Лекція 5

Тема 2. Множини, функції, відношення

План лекції

- **Поняття множини і кортежу. Декартів добуток**
- **Булева алгебра множин**
- **Розбиття множини**
- **Доведення рівностей з множинами**
- **Доведення рівностей з множинами**
- **Функції**
- **Зростання функції. Оцінки складності алгоритмів: *O*-велике нотація**

Об'єкти, які утворюють *множину*, називають її *елементами*. Про множину говорять, що вона *містить* ці елементи. Якщо об'єкт a є елементом множини A, то пишуть $a \in A$; а ні, то $a \notin A$. Синоніми: *сукупність, система, набір*.

Для часто використовуваних множин є спеціальні позначення:

$\varnothing$ – *порожня* множина, яка не містить жодного елемента;

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

R – множина дійсних чисел;

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

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

Задати множину можна, зазначивши спільну властивість усіх її елементів. Тоді множину A задають за допомогою позначення $A=\{x \mid P(x)\}$, яке читають так: «A – це множина об'єктів x, які мають властивість $P(x)$». Наприклад, $A=\{x \mid x\in N_0, x<7\}$ – це множина $\{0, 1, 2, 3, 4, 5, 6\}$.

Дві множини A та B називають *рівними*, якщо вони складаються з одних і тих самих елементів. Рівність множин A та B записують як $A=B$.

Множину A називають *підмножиною* множини B, якщо кожний елемент множини A належить множині B. У такому разі пишуть $A\subset B$, причому це не виключає, що $A=B$. Якщо $A=B$ або $A=\varnothing$, то A називають *невласною* підмножиною множини B. Якщо $A\neq B$ і $A\neq\varnothing$, то A називають *власною* підмножиною множини B. Для будь-якої множини A правдиве включення $\varnothing\subset A$.

Зазначимо, що в літературі іноді використовують позначення $A\subseteq B$; тоді позначення $A\subset B$ резервують для випадку, коли $A\subseteq B$ і $A\neq B$.

Часто всі розглядувані в певній ситуації множини являють собою підмножини якоїсь множини, яку називають *універсальною множиною* або *універсумом*. Універсальну множину позначають як U.

Для заданої множини A можна розглянути множину всіх її підмножин, включно з порожньою множиною $\varnothing$ і самою множиною A. Цю множину позначають 2^A чи $P(A)$ й називають *множиною-степенем*, або *булеаном* множини A. Для скінченної множини A множина 2^A містить $2^{|A|}$ елементів.

Приклад. Нехай $A=\{0, 1, 2\}$. Тоді $2^A=\{\varnothing, \{0\}, \{1\}, \{2\}, \{0,1\}, \{0,2\} \{1,2\}, \{0,1,2\}\}$. Ця множина містить $2^3 = 8$ елементів.

Кортеж – це впорядкований набір елементів. Сказане не слід розглядати як означення кортежу, оскільки тоді потрібно дати пояснення з приводу його синоніма «впорядкований набір». Поняття «кортеж» (синоніми – *вектор*, *рядок*, *ланцюжок*, *слово*) уважатимемо, як і поняття множини, первісним, тобто неозначуваним. Елементи, що утворюють кортеж, називають його *компонентами*. Компоненти нумерують, кількість компонент називають *довжиною* або *розмірністю* кортежу. Нескінченні кортежі не розглядатимемо.

На відміну від елементів множини, компоненти кортежу можуть повторюватись. Кортеж записують у круглих дужках, наприклад (a, b, c, a, d) – кортеж довжиною 5. Іноді дужки й навіть коми не пишуть, наприклад кортеж 011001. Кортежі довжиною 2 часто називають *парами*, довжиною 3 – *трійками*. Кортежі довжиною n іноді називають *n-ками* («*енками*»).

Два кортежі рівні, якщо вони мають однакову довжину та відповідні їх компоненти рівні. Іншими словами, кортежі $(a_1, \ldots, a_m)$ *та* $(b_1, \ldots, b_n)$ *рівні, якщо* $m = n$ *та* $a_1=b_1$, $a_2=b_2$, …, $a_m=b_n$.

