зсув	0x86
4	10000110	1	Зсув → XOR 0x07	0x0B
5	00001011	0	Тільки зсув	0x16
6	00010110	0	Тільки зсув	0x2C
7	00101100	0	Тільки зсув	0x58

Результат 0x58 гарантує виявлення будь-яких помилок перестановки бітів, чого не можуть попередні два методи.

3. Приклад доповнення до модуля (Modulo-N)

Цей метод є варіацією попереднього, але оптимізований для спрощення перевірки. Мета – підібрати таке число C , щоб загальна сума дорівнювала нулю (для 8-бітної системи – це $0(mod256)$). Використовуємо ті ж дані: [12, 34, 56].

- Обчислюємо суму даних: 102.
- Шукаємо доповнення до 256: $C = 256 - 102 = 154$.
- Результат: контрольна сума = 154 (або 0x9A).
- Перевірка: приймач просто додає всі числа разом:
 $12 + 34 + 56 + 154 = 256$.

Оскільки в однобайтовому регістрі число 256 стає 0 (переповнення), отримання нуля означає успішну передачу.

Алгоритм перевірки цілісності

Загальний алгоритм використання контрольних сум у системах передачі даних:

- 1) розбити повідомлення на байти;
- 2) обчислити контрольну суму (CRC або просту) на стороні відправника;
- 3) передати дані разом із обчисленою сумою;
- 4) на стороні приймача повторити обчислення для отриманих даних;
- 5) порівняти нову суму з отриманою:
 - збігаються → дані коректні;
 - не збігаються → помилка передачі.

7.6.1. Програмна реалізація на Python

Нижче наведено код, що реалізує обидва методи та інтерфейс для порівняння їхньої роботи.

```
def simple_checksum(data):  
    """Обчислює просту 8-бітну контрольну суму."""  
    return sum(data) % 256  
def crc8(data, poly=0x07):  
    """Обчислює 8-бітний циклічний надлишковий код (CRC-8)."""  
    crc = 0  
    for byte in data:  
        crc ^= byte  
        for _ in range(8):  
            if crc & 0x80:  
                crc = ((crc << 1) ^ poly) & 0xFF  
            else:  
                crc = (crc << 1) & 0xFF  
    return crc  
if __name__ == "__main__":  
    print("Програма для порівняння простої контрольної суми та CRC-8.")  
    print("Для виходу з програми введіть 'вихід'.")  
    while True:  
        # 1. Запитуємо ввід від користувача  
        user_input = input("\nВведіть дані для обчислення > ")
```



```

# 2. Перевіряємо умову виходу
if user_input.lower() in ['вихід', 'exit', 'quit']:
    print("Завершення роботи.")
    break
# 3. Перетворюємо введений текст у послідовність байтів
#     (це необхідно, оскільки алгоритми працюють з байтами,
а не з текстом)
data_bytes = user_input.encode('utf-8')
# 4. Обчислюємо та виводимо результати обох функцій
try:
    checksum_result = simple_checksum(data_bytes)
    crc_result = crc8(data_bytes)
    print(f" -> Введені дані (у байтах):
{list(data_bytes)})")
    print(f" -> Проста контрольна сума: {checksum_result}")
    print(f" -> CRC-8: {crc_result} (або
{hex(crc_result)})")
except Exception as e:
    print(f"Сталася непередбачена помилка: {e}")
finally:
    print("-" * 40)

```

Тестування працездатності програми

Щоб переконатися, що обидві функції для генерації контрольних кодів (*simple_checksum* та *crc8*) працюють коректно, та щоб наочно порівняти їхню надійність, проведемо серію тестів. Перевіримо як основну логіку кожного алгоритму, так і його здатність виявляти поширені типи помилок у даних. Наведений повний код інтерактивної програми, яка буде використовуватися для всіх тестів, дозволяє вводити текстові дані, **перетворює** їх на байти та обчислює для них обидва контрольні коди, що ідеально підходить для порівняння.

Тестування *simple_checksum*

Продемонструємо базову роботу алгоритму та виявимо його ключові слабкості, через які він не є надійним:

- перевірка, що для одного байта сума дорівнює значенню цього байта (рис. 7.28).

Програма для порівняння простої контрольної суми та CRC-8.
Для виходу з програми введіть 'вихід'.

Введіть дані для обчислення > A
-> Введені дані (у байтах): [65]
-> Проста контрольна сума: 65
-> CRC-8: 192 (або 0xc0)

Введіть дані для обчислення > вихід
Завершення роботи.

Рис. 7.28. Перевірка на одному байті

Тест підтверджує, що базова логіка підсумовування для одного елемента працює коректно, оскільки $65 \% 256 = 65$.

– перевірка, що алгоритм нечутливий до порядку байтів, що є його фундаментальним недоліком (рис. 7.29).

```
Програма для порівняння простої контрольної суми та CRC-8.  
Для виходу з програми введіть 'вихід'.  
  
Введіть дані для обчислення > AB  
-> Введені дані (у байтах): [65, 66]  
-> Проста контрольна сума: 131  
-> CRC-8: 135 (або 0x87)  
-----  
  
Введіть дані для обчислення > вихід  
Завершення роботи.
```

Рис. 7.29. Перевірка на перестановку байтів

Тест наочно демонструє слабкість алгоритму. Хоча дані *AB* та *BA* є різними, їхня контрольна сума однакова (131). Це означає, що помилка передачі, яка просто міняє байти місцями, залишиться непоміченою.

– перевірка, що різні помилки в даних можуть «компенсувати» одна одну, залишаючи контрольну суму незмінною (рис. 7.30).

```
Програма для порівняння простої контрольної суми та CRC-8.  
Для виходу з програми введіть 'вихід'.  
  
Введіть дані для обчислення > BC  
-> Введені дані (у байтах): [66, 67]  
-> Проста контрольна сума: 133  
-> CRC-8: 191 (або 0xbf)  
-----  
  
Введіть дані для обчислення > вихід  
Завершення роботи.
```

Рис. 7.30. Перевірка на взаємокомпенсуючі помилки

Цей тест підтверджує ще один недолік. Дані *BC* та *AD* абсолютно різні, але оскільки сума їхніх байтів однакова ($66+67 = 133$ та $65+68 = 133$), контрольна сума збігається. Такі помилки також не будуть виявлені.

Тестування CRC8

Покажемо, що CRC є значно надійнішим і позбавлений недоліків простої контрольної суми:

– перевірка, що наша реалізація CRC-8 відповідає загальноприйнятому стандарту (рис. 7.31).

```

Програма для порівняння простої контрольної суми та CRC-8.
Для виходу з програми введіть 'вихід'.

Введіть дані для обчислення > 123456789
-> Введені дані (у байтах): [49, 50, 51, 52, 53, 54, 55, 56, 57]
-> Проста контрольна сума: 221
-> CRC-8: 244 (або 0xf4)

-----

Введіть дані для обчислення > вихід
Завершення роботи.

```

Рис. 7.31. Перевірка на стандартному векторі, (CRC)

Результат 0xf4 для даних «123456789» з поліномом 0x07 є еталонним. Це підтверджує, що наш алгоритм *crc8* реалізовано коректно.

– перевірка, що CRC, на відміну від простої суми, чутливий до порядку байтів (рис. 7.32).

```

Програма для порівняння простої контрольної суми та CRC-8.
Для виходу з програми введіть 'вихід'.

Введіть дані для обчислення > BC
-> Введені дані (у байтах): [66, 67]
-> Проста контрольна сума: 133
-> CRC-8: 191 (або 0xbf)

-----

Введіть дані для обчислення > CB
-> Введені дані (у байтах): [67, 66]
-> Проста контрольна сума: 133
-> CRC-8: 173 (або 0xad)

-----

Введіть дані для обчислення > вихід
Завершення роботи.

```

Рис. 7.32. Перевірка на перестановку байтів, (CRC)

Тест успішний, бо для різних послідовностей (BC та CB) алгоритм CRC-8 згенерував різні коди (0xbf та 0xad). Це доводить його здатність виявляти помилки, пов'язані з перестановкою даних.

– перевірка, що CRC успішно виявляє помилки, які «маскувалися» під час обчислення простої суми (рис. 7.33).