Декартовим добутком множин A та B (позначають $A\times B$) називають множину всіх пар (a, b) таких, що $a\in A$, $b\in B$. Зокрема, якщо $A=B$, то обидві компоненти належать A. Такий добуток позначають як A^2 та називають *декартовим квадратом* множини A. Аналогічно, декартовим добутком n множин $A_1, \ldots, A_n$ (позначають $A_1\times\ldots\times A_n$) називають множину всіх кортежів $(a_1, \ldots, a_n)$ довжиною n таких, що $a_1\in A_1$, …, $a_n\in A_n$. Частковий випадок $A\times\ldots\times A$ позначають як A^n і називають *n-м степенем* множини A.

Приклад Нехай $A=\{1, 2\}$, $B=\{a, b, c\}$. Тоді $A\times B=\{(1,a), (1,b), (1,c), (2,a), (2,b), (2,c)\}$, $B\times A=\{(a,1), (a,2), (b,1), (b,2), (c,1), (c,2)\}$. Зрозуміло, що загалом $A\times B\neq B\times A$.

Для скінченних множин потужність (кількість елементів) декартового добутку дорівнює добутку потужностей цих множин: $|A\times B|=|A|\cdot|B|$.

Приклад Нехай $A=\{1, 2\}$, $B=\{a, b, c\}$, $C=\{x, y\}$. Тоді
$A\times B\times C=\{(1,a,x), (1,a,y), (1,b,x), (1,b,y), (1,c,x), (1,c,y), (2,a,x), (2,a,y), (2,b,x), (2,b,y), (2,c,x), (2,c,y)\}$.

*Зауваження***.** Якщо A, B і C – множини, то множина $(A\times B)\times C$ – НЕ ТЕ САМЕ, ЩО множина $A\times B\times C$. За означенням елементом множини $A\times B\times C$ є (a, b, c) (тобто трійка), де $a\in A$, $b\in B$, $c\in C$, а елементом множини $(A\times B)\times C$ є $((a, b), c)$. Тобто елементом множини $(A\times B)\times C$ є пара (p, c), де $p\in A\times B$, $p = (a, b)$. Звичайно трійки і пари – це різні креатури, навіть якщо трійки та пари **у цьому випадку** несуть точно одну й ту саму інформацію. Більш точно, існує цілком природне взаємно-однозначне відображення (бієкція) між множинами $A \times B \times C$ та $(A \times B) \times C$, яку задають так: $(a, b, c) \leftrightarrow ((a, b), c)$.

Булева алгебра множин

Будемо вважати, що всі розглядувані множини – підмножинами деякого універсума U. Для довільних множин A та B можна побудувати нові множини за допомогою *теоретико-множинних операцій*:

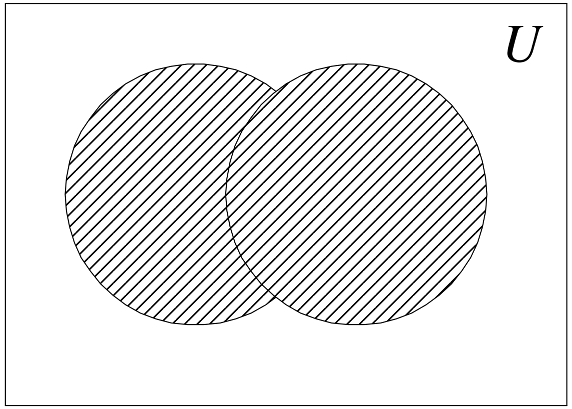
об'єднанням множин A та B називають множину $A \cup B = \{ x \mid (x \in A) \text{ або } (x \in B)\}$;

перетином множин A та B називають множину $A \cap B = \{ x \mid (x \in A) \text{ і } (x \in B)\}$;

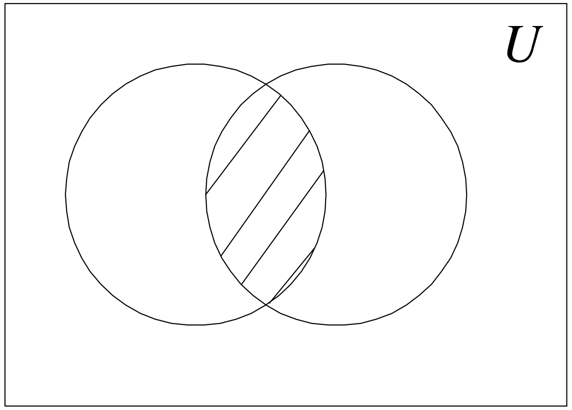
різницею множин A та B називають множину $A \setminus B = \{ x \mid (x \in A) \text{ і } (x \notin B)\}$;

доповненням множини A називають множину $\overline{A} = U \setminus A$, де U – універсальна множина.

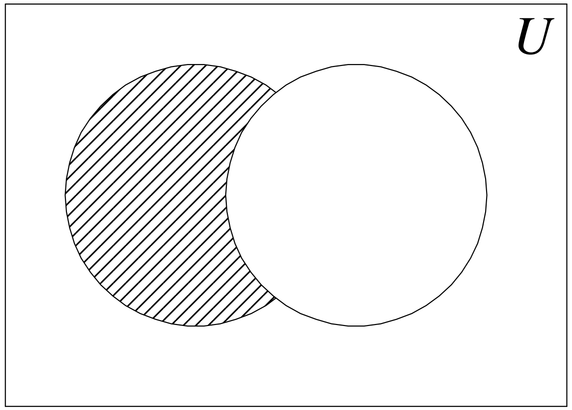
Як можна побачити, всі підмножини універсальної множини утворюють булеву алгебру відносно операцій перетину, об'єднання та доповнення, роль одиниці відіграє універсальна множина, а нуля – порожня. Наведемо основні закони для операцій з множинами.



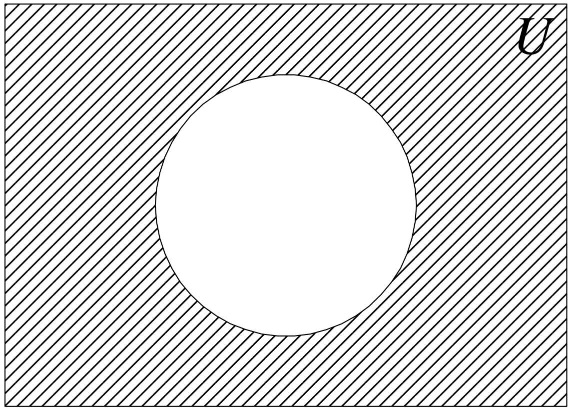
$A \cup B$



$A \cap B$



$A \setminus B$



$\overline{A}$

Основні закони для операцій з множинами

	Назва закону	Формулювання закону
1	Закони комутативності	*а*) $A \cup B = B \cup A$ *б*) $A \cap B = B \cap A$
2	Закони асоціативності	*а*) $A \cup (B \cup C) = (A \cup B) \cup C$ *б*) $A \cap (B \cap C) = (A \cap B) \cap C$
3	Закони дистрибутивності	*а*) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ *б*) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
4	Закон подвійного доповнення	$\overline{\left(\overline{A}\right)} = A$
5	Закони ідемпотентності	*а*) $A \cap A = A$ *б*) $A \cup A = A$
6	Закони де Моргана	*а*) $\overline{A \cup B} = \overline{A} \cap \overline{B}$ *б*) $\overline{A \cap B} = \overline{A} \cup \overline{B}$
7	Закони поглинання	*а*) $A \cap (A \cup B) = A$ *б*) $A \cup (A \cap B) = A$
8	Закони тотожності	*а*) $A \cup \varnothing = A$ *б*) $A \cap U = A$
9	Закони домінування	*а*) $A \cup U = U$ *б*) $A \cap \varnothing = \varnothing$
10	Закони доповнення	*а*) $A \cup \overline{A} = U$ *б*) $A \cap \overline{A} = \varnothing$

Розбиття множини

Систему $S=\{A_i\}$ ($i\in I$, де I – множина індексів) підмножин множини A називають *розбиттям* множини A якщо:

1) $A_i \neq \varnothing$ для всіх $i\in I$;

2) $A_i \cap A_j=\varnothing$, $i\neq j$;

3) $\bigcup\limits_{i\in I} A_i = A$.