CRC знову демонструє свою надійність. Для даних CB та DA з однаковою сумою байтів були отримані різні контрольні коди (0xad та 0xcf), що дозволяє виявити помилку.

Проведені тести наочно демонструють, що, хоча проста контрольна сума є легкою в реалізації, вона нездатна виявляти поширені типи помилок, такі як перестановка байтів чи взаємокомпенсуючі спотворення.

```

Програма для порівняння простої контрольної суми та CRC-8.
Для виходу з програми введіть 'вихід'.

Введіть дані для обчислення > CB
-> Введені дані (у байтах): [67, 66]
-> Проста контрольна сума: 133
-> CRC-8: 173 (або 0xad)
-----

Введіть дані для обчислення > DA
-> Введені дані (у байтах): [68, 65]
-> Проста контрольна сума: 133
-> CRC-8: 207 (або 0xcf)
-----

Введіть дані для обчислення > вихід
Завершення роботи.

```

Рис. 7.33. Перевірка на взаємокомпенсуючі помилки, (CRC)

Алгоритм *CRC*, хоч і складніший, є значно надійнішим інструментом контролю цілісності даних, оскільки він чутливий до порядку байтів та інших складних змін. Саме тому *CRC* є промисловим стандартом у протоколах передачі даних (Ethernet, USB, Bluetooth) та системах зберігання.

7.6.2. Практичні завдання для самостійної роботи

Запропоновані завдання спрямовані на практичне освоєння методів контролю цілісності даних. Виконання цих вправ дозволить зрозуміти різницю між арифметичними контрольними сумами та циклічними кодами (*CRC*), а також навчитися виявляти колізії та реалізовувати алгоритми перевірки програмно.

Завдання 1. Розрахунок простої контрольної суми. Дано послідовність із трьох байтів (у десятковій системі): [200, 150, 50].

- Обчисліть суму цих чисел.
- Визначте значення контрольної суми для 8-бітної системи (залиште лише молодший байт, тобто візьміть результат за модулем 256).
- Запишіть отриманий результат у шістнадцятковому форматі.

Завдання 2. Метод доповнення до модуля. Система передачі даних використовує метод «Доповнення до 256» (Modulo-256 Complement). Вхідне повідомлення складається з двох чисел: [80, 120].

- Знайдіть таке число C (контрольну суму), щоб сума всіх елементів $80 + 120 + C$ давала 0 за модулем 256.
- Перевірте свій результат, додавши всі три числа.

Завдання 3. Аналіз вразливостей (колізії). Одним із недоліків простого сумування є нечутливість до перестановки байтів. Знайдіть таку пару різних двобайтових повідомлень (наприклад, $[A, B]$ та $[B, A]$), які мають однакову просту контрольну суму, але різний зміст. Поясніть, чому це є проблемою для надійності передачі даних.

Завдання 4. Ручний розрахунок CRC-8. Створіть таблицю покрокового розрахунку CRC-8 (аналогічну табл. 7.1) для вхідного байта 0xC2 (двійковий код 11000010).

- Поліном: 0x07 (111).
- Виконайте вісім кроків зсуву. Не забувайте виконувати операцію *XOR* з поліномом лише тоді, коли старший біт (MSB), що висувається, дорівнює 1.
- Запишіть кінцеве значення у *hex*-форматі.

Завдання 5. Програмна реалізація. Використайте наведений у розділі код як основу та:

- додайте функцію `checksum_modulo_n(data)`, яка реалізує алгоритм із **Завдання 2** (повертає доповнення до 256);
- модифікуйте блок тестування так, щоб програма виводила результати трьох методів: *Simple Checksum*, *Modulo-N* та *CRC-8*.

Узагальнена схема процесу

Весь розглянутий матеріал можна звести до єдиного логічного алгоритму, зображеного на рис. 7.34. Незалежно від обраного математичного методу (просте сумування, CRC чи хешування), процес контролю цілісності завжди складається з двох етапів: *обчислення* та *порівняння*.

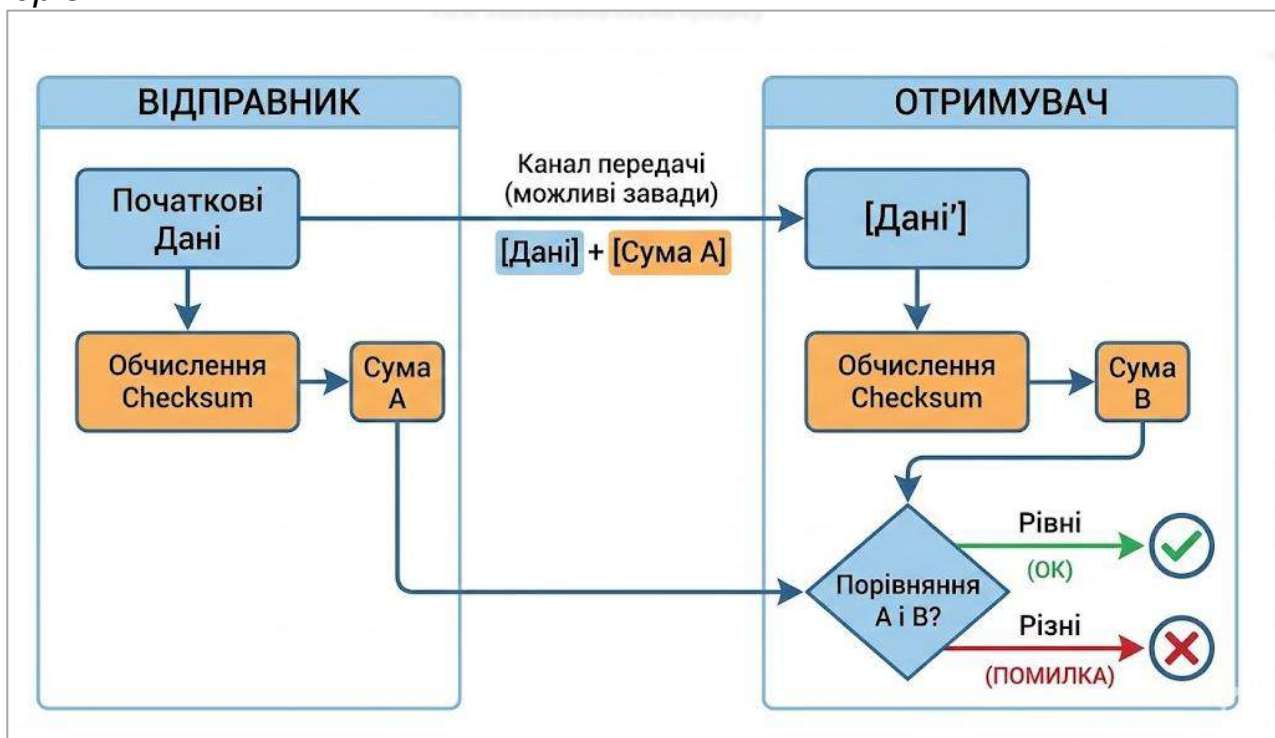


Рис. 7.34. Загальна схема контролю цілісності даних

Пояснення до схеми

1. **Відправник (ліва частина)** бере початкові дані та пропускає їх через алгоритм обчислення (наприклад, CRC-8), отримуючи еталонну контрольну суму (на схемі – **«Сума А»**).

2. Дані разом із «Сумою А» відправляються отримувачу (**Канал передачі**). На цьому етапі можуть виникнути завади, що спотворять інформацію (тому отримувач отримує **[Дані']**, які можуть відрізнятися від початкових).

3. Отримані дані знову пропускаються через той самий алгоритм (**Отримувач (права частина)**). Результат цього обчислення – нова контрольна сума (**«Сума В»**).

4. Система порівнює отриману «Суму А» з обчисленою «Сумою В». (**Порівняння (Блок рішень)**).

- Якщо $A = B$ (зелена стрілка) – дані вважаються коректними.
- Якщо $A \neq B$ (червона стрілка) – фіксується помилка передачі.

7.7. Рекурентні рівняння. Метод Ейлера

У комп'ютерному моделюванні фізичних процесів часто неможливо отримати точну формулу, що описує поведінку системи. Замість цього використовують наближені методи, які базуються на переході від неперервних функцій до дискретних кроків.