Приклад. $A=\{a, b, c\}$. Ось (всі) різні розбиття множини A.

$$S_1=\{\{a\},\{b\},\{c\}\},$$

$$S_2=\{\{a,b\},\{c\}\},$$

$$S_3=\{\{a\},\{b,c\}\},$$

$$S_4=\{\{a,c\},\{b\}\},$$

$$S_5=\{\{a,b,c\}\}.$$

Доведення рівностей з множинами

Спосіб 1. Цей спосіб ґрунтується на такій теоремі.

Теорема. Множини A і B рівні тоді й лише тоді, коли $A \subset B$ та $B \subset A$.

Приклад. Доведемо рівність множин, яка являє собою формулювання закону де Моргана $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Припустимо, що $x \in \overline{A \cap B}$. Тоді $x \notin A \cap B$, звідки випливає, що $x \notin A$ або $x \notin B$. Отже $x \in \overline{A}$ або $x \in \overline{B}$, а це означає, що $x \in \overline{A} \cup \overline{B}$. Ми довели, що $\overline{A \cap B} \subset \overline{A} \cup \overline{B}$. Навпаки, нехай $x \in \overline{A} \cup \overline{B}$. Тоді $x \in \overline{A}$ або $x \in \overline{B}$, звідки випливає, що $x \notin A$ або $x \notin B$. Це означає, що $x \notin A \cap B$, тобто $x \in \overline{A \cap B}$. Отже $\overline{A} \cup \overline{B} \subset \overline{A \cap B}$.

Спосіб 2. Доведення рівності множин за допомогою *таблиць належності*. Ці таблиці містять усі можливі комбінації належності елементів множинам (1 – елемент належить множині, 0 – не належить).

Приклад. Доведемо цим способом рівність $\overline{A \cap B} = \overline{A} \cup \overline{B}$. Доведення подано в таблиці.

A	B	$A \cap B$	$\overline{A \cap B}$	$\overline{A}$	$\overline{B}$	$\overline{A} \cup \overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

Стовпчики, які у таблиці позначено $\overline{A \cap B}$ та $\overline{A} \cup \overline{B}$, однакові, отже $\overline{A \cap B} = \overline{A} \cup \overline{B}$.

Спосіб 3. Доведення рівності множин з використанням основних законів, яким задовольняють теоретико-множинні операції (див таблицю).

Приклад. Довести тотожність $\overline{A \cup (B \cap C)} = (\overline{C} \cup \overline{B}) \cap \overline{A}$. Використовуючи закони де Моргана та комутативності, можна записати таку послідовність рівних множин:

$\overline{A \cup (B \cap C)} = \overline{A} \cap \overline{(B \cap C)} =$ за законом де Моргана 6*а*

$= \overline{A} \cap (\overline{B} \cup \overline{C}) =$ за законом де Моргана 6*б*

$= (\overline{B} \cup \overline{C}) \cap \overline{A} =$ за законом комутативності 1*б*

$= (\overline{C} \cup \overline{B}) \cap \overline{A}$ за законом комутативності 1*а*.

Комп'ютерне подання множин

Комп'ютерні операції над бітами відповідають булевим операціям $\vee$, $\wedge$ та $\oplus$, над бітами. Ми будемо також використовувати нотацію OR, AND і XOR (eXclusive OR), відповідно для операцій $\vee$, $\wedge$ та $\oplus$, як це зроблено в багатьох мовах програмування. Значення операцій OR, AND і XOR над бітами наведено в таблиці

x	y	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Приклад. Знайдемо результати операцій порозрядного OR, порозрядного AND і порозрядного XOR бітових рядків 10 1100 0011 та 11 0101 0101. У результаті одержимо

10 1100 0011
<u>11 0101 0101</u>
11 1101 0111 – порозрядне OR,
10 0100 0001 – порозрядне AND,
01 1001 0110 – порозрядне XOR.

Один із найпоширеніших і найпростіших способів – подання множин за допомогою бітових рядків. Упорядкуємо довільним способом елементи універсальної множини. Нехай універсальна множина U містить n елементів, тоді $U=\{a_1, a_2, a_3, \ldots, a_{n-1}, a_n\}$.

Множину $A\subset U$ подають у комп’ютері рядком із 0 та 1 довжиною n так: якщо $a_i\in A$, то i-й біт дорівнює 1, а ні, то 0.

Приклад. Нехай $U=\{a, b, c, d, e, f, m, n, p, q\}$, $A=\{b, m, n, q\}$, $B=\{a, b, f, m, q\}$. Тоді множину A подають рядком 01 0000 1101, а множину B – рядком 11 0001 1001.

Приклад. Використаємо бітові рядки, які зображають множини A та B з попереднього прикладу. Бітовий рядок, який відповідає об’єднанню цих множин $A \cup B=\{a, b, f, m, n, q\}$ знаходимо як результат виконання операції порозрядного OR:

01 0000 1101
<u>11 0001 1001</u>
11 0001 1101.

Бітовий рядок, який відповідає перетину множин $A \cap B=\{b, m, q\}$ знаходимо як результат виконання операції порозрядного AND:

01 0000 1101
<u>11 0001 1001</u>
01 0000 1001.

Якщо універсальна множина U має велику потужність, а її підмножини не дуже потужні, то подання за допомогою бітових рядків неефективне щодо витрат пам’яті. У такому разі для зображення множин доцільно використовувати інші структури даних.

Функції

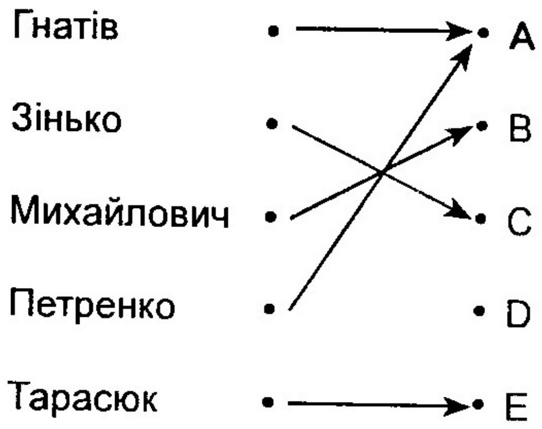


Рис. 1

Наведена відповідність – приклад функції.

Нехай A та B – множини. *Функція* f з A в B – це відповідність кожному елементу множини A певного одного елемента множини B.

Пишуть $f(a)=b$ якщо b є той єдиний елемент з B, який відповідає елементу a із множини A. Якщо f – функція з A в B, то це записують як $f: A \to B$.

Іноді замість функцій говорять про *відображення*, резервуючи термін „функція" для відображень із числовими множинами A та B. Ми не будемо строго притримуватись таких відмінностей, використовуючи терміни „функція" та „відображення" як синоніми.

Функції задають різними способами. Іноді явно формулюють відповідність, як у щойно наведеному прикладі. Дуже часто функцію подають формулою, як, наприклад $f(x)=x+1$. Іншим разом для подання функції використовують комп'ютерну програму.

Якщо f – функція з A в B, то A називають *областю визначення* f. Якщо $f(a)=b$, то b називають *образом* a, у свою чергу a називають *прообразом* b.

Множину $\{b \mid b=f(a), a\in A\}$ образів усіх елементів множини A називають *областю значень* функції f. Очевидно, область значень є підмножиною множини B.

Якщо f – функція з A в B, то говорять, що f *відображає* A в B.

Повернемось до прикладу з оцінюванням студентів групи. Нехай g – функція, яка присвоює кожному студенту групи, яка вивчає дискретну математику, буквену оцінку. Область визначення – множина {Гнатів, Зінько, Михайлович, Петренко, Тарасюк}. Функція g відображає цю множину на множину $\{A, B, C, D, E\}$, а область значень функції g – множина $\{A, B, C, E\}$, бо студенти одержали всі оцінки, окрім D.