Рекурентне рівняння – це формула, яка визначає наступне значення послідовності через попередні. Саме такі рівняння є основою для чисельного розв'язання диференціальних рівнянь.

Більшість процесів руху або змін описуються диференціальними рівняннями (рівняннями, що містять похідні). Однак диференціальне рівняння, узятє окремо, має безліч розв'язків.

Задача Коші полягає у пошуку єдиного конкретного розв'язку диференціального рівняння, який задовольняє заданим початковим умовам. Вона складається з двох компонентів:

- 1) **закон змін** (рівняння): $y' = f(x, y)$ вказує напрямок руху (нахил графіка) у кожній точці;
- 2) **точка старту** (початкова умова): $y(x_0) = y_0$ «прив'язує» розв'язок до конкретного моменту часу або координати.

Геометричний зміст: серед нескінченної кількості інтегральних кривих ми шукаємо ту єдину, що проходить через точку (x_0, y_0) .

Метод Ейлера – це найпростіший метод розв'язання задачі Коші. Його ідея – замінити невідому криву функцію на набір коротких прямих відрізків. Алгоритм базується на розкладанні функції в ряд Тейлора або заміні похідної скінченною різницею:

$$y'(x) \approx \frac{y_{n+1} - y_n}{h}.$$

Отримуємо розрахункові формули методу (рекурентні співвідношення):

$$\begin{aligned}x_{n+1} &= x_n + h \\ y_{n+1} &= y_n + h \cdot f(x_n, y_n),\end{aligned}$$

де h – крок інтегрування (мала величина, на яку змінюється аргумент).

Приклад ручного розрахунку

Розглянемо задачу Коші.

- Рівняння: $y' = x + y$
- Початкова умова: $y(0) = 1$
- Крок: $h = 0.1$
- Завдання: знайти наближене значення y в точці $x = 0.2$.

1. Перехід від x_0 до x_1 : маємо $x_0 = 0$, $y_0 = 1$:
 - обчислюємо значення похідної (нахил): $f(0,1) = 0 + 1 = 1$;
 - обчислюємо нове значення функції: $y_1 = 1 + 0.1 \cdot 1 = 1.1$;
 - новий аргумент: $x_1 = 0.1$.
2. Перехід від x_1 до x_2 : маємо $x_1 = 0.1$, $y_1 = 1.1$:
 - обчислюємо значення похідної: $f(0.1, 1.1) = 0.1 + 1.1 = 1.2$;
 - обчислюємо нове значення функції: $y_2 = 1.1 + 0.1 \cdot 1.2 = 1.22$;
 - новий аргумент: $x_2 = 0.2$.

Результат: $y(0.2) \approx 1.22$.

Точність методу та похибки

Оскільки метод Ейлера замінює криволінійну траєкторію набір ламаних ліній, отриманий розв'язок завжди є наближеним. Відхилення отриманого значення від істинного називається похибкою методу. Розрізняють два види похибок.

1. Локальна похибка (Local Truncation Error)

Це помилка, що виникає на одному окремому кроці інтегрування за умови, що попереднє значення (y_n) було точним. Математично метод Ейлера базується на розкладанні функції в ряд Тейлора, де відкидаються всі доданки, починаючи з другого порядку. Тому локальна похибка пропорційна квадрату кроку:

$$E_{loc} \approx C \cdot h^2 = O(h^2).$$

2. Глобальна похибка (Global Truncation Error)

Це накопичена помилка на всьому інтервалі інтегрування (від x_0 до x_{end}). Оскільки для проходження інтервалу потрібно зробити $N \approx 1/h$ кроків, *локальні* помилки сумуються. Це призводить до зниження порядку точності на одиницю. *Глобальна* похибка пропорційна кроку в першому степені:

$$E_{glob} \approx N \cdot E_{loc} \approx \frac{1}{h} \cdot h^2 = h.$$

Отже, глобальна похибка становить $O(h)$.

Геометрична інтерпретація похибки

На рис. 7.35 видно, як формується розбіжність:

- На кожному кроці дотична (вектор методу Ейлера) трохи відходить від реальної інтегральної кривої.
- На наступному кроці ми починаємо рух вже з хибної точки, тому помилка не просто додається, а «наростає» лавиноподібно.

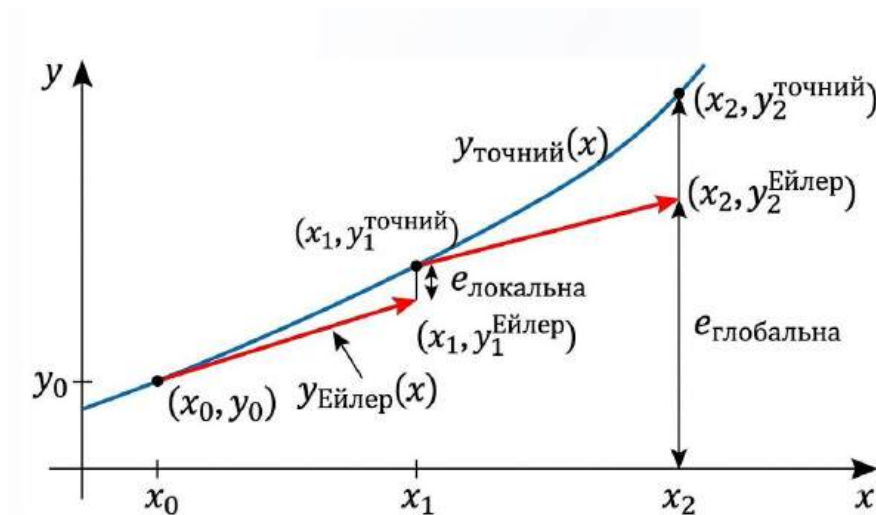


Рис. 7.35. Геометрична інтерпретація локальної та глобальної похибки в методі Ейлера

Порядок точності та обчислювальні витрати

Той факт, що глобальна похибка пропорційна h , класифікує метод Ейлера як метод першого порядку точності.

Це має важливий практичний наслідок: для зменшення похибки результату в 10 разів, необхідно зменшити крок інтегрування h також у 10 разів.

Це лінійна залежність, яка робить метод Ейлера неефективним для задач, що вимагають високої точності, оскільки це призводить до значного зростання кількості обчислень.

Саме через низьку точність ($O(h)$) метод Ейлера в чистому вигляді рідко використовують в інженерних розрахунках, надаючи перевагу методам вищих порядків (наприклад, Рунге-Кутта 4-го порядку, де похибка становить $O(h^4)$).

Поняття стійкості методу

Окрім похибки, критично важливим параметром є чисельна стійкість. Якщо обрати занадто великий крок h , метод Ейлера може почати «розхитуватися»: замість наближення до розв'язку значення почнуть нескінченно зростати (дивергувати). Стійкість залежить від самої функції $f(x, y)$. Для багатьох фізичних систем існує критичне значення кроку h_{crit} , перевищення якого робить моделювання беззмістовним.

Модифікації для підвищення точності

Щоб подолати недоліки методу Ейлера, не переходячи до складних методів Рунге-Кутта, часто використовують вдосконалений метод Ейлера (метод трапецій). Його ідея полягає в тому, що нахил обчислюється не в одній точці, а як середнє арифметичне нахилів на початку та в кінці кроку:

$$y_{n+1} = y_n + \frac{h}{2} [f(x_n, y_n) + f(x_{n+1}, \widetilde{y}_{n+1})]$$

Це підвищує порядок точності до другого ($O(h^2)$), що значно зменшує глобальну похибку при тій же кількості кроків.

7.7.1. Програмна реалізація на Python

Нижче наведено код для чисельного розв'язання задачі Коші та порівняння результату з точним аналітичним розв'язком.

```
import math
def euler(f, x0, y0, h, N):
    """
    Реалізація методу Ейлера для розв'язання  $y' = f(x, y)$ 
    з початковою умовою  $y(x_0) = y_0$ .
    """
    x, y = x0, y0
    xs, ys = [x], [y]

    for _ in range(N):
        y = y + h * f(x, y)
        x = x + h
        xs.append(x)
        ys.append(y)
    return xs, ys

functions = {

    '1': {
        'desc': "y' = 2",
        'func': lambda x, y: 2,
        # Точний розв'язок:  $y(x) = 2*x + C$ , де  $C = y_0 - 2*x_0$ 
        'exact': lambda x, x0, y0: 2*x + (y0 - 2*x0)
    },

    '2': {
        'desc': "y' = x",
        'func': lambda x, y: x,
        # Точний розв'язок:  $y(x) = x^2/2 + C$ , де  $C = y_0 - x_0^2/2$ 
        'exact': lambda x, x0, y0: x**2/2 + (y0 - x0**2/2)
    },

    '3': {
        'desc': "y' = y",
        'func': lambda x, y: y,
        # Точний розв'язок:  $y(x) = C*e^x$ , де  $C = y_0 / e^{x_0}$ 
        'exact': lambda x, x0, y0: (y0 / math.exp(x0)) *
math.exp(x)
    },

    '4': {
        'desc': "y' = x + y",
        'func': lambda x, y: x + y,
        # Точний розв'язок:  $y(x) = C*e^x - x - 1$ , де  $C = (y_0 + x_0 + 1) / e^{x_0}$ 
        'exact': lambda x, x0, y0: ((y0 + x0 + 1) / math.exp(x0)) *
math.exp(x) - x - 1
    }
}
```



```

if __name__ == "__main__":
    print("Програма для розв'язання диференціальних рівнянь методом  

    Ейлера.")
    print("Для виходу з програми введіть 'вихід'.")
    while True:
        print("\nОберіть диференціальне рівняння для розв'язання:")
        for key, value in functions.items():
            print(f" {key}. {value['desc']}")
        print(" q. Вихід")
        choice = input("Ваш вибір > ")
        if choice.lower() in ['q', 'вихід', 'quit']:
            print("Завершення роботи.")
            break
        if choice not in functions:
            print("Помилка: невідомий вибір. Спробуйте ще раз.")
            continue
        selected_task = functions[choice]
        selected_function = selected_task['func']
        exact_solution = selected_task['exact']
        try:
            x0 = float(input("Введіть початкове x0: "))
            y0 = float(input("Введіть початкове y0: "))
            h = float(input("Введіть розмір кроку h: "))
            N = int(input("Введіть кількість кроків N: "))
            if h <= 0 or N < 0:
                print("Помилка: h має бути > 0, а N має бути >=
0.")
                continue
        except ValueError:
            print("Помилка: введено некоректне число. Спробуйте ще
раз.")
            continue
        # Розрахунок та вивід результатів
        xs, ys = euler(selected_function, x0, y0, h, N)
        print("\n--- Результати обчислення ---")
        print(f"{'x':<8} | {'Euler y':<12} | {'Точний y':<12} |  

{'Похибка':<12}")
        print("-" * 50)
        for xi, yi in zip(xs, ys):
            exact_yi = exact_solution(xi, x0, y0)
            error = abs(yi - exact_yi)
            print(f"{xi:<8.2f} | {yi:<12.4f} | {exact_yi:<12.4f} |  

{error:<12.4f}")
        print("-" * 50)

```

Тестування працездатності програми

Щоб переконатися у коректності роботи програми, перевіримо як основну логіку ітераційного кроку, так і загальну поведінку методу на різних типах диференціальних рівнянь, а також його чутливість до розміру кроку.

Виконаємо тести, які підтверджують, що найголовніші властивості методу, а саме здатність точно розв'язувати найпростіші рівняння та давати очікуване наближення для нелінійних випадків, працюють.

– перевірка, що програма видає абсолютно точний результат для лінійної функції (рис. 7.36).

```
Програма для розв'язання диференціальних рівнянь методом Ейлера.  
Для виходу з програми введіть 'вихід' або 'q'.  
  
Оберіть диференціальне рівняння для розв'язання:  
1.  $y' = 2$   
2.  $y' = x$   
3.  $y' = y$   
4.  $y' = x + y$   
q. Вихід  
Ваш вибір > 1  
Введіть початкове  $x_0$ : 0  
Введіть початкове  $y_0$ : 0  
Введіть розмір кроку  $h$ : 0.5  
Введіть кількість кроків  $N$ : 4  
  
--- Результати обчислення ---  
x          | Euler y      | Точний y      | Похибка  
-----  
0.00       | 0.0000       | 0.0000       | 0.0000  
0.50       | 1.0000       | 1.0000       | 0.0000  
1.00       | 2.0000       | 2.0000       | 0.0000  
1.50       | 3.0000       | 3.0000       | 0.0000  
2.00       | 4.0000       | 4.0000       | 0.0000  
-----  
  
Оберіть диференціальне рівняння для розв'язання:  
1.  $y' = 2$   
2.  $y' = x$   
3.  $y' = y$   
4.  $y' = x + y$   
q. Вихід  
Ваш вибір > вихід  
Завершення роботи.
```

Рис. 7.36. Рівняння з постійною похідною ($y' = 2$)

Тест підтверджує, що програма реалізує базовий крок. Оскільки стовпці «*Euler y*» та «Точний *y*» повністю збігаються, а стовпець «Похибка» містить нулі, бачимо доказ того, що для лінійних функцій метод Ейлера є абсолютно точним.

– перевірка, як програма наочно показує накопичення похибки для нелінійної функції (рис. 7.37).

```

Програма для розв'язання диференціальних рівнянь методом Ейлера.
Для виходу з програми введіть 'вихід' або 'q'.

Оберіть диференціальне рівняння для розв'язання:
1.  $y' = 2$ 
2.  $y' = x$ 
3.  $y' = y$ 
4.  $y' = x + y$ 
q. Вихід
Ваш вибір > 2
Введіть початкове  $x_0$ : 0
Введіть початкове  $y_0$ : 0
Введіть розмір кроку  $h$ : 0.2
Введіть кількість кроків  $N$ : 5

--- Результати обчислення ---
x          | Euler y      | Точний y     | Похибка
-----
0.00       | 0.00000      | 0.00000      | 0.00000
0.20       | 0.00000      | 0.02000      | 0.02000
0.40       | 0.04000      | 0.08000      | 0.04000
0.60       | 0.12000      | 0.18000      | 0.06000
0.80       | 0.24000      | 0.32000      | 0.08000
1.00       | 0.40000      | 0.50000      | 0.10000
-----

Оберіть диференціальне рівняння для розв'язання:
1.  $y' = 2$ 
2.  $y' = x$ 
3.  $y' = y$ 
4.  $y' = x + y$ 
q. Вихід
Ваш вибір > вихід
Завершення роботи.

```

Рис. 7.37. Просте нелінійне рівняння ($y' = x$)

Програма демонструє головну властивість методу: стовпець «Похибка» чітко показує, як з кожним кроком наближений розв'язок все більше відхиляється від точного, що є очікуваною поведінкою.

– аналіз точності методу на функції, що швидко зростає (рис. 7.38).

Програма для розв'язання диференціальних рівнянь методом Ейлера.
Для виходу з програми введіть 'вихід' або 'q'.

Оберіть диференціальне рівняння для розв'язання:

1. $y' = 2$
2. $y' = x$
3. $y' = y$
4. $y' = x + y$
- q. Вихід

Ваш вибір > 3

Введіть початкове x_0 : 0

Введіть початкове y_0 : 1

Введіть розмір кроку h : 0.1

Введіть кількість кроків N : 10

--- Результати обчислення ---

x	Euler y	Точний y	Похибка
0.00	1.0000	1.0000	0.0000
0.10	1.1000	1.1052	0.0052
0.20	1.2100	1.2214	0.0114
0.30	1.3310	1.3499	0.0189
0.40	1.4641	1.4918	0.0277
0.50	1.6105	1.6487	0.0382
0.60	1.7716	1.8221	0.0506
0.70	1.9487	2.0138	0.0650
0.80	2.1436	2.2255	0.0820
0.90	2.3579	2.4596	0.1017
1.00	2.5937	2.7183	0.1245

Оберіть диференціальне рівняння для розв'язання:

1. $y' = 2$
2. $y' = x$
3. $y' = y$
4. $y' = x + y$
- q. Вихід

Ваш вибір > вихід

Завершення роботи.

Рис. 7.38. Експоненційне зростання ($y' = y$)

Програма дозволяє кількісно оцінити неточність методу: для експоненційної функції похибка стає досить суттєвою, що підкреслює обмеження методу Ейлера.