Приклад. Нехай f – функція з множини всіх бітових рядків довжиною два або більше в цю ж множину, яка виділяє в рядку останні два біта. Отже, область визначення f – множина всіх бітових рядків довжиною не менше двох, а множина значень – {00, 01, 10, 11}.

Приклад. Нехай f – функція з Z у Z, яка кожному цілому числу ставить у відповідність його квадрат. Отже, $f(x)=x^2$, область визначення – множина всіх цілих чисел, а область значень – множина точних квадратів, тобто {0, 1, 4, 9, 16, 25, 36, …}.

Приклад. Для функції $f: A \to B$ множини A та B часто *визначають* у мовах програмування. Наприклад, на мові Паскаль оператор

function *floor* (x: real): integer

задає множину A (область визначення) функції *floor* як множину дійсних чисел, а множину B – як множину цілих чисел.

Нехай f – функція з A в B і нехай множина $S \subset A$. Образ S – це підмножина множини B, яка складається з образів усіх елементів множини S. Образ множини S позначають як $f(S)$, отже

$$f(S) = \{b \mid b = f(a), a \in S\}.$$

Приклад. Нехай $A = \{a, b, c, d, e\}$, $B = \{1, 2, 3, 4\}$ і нехай $f(a) = 2$, $f(b) = 1$, $f(c) = 4$, $f(d) = 1$ та $f(e) = 1$. Тоді образ множини $S = \{b, c, d\}$ – це множина $f(S) = \{1, 4\}$.

Деякі функції мають різні образи для різних елементів своєї області визначення.

Функцію $f: A \to B$ називають *ін'єктивною* (*англ. one-to-one*), або *ін'єкцією*, якщо вона відображає різні елементи в різні, тобто якщо $f(a_1) \neq f(a_2)$ при $a_1 \neq a_2$. Рис. 2 ілюструє поняття ін'єктивної функції.

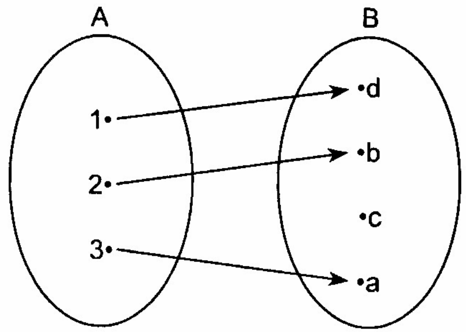


Рис. 2

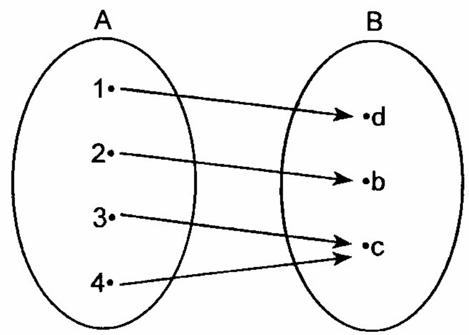


Рис. 3

Для деяких функцій область їхніх значень співпадає із множиною *B*.

Функцію $f: A \to B$ називають *сюр'єктивною* (*англ. onto*), або *сюр'єкцією*, якщо область її значень – уся множина B. Інакше кажучи, функція сюр'єктивна, якщо й тільки якщо для кожного елемента $b \in B$ існує такий елемент $a \in A$, що $f(a) = b$. Іноді сюр'єктивні функції називають відображеннями ***на*** B. Приклад сюр'єктивної функції подано на рис. 3.

Зазначимо, що функція на рис. 2 – не сюр'єктивна, а функція на рис. 3 – не ін'єктивна.

Функцію (відображення) $f: A \to B$, яка водночас є ін'єкцією й сюр'єкцією, називають *бієкцією* (*бієктивним відображенням*), або *взаємно однозначною відповідністю* (*англ. one-to-one correspondence*).