– використання програми як інструменту, щоб довести, що менший крок h дає меншу похибку (рис. 7.39–7.40).

⇒ Програма для розв'язання диференціальних рівнянь методом Ейлера.
Для виходу з програми введіть 'вихід'.

Оберіть диференціальне рівняння для розв'язання:

1. $y' = 2$
2. $y' = x$
3. $y' = y$
4. $y' = x + y$
- q. Вихід

Ваш вибір > 3

Введіть початкове x_0 : 0

Введіть початкове y_0 : 1

Введіть розмір кроку h : 0.01

Введіть кількість кроків N : 100

--- Результати обчислення ---

x	Euler y	Точний y	Похибка
0.00	1.0000	1.0000	0.0000
0.01	1.0100	1.0101	0.0001
0.02	1.0201	1.0202	0.0001
0.03	1.0303	1.0305	0.0002

....

0.99	2.6780	2.6912	0.0132
1.00	2.7048	2.7183	0.0135

Оберіть диференціальне рівняння для розв'язання:

1. $y' = 2$
2. $y' = x$
3. $y' = y$
4. $y' = x + y$
- q. Вихід

Ваш вибір > вихід

Завершення роботи.

Рис. 7.39. Вплив розміру кроку ($h=0.01$)

На наведеному прикладі продемонстровано використання програми як інструменту для чисельного аналізу диференціального рівняння $y' = y$. Обчислення з кроком $h = 0.01$ дозволяють отримати високу точність: на кінцевому етапі ($x = 1.00$) різниця між наближеним значенням за методом Ейлера (2.7048) та точним розв'язком (2.7183) становить лише 0.0135. Це підтверджує практичне правило: вибір малого кроку дискретизації мінімізує накопичену похибку та забезпечує збіжність чисельного методу з теоретичною моделлю.



Програма для розв'язання диференціальних рівнянь методом Ейлера.
Для виходу з програми введіть 'вихід'.

Оберіть диференціальне рівняння для розв'язання:

1. $y' = 2$
2. $y' = x$
3. $y' = y$
4. $y' = x + y$
- q. Вихід

Ваш вибір > 3

Введіть початкове x_0 : 0

Введіть початкове y_0 : 1

Введіть розмір кроку h : 0.2

Введіть кількість кроків N : 5

--- Результати обчислення ---

x	Euler y	Точний y	Похибка
0.00	1.0000	1.0000	0.0000
0.20	1.2000	1.2214	0.0214
0.40	1.4400	1.4918	0.0518
0.60	1.7280	1.8221	0.0941
0.80	2.0736	2.2255	0.1519
1.00	2.4883	2.7183	0.2300

Оберіть диференціальне рівняння для розв'язання:

1. $y' = 2$
2. $y' = x$
3. $y' = y$
4. $y' = x + y$
- q. Вихід

Ваш вибір > вихід

Завершення роботи.

Рис. 7.40. Вплив розміру кроку ($h=0.2$)

Порівняльний аналіз результатів, поданих на рис. 7.39 та рис. 7.40, наочно демонструє залежність точності обчислень від величини кроку h . У разі великого кроку ($h = 0.2$) накопичена похибка на кінці інтервалу становить 0.2300, тоді як у разі його зменшення до $h = 0.01$ вона скорочується до 0.0135. Цей тест доводить, що зменшення кроку значно підвищує точність наближеного розв'язку. Таким чином, проведений порівняльний аналіз не лише підтверджує теоретичні положення щодо порядку точності методу Ейлера, а й слугує практичним обґрунтуванням для вибору оптимальних параметрів моделювання під час розв'язання прикладних задач динаміки.

– перевірка, як програма обробляє граничний випадок з нульовою кількістю кроків (рис. 7.41).

```
➡ Програма для розв'язання диференціальних рівнянь методом Ейлера.  
Для виходу з програми введіть 'вихід'.  
  
Оберіть диференціальне рівняння для розв'язання:  
1.  $y' = 2$   
2.  $y' = x$   
3.  $y' = y$   
4.  $y' = x + y$   
q. Вихід  
Ваш вибір > 1  
Введіть початкове  $x_0$ : 10  
Введіть початкове  $y_0$ : 20  
Введіть розмір кроку  $h$ : 0.1  
Введіть кількість кроків  $N$ : 0  
  
--- Результати обчислення ---  
x          | Euler y          | Точний y          | Похибка  
-----  
10.00      | 20.00000         | 20.00000         | 0.00000  
-----  
  
Оберіть диференціальне рівняння для розв'язання:  
1.  $y' = 2$   
2.  $y' = x$   
3.  $y' = y$   
4.  $y' = x + y$   
q. Вихід  
Ваш вибір > вихід  
Завершення роботи.
```

Рис. 7.41. Нуль кроків ($N=0$)

Програма коректно обробляє цей випадок, не виконуючи жодної ітерації циклу і повертаючи лише початкову точку, для якої похибка, звісно, дорівнює нулю. Тестування підтверджує, що програма є коректною реалізацією методу Ейлера. Вона дозволяє не тільки обчислювати наближені розв'язки, але й миттєво аналізувати їхню точність, досліджувати накопичення похибки та наочно вивчати ключові властивості чисельних методів, такі як залежність точності від розміру кроку.

Крім того, успішна верифікація алгоритму на різних типах диференціальних рівнянь свідчить про надійність програмного забезпечення як навчального та дослідницького інструмента для вивчення обчислювальної математики. Це підтверджує ефективність переходу від неперервних моделей до дискретних обчислювальних структур із контрольованим рівнем похибки. Такий підхід демонструє фундаментальний зв'язок між аналітичними функціями та їх дискретними апроксимаціями, що є базовим принципом для побудови надійних обчислювальних моделей.

7.7.2. Практичні завдання для самостійної роботи

Наведені нижче завдання розроблено для практичного засвоєння чисельних методів розв'язання звичайних диференціальних рівнянь. Виконання цих вправ дозволить не лише опанувати алгоритм методу Ейлера, а й наочно дослідити природу похибок апроксимації та їхню залежність від величини кроку інтегрування.

Завдання 1. Ручний розрахунок (Метод Ейлера). Дано диференціальне рівняння:

$$y' = y - x^2, \quad y(0) = 1.$$

Необхідно знайти наближений розв'язок на відрізку $[0; 1]$.

- Виконайте розрахунок вручну з кроком $h = 0.2$ (всього 5 кроків).
- Результати занесіть у таблицю вигляду:

i	x_i	y_i
0	0.0	1.000
1	0.2	...
...	...	...

Завдання 2. Оцінка похибки. Для рівняння із **Завдання 1** відомий точний аналітичний розв'язок:

$$y_{exact}(x) = x^2 + 2x + 2 - e^x.$$

- Обчисліть точні значення y_{exact} у вузлових точках ($x = 0.2, 0.4, \dots, 1.0$).
- Обчисліть абсолютну похибку $\Delta_i = |y_i^{Euler} - y_i^{exact}|$ для кожного кроку.
- Зробіть висновок: як змінюється похибка з віддаленням від початкової точки?

Завдання 3. Програмне моделювання. Використайте програмний код, наведений у розділі, та:

- реалізуйте функцію $f(x, y) = y - x^2$ та точний розв'язок;
- виконайте розрахунок для двох різних кроків: $h_1 = 0.1$ та $h_2 = 0.05$;
- порівняйте похибку в кінцевій точці $x = 1$ для обох випадків. Чи підтверджується теорія про те, що зменшення кроку у два рази зменшує похибку приблизно у два рази (для методу першого порядку)?

Завдання 4. Візуалізація. Побудуйте на одній координатній площині (у зошиті або використовуючи бібліотеку *matplotlib*):

- графік точного розв'язку (суцільна лінія);
 - точки наближеного розв'язку, отримані в завданні 1 (маркери).
- Проаналізуйте, у який бік (вгору чи вниз) відхиляється наближений графік від точного, і спробуйте пояснити це, дивлячись на знак другої похідної (y'').

7.8. Рекурентні рівняння. Як «точні» розрахунки стають помилкою?

7.8.1. Постановка завдання

Розглянемо послідовність:

$$I_n = \frac{1}{e} \int_0^1 x^n e^x dx.$$

Інтеграл існує для всіх $n \geq 0$, а сама величина I_n є додатною та монотонно спадною. Змістовно: інтеграл – це площа під кривою $x^n e^x$ (рис. 7.42). Легко побачити, що при $n \rightarrow \infty I_n \rightarrow 0$.

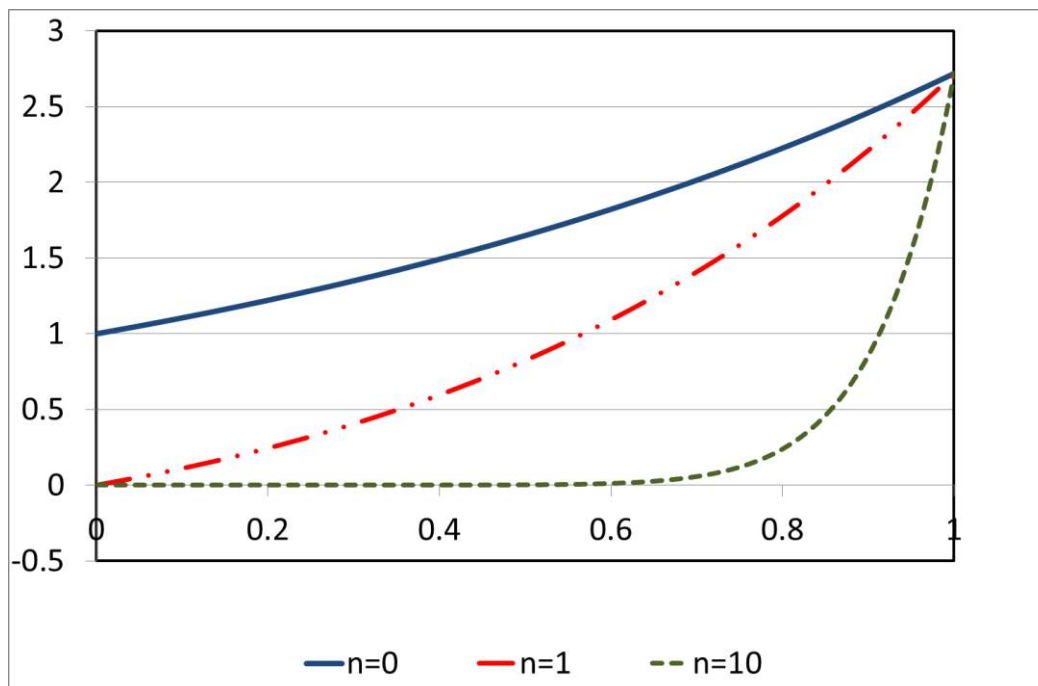


Рис 7.42. Графік залежності $x^n e^x$ від x при різних n

Інтегруючи частинами, отримуємо рекурентну формулу:

$$I_n = 1 - nI_{n-1}, \quad I_0 = 1 - \frac{1}{e}.$$

Таким чином, задачу обчислення інтегралу можна звести до обчислення рекурентної послідовності.

На перший погляд, ситуація ідеальна:

- формулу отримано аналітично;
- початкове значення відомо точно;
- кожен наступний член визначається простою формулою.

Однак чисельний експеримент показує парадоксальний результат: уже приблизно при $n \approx 20$ обчислення у стандартному 64-бітному типі float призводить до різкої осциляції значень і подальшого переповнення числа float.

Постає питання: як може аналітично точна формула давати катастрофічно неправильний результат?

7.8.2. Програмна реалізація на Python для чисельного інтегрування

Спочатку обчислимо інтеграл без використання рекурентної формули – через чисельне інтегрування:

```
import math
def numerical_integral(n, steps=10000):
    a, b = 0.0, 1.0
    h = (b - a) / steps
    s = 0.0
    for i in range(steps):
        x = a + i*h
        s += (x**n) * math.exp(x)
    return (h * s) / math.e
for n in range(0, 10):
    print(n, numerical_integral(n))
```

Цей підхід є обчислювально затратним, але дає стабільний результат (рис. 7.43): похибка потроху накопичується, але візуально все відповідає очікуванням.

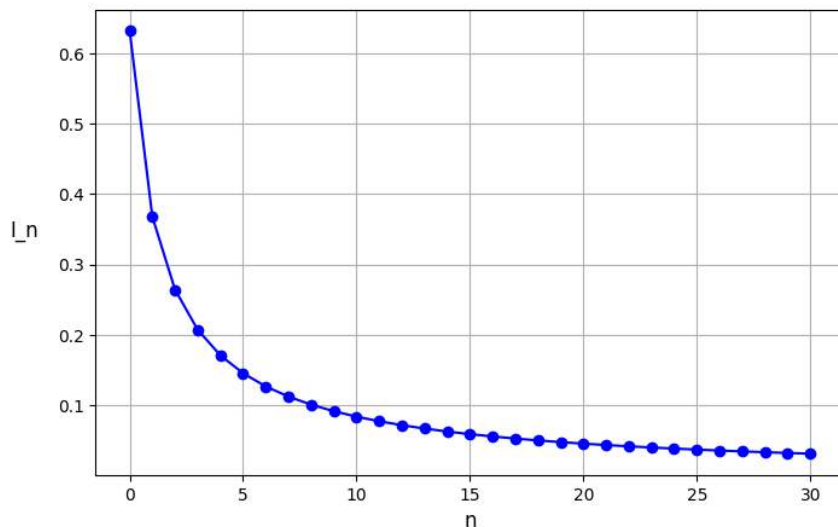


Рис. 7.43. Графік залежності I_n від n , обчисленого через чисельне інтегрування (монотонне спадання до нуля)

7.8.3. Метод «точного» розрахунку на основі аналітичного рішення

Маємо точну рекурентну формулу:

$$I_n = 1 - nI_{n-1}.$$

Із математичного погляду це абсолютно коректне співвідношення.

Але звернімо увагу на структуру формули:

- істинне значення I_n мале;
- воно обчислюється як різниця двох близьких чисел: $1 - nI_{n-1}$.

При великих n :

$$nI_{n-1} \approx 1.$$

Тобто ми віднімаємо майже рівні числа.

Це класичний механізм **катастрофічного віднімання**. Навіть якщо похибка на попередньому кроці була дуже малою, вона:

- множиться на n ;
- проходить через віднімання;
- передається наступному кроку;
- знову множиться вже на $n + 1$.

У результаті похибка зростає приблизно факторіально.

7.8.4. Програмна реалізація «точного» розрахунку

Програма для розрахунку рекурентної формули в прямому напрямку така:

```
import math
def forward_recursion(N):
    I = 1 - 1/math.e
    values = [I]
    for n in range(1, N+1):
        I = 1 - n*I
        values.append(I)
    return values
vals = forward_recursion(30)
for n, v in enumerate(vals):
    print(n, v)
```

Якщо додатково зробити чисельний експеримент із точністю розрахунку $I = 1 - 1/\text{math.e}$ шляхом застосування округлення до результату до певної кількості десяткових цифр у числі, то результат буде такий (рис. 7.44):

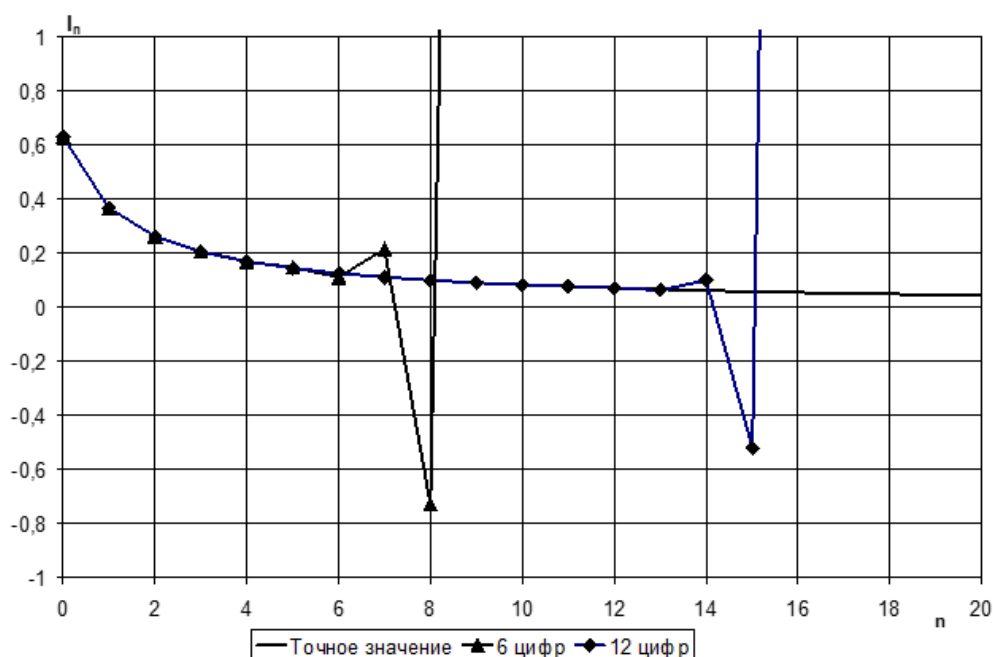


Рис. 7.44. Графіки залежності I_n від n та кількості десяткових цифр у розрахунку

- спочатку значення поведуться коректно;
- після приблизно $n \approx 7-15$ виникає осциляція, значення n залежить від кількості десяткових цифр, які ми залишили при розрахунку $I = 1 - 1/\text{math.e}$;

- далі амплітуда різко зростає;
- врешті-решт виникає переповнення (inf) числа float.

Але можна виконувати рекурентний розрахунок у зворотному напрямі. При великому значенні n можна прийняти $I_n = 0$, тоді:

$$I_{n-1} = \frac{1 - I_n}{n}.$$

Програма для розрахунку рекурентної формули у зворотному напрямі така:

```
def backward_recursion(N):
    I = 0.0
    values = [0]*(N+1)
    values[N] = I
    for n in range(N, 0, -1):
        I = (1 - I)/n
        values[n-1] = I
    return values
vals_back = backward_recursion(30)
for n, v in enumerate(vals_back):
    print(n, v)
```

Виявляється, що розрахунок у зворотному напрямі дає стабільний результат, що узгоджується з результатами чисельного інтегрування.

7.8.5. Обговорення результатів

Маємо парадокс:

- аналітична формула – точна;
- математичні перетворення – бездоганні;
- алгоритм – нестійкий.

Причина полягає у властивостях рекурентного співвідношення.

Рекурентне співвідношення

$$I_n = 1 - nI_{n-1}$$

є нестійким у прямому напрямі, але стійким у зворотному.

Це означає, що мала похибка:

- у прямому напрямі зростає приблизно як $n!$;
- у зворотному напрямі, навпаки, згасає.

При $n \approx 20$ для 64-бітного числа (float) накопичена похибка досягає рівня машинної точності, після чого:

- виникає сильна осциляція;
- значення швидко зростають за модулем;
- відбувається переповнення.

Таким чином, проблема виникає не через математику, а через неправильний вибір алгоритму.

Це класичний приклад того, як «точний» розрахунок стає джерелом грубої помилки та демонструє фундаментальний принцип чисельної математики:

- аналітична правильність формули не гарантує чисельної стабільності алгоритму;
- розуміння властивостей рекурентних співвідношень є необхідною умовою коректного комп'ютерного моделювання.

Аналіз механізму накопичення похибки

Для остаточного розуміння того, чому «точна» формула підвела нас у прямому напрямі, розглянемо динаміку похибки ϵ :

- Пряма рекурсія (кожен крок множить попередню похибку на n). Наприклад, при $n = 20$ похибка, що виникла через машинне округлення (10^{-16}), буде помножена на $20! \approx 2.4 \cdot 10^{18}$. Це миттєво знищує корисний сигнал, перетворюючи результат на «білий шум».
- Зворотна рекурсія (тут працює механізм самокорекції. Будь-яка похибка на кроці n при переході до $n - 1$ ділиться на n . Це означає, що навіть якщо ми почнемо з грубого припущення (наприклад, $I_{50} = 0$), уже через 10–15 кроків алгоритм «забуде» цю помилку і вийде на істинну траєкторію.

7.8.6. Практичні завдання для самостійної роботи

Завдання 1. Дослідження моменту втрати точності.

Експериментально визначте, при якому n у прямій рекурсії відбувається різке зростання похибки. Порівняйте цей результат із теоретичною межею точності типу *float64* ($\approx 10^{-16}$).

Завдання 2. Вплив типу даних.

Повторіть експеримент, використовуючи модуль *decimal* з різними рівнями точності (наприклад, 30 та 60 знаків). Побудуйте таблицю залежності «критичного n » від обраної точності.

Завдання 3. Аналіз рекурентного співвідношення.

Розгляньте рекурентне співвідношення $x_{n+1} = a \cdot x_n \cdot (1 - x_n)$. Дослідіть, чи є воно стійким при різних значеннях параметра a . При якому a поведінка системи стає хаотичною?

Завдання 4. Дослідження самокорекції. Використовуючи програму для зворотної рекурсії, введіть завідомо хибне початкове значення (наприклад, $I_{50} = 100$). Простежте, на якому кроці значення I_n «стабілізується» і почне збігатися з результатами чисельного інтегрування. Поясніть цей ефект.

ПІСЛЯМОВА

Дискретна математика є фундаментальною складовою сучасних комп'ютерних наук і відіграє ключову роль у формуванні теоретичних та практичних основ програмування, алгоритмізації й побудови інформаційних систем. Запропонований у цьому посібнику матеріал спрямований на цілісне осмислення базових понять і методів комп'ютерної дискретної математики та їх безпосереднє застосування у практичній діяльності фахівців комп'ютерного профілю.

У посібнику систематично викладено основні розділи дисципліни – теорії множин, відношень і функцій, алгебру висловлювань, булеві функції, числення висловлювань, логіку предикатів і комбінаторики. Послідовна структура матеріалу дає змогу поступово переходити від базових абстрактних понять до більш складних логічних і комбінаторних моделей, що широко застосовуються в аналізі алгоритмів, проектуванні програмного забезпечення та дослідженні обчислювальних процесів.

Особливістю посібника є поєднання теоретичних положень із практичними прикладами, реалізованими мовою Python. Такий підхід сприяє кращому розумінню абстрактних математичних конструкцій, розвитку алгоритмічного мислення та формуванню навичок програмної реалізації математичних моделей. Практичні завдання та приклади програм дозволяють читачеві самостійно перевіряти правильність розв'язань і експериментувати з різними підходами до розв'язання задач.

Опрацювання матеріалу посібника забезпечує формування необхідних компетентностей для подальшого вивчення спеціальних дисциплін комп'ютерного циклу, а також створює міцне підґрунтя для професійної діяльності у сфері програмної інженерії, комп'ютерних наук і кібербезпеки. Здобуті знання й уміння можуть бути використані як у навчальному процесі, так і у практичних проєктах, пов'язаних із розробленням алгоритмів, аналізом даних і побудовою логічних моделей.

Таким чином, запропонований посібник орієнтований на формування цілісного бачення ролі дискретної математики в сучасних інформаційних технологіях і покликаний сприяти підготовці компетентних фахівців, здатних ефективно застосовувати математичний апарат у розв'язанні актуальних задач комп'ютерних наук.

ЗМІСТ

ВСТУП	3
1. ОСНОВИ ТЕОРІЇ МНОЖИН. ВІДНОШЕННЯ І ФУНКЦІЇ	5
1.1. Поняття множини. Способи задання множин	5
1.2. Операції над множинами	7
1.3. Алгебра множин.....	9
1.4. Відношення	12
1.5. Функціональні відношення	14
1.6. Вправи до розділу 1.....	15
1.7. Програмна реалізація вправ	23
1.7.1. Завдання для програмної реалізації.....	23
1.7.2. Основи Python для операцій з множинами.....	23
1.7.3. Виконання завдань для програмної реалізації.....	29
2. АЛГЕБРА ВИСЛОВЛЮВАНЬ.....	38
2.1. Висловлювання й алгебра висловлювань. Тавтології і протиріччя	38
2.2. Рівносильні пропозиційні формули	45
2.3. Логічний наслідок.....	46
2.4. Розкладання ПФ і доскональні нормальні форми	47
2.5. Диз'юнктивна і кон'юнктивна нормальні форми	51
2.6. Закон подвійності.....	53
2.7. Повнота мов (системи зв'язувань)	54
2.8. Вправи до розділу.....	55
2.9. Програмна реалізація вправ	64
3. БУЛЕВІ ФУНКЦІЇ ТА ЇХ ВИКОРИСТАННЯ.....	84
3.1. Класи булевих функцій. Повні системи. Теорема Яблонського	84
3.2. Подання БФ у вигляді полінома Жегалкіна.....	88
3.3. Застосування булевих функцій.....	88
3.4. Задача мінімізації БФ у класі ДНФ і алгоритм її розв'язання.....	90
3.5. Формулювання задачі мінімізації у класі ДНФ	93
3.6. Методи мінімізації у класі ДНФ. Метод Квайна – Мак-Класкі.....	95
3.7. Вправи до розділу.....	98
3.8. Програмна реалізація вправ	102
4. ЧИСЛЕННЯ ВИСЛОВЛЮВАНЬ (ТЕОРІЯ L)	111
4.1. Виведення у теорії L	111
4.2. Приклади доведення теорем в теорії L	113
4.3. Вправи до розділу.....	116

5. ЛОГІКА І ЧИСЛЕННЯ ПРЕДИКАТІВ.....	117
5.1. Формули логіки предикатів і їх перетворення.....	117
5.2. Оператори кватифікації та їхні властивості	118
5.3. Перетворення формул логіки предикатів	119
5.4. Основні визначення і приклади предикатів.....	120
5.5. Основні властивості операцій квантифікації.....	125
5.6. Тотожно-істинні предикатні формули.....	126
5.7. Числення предикатів	127
5.8. Вправи до розділу.....	127
5.9. Приклад програмної реалізації вправ	135
6. ОСНОВИ КОМБІНАТОРИКИ	146
6.1. Основні поняття і визначення	146
6.2. Формули для підрахунку числа розміщень і сполучень	147
6.3. Біноміальні коефіцієнти.....	149
6.4. Формула включення-виключення	151
6.5. Вправи до розділу.....	152
6.6. Використання основ мови Python у комбінаториці	162
7. ДОДАТКОВІ ПОЛОЖЕННЯ	167
7.1. Метод зваженої суми для розв'язування багатокритеріальних задач	167
7.1.1. Реалізація алгоритму мовою Python.....	168
7.1.2. Практичні завдання для самостійної роботи.....	171
7.2. Зворотний польський запис	172
7.2.1. Програмна реалізація на Python	174
7.2.2. Практичні завдання для самостійної роботи.....	179
7.3. Математична індукція.....	181
7.3.1. Практичні завдання для самостійної роботи.....	183
7.3.2. Програмна реалізація на Python	183
7.4. Відстань Хеммінга	187
7.4.1. Програмна реалізація на Python	188
7.4.2. Практичні завдання для самостійної роботи.....	191
7.5. Відстань Левенштейна	192
7.5.1. Програмна реалізація на Python	194
7.5.2. Практичні завдання для самостійної роботи.....	198
7.6. Методи контрольного сумування.....	199
7.6.1. Програмна реалізація на Python	202
7.6.2. Практичні завдання для самостійної роботи.....	206

7.7. Рекурентні рівняння. Метод Ейлера.....	208
7.7.1. Програмна реалізація на Python	211
7.7.2. Практичні завдання для самостійної роботи.....	219
7.8. Рекурентні рівняння. Як «точні» розрахунки стають помилкою?	220
7.8.1. Постановка завдання	220
7.8.2. Програмна реалізація на Python для чисельного інтегрування	221
7.8.3. Метод «точного» розрахунку на основі аналітичного рішення	221
7.8.4. Програмна реалізація «точного» розрахунку	222
7.8.5. Обговорення результатів.....	223
7.8.6. Практичні завдання для самостійної роботи.....	224
ПІСЛЯМОВА.....	225
БІБЛІОГРАФІЧНИЙ СПИСОК.....	229

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Андрійчук В. І., Комарницький М. Я., Іщук Ю. Б. Вступ до дискретної математики. Львів : Видавничий центр ЛНУ ім. Івана Франка, 2003. 254 с.
2. Бардачов Ю. М., Ходаков В. Є., Щербина В. В. Дискретна математика. Київ : Вища школа, 2007. 383 с.
3. Бондаренко М. Ф., Білоус Н. В., Руткас О. Г. Комп'ютерна дискретна математика : підручник. Харків : Компанія СМІТ, 2004. 480 с.
4. Глибовець М. М. Основи комп'ютерних алгоритмів. Київ : Видавничий дім «Києво-Могилянська академія», 2003. 452 с.
5. Голощук Р. О., Козачко А. М. Дискретна математика : навч. посіб. Рівне : НУВГП, 2022. 245 с.
6. Дроздик М. І., Сердюк П. В. Практикум з комп'ютерної дискретної математики. Львів : Видавництво Львівської політехніки, 2020. 316 с.
7. Журавчак Л. М. Дискретна математика для програмістів : навч. посіб. Львів : Видавництво Львівської політехніки, 2019. 420 с.
8. Капітонова Ю. В., Кривий С. Л., Летичевський О. А. Основи дискретної математики. Київ : Наукова думка, 2002. 578 с.
9. Кривий С. Л. Дискретна математика. Вибрані питання. Київ : Видавничий дім «Києво-Могилянська академія», 2007. 572 с.
10. Кузнецов О. В. Дискретна математика : теорія та алгоритми. Харків : ХНУРЕ, 2021. 280 с.
11. Манзій О. С., Тесак І. Є., Кавалець І. І. Дискретна математика : практикум. Львів : Видавництво Львівської політехніки, 2016. 324 с.
12. Матвієнко М. П. Дискретна математика : навч. посіб. 2-ге вид., переробл. та доповн. Київ : Ліра-К, 2023. 324 с.
13. Нікольський Ю. В., Пасічник В. В., Щербина Ю. М. Дискретна математика : підручник. Київ : Магнолія 2006, 2024. 432 с.
14. Трохимчук П. П. Дискретна математика : навч. посіб. Луцьк : Вежа-Друк, 2021. 240 с.
15. Bondy J. A., Murty U. S. R. Graph theory. London : Springer, 2008. 651 p.
16. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to algorithms. 4th ed. Cambridge : MIT Press, 2022. 1312 p.
17. Diestel R. Graph theory. 5th ed. Berlin : Springer, 2017. 447 p.
18. Epp S. S. Discrete mathematics with applications. 5th ed. Boston : Cengage Learning, 2020. 1056 p.
19. Graham R. L., Knuth D. E., Patashnik O. Concrete mathematics: a foundation for computer science. 2nd ed. Reading : Addison-Wesley, 1994. 672 p.
20. Knuth D. E. The art of computer programming. Vol. 1: Fundamental algorithms. 3rd ed. Reading : Addison-Wesley, 1997. 672 p.
21. Rosen K. H. Discrete mathematics and its applications. 8th ed. New York : McGraw-Hill, 2019. 1120 p.

Навчальне видання

**І. Б. Туркін, А. Г. Чухрай,
О. В. Лучшева, Т. Г. Дегтярьова**

ДИСКРЕТНА МАТЕМАТИКА З ПРИКЛАДАМИ НА PYTHON

Навчальний посібник

Редактор: *Дмитренко Євгенія Валеріївна*, канд. філол.
наук, доцент, доцент кафедри української мови та
мовної комунікації Національного аерокосмічного
університету «Харківський авіаційний інститут»

Керівник видавничого проекту *Віталій Зарицький*

Підписано до друку 17.03.2026. Формат 70×100 1/16.
Папір офсетний. Друк офсетний.
Умовн. друк. аркушів – 18,85. Обл.-вид. аркушів – 15,84.

Видавець і виготовлювач: ТОВ «Видавництво Ліра-К»
Свідоцтво № 3981, серія ДК.
03115, м. Київ, вул. С. Чобану, 24
тел.: (050) 462-95-48; (067) 820-84-77
Сайт: lira-k.com.ua, редакція: zv_lira@ukr.net