Якщо f – бієкція, то існує *обернена функція* f^{-1}, для якої $f^{-1}(b) = a$ тоді й тільки тоді, коли $f(a) = b$.

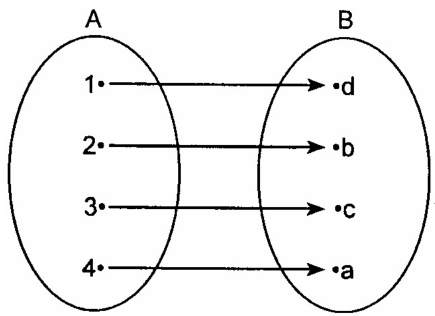


Рис. 4

Приклад. Нехай f – функція з $\{1, 2, 3, 4\}$ на $\{a, b, c, d\}$ така, що $f(1) = d$, $f(2) = b$, $f(3) = c$, $f(4) = a$. Функція f – бієкція, і тому має обернену функцію f^{-1}, причому $f^{-1}(a) = 4$, $f^{-1}(b) = 2$, $f^{-1}(c) = 3$, $f^{-1}(d) = 1$. Рис. 4 ілюструє цей приклад.

Приклад. Нехай f – функція з Z у Z, $f(x) = x^2$. Оскільки $f(-1) = f(1) = 1$, то ця функція **не** ін'єктивна. Якщо б визначити обернену функцію, то елементу 1 потрібно поставити у відповідність два елемента. Отже, оберненої для f функції не існує.

Композицією двох функцій $f: A \to B$ і $g: B \to C$ називають функцію $h: A \to C$, визначену співвідношенням $h(a) = g(f(a))$.

Композицію функцій позначають як $g \circ f$ (ми, як і в більшості книг, пишемо справа функцію, застосовану першою):

$$(g \circ f)(a) = g(f(a)).$$

Зазначимо, композиція $g \circ f$ невизначена, якщо область значень f не є підмножиною області визначення g.

Приклад. Нехай функція f відображає множину $\{a, b, c\}$ в себе й визначена так: $f(a) = b$, $f(b) = c$ і $f(c) = a$. Нехай g – функція з множини $\{a, b, c\}$ в множину $\{1, 2, 3\}$, $g(a) = 3$, $g(b) = 2$, $g(c) = 1$. Тоді композицію $g \circ f$ визначимо як $(g \circ f)(a) = g(f(a)) = g(b) = 2$, $(g \circ f)(b) = g(f(b)) = g(c) = 1$, $(g \circ f)(c) = g(f(c)) = g(a) = 3$.

Нехай $f: A \to B$. Графіком функції f називають множину впорядкованих пар $\{(a, b) \mid a \in A, f(a) = b\}$.

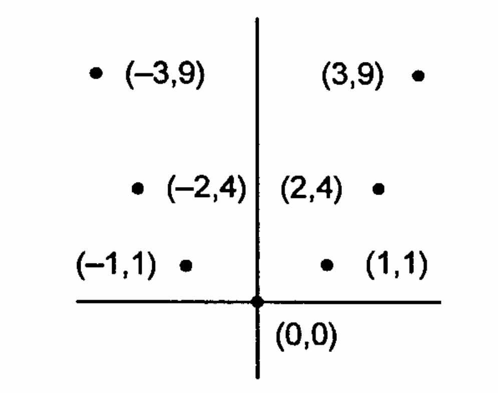


Рис. 5

Приклад. Розглянемо функцію $f(x) = x^2$ із множини Z в множину Z (Z – множина цілих чисел). Її графік зображено на рис. 5; він складається із впорядкованих пар вигляду $(x, f(x)) = (x, x^2)$, де x – ціле число.

Уведемо дві важливі в дискретній математиці функції, які відображають множину дійсних чисел R у множину цілих чисел Z. (Нині в літературі їх часто називають «*підлога*» і «*стеля*».) Перша – це функція $\lfloor x \rfloor$ – найбільше ціле, яке не більше ніж x (ціла частина числа x). Графік цієї функції подано на рис. 6 а. (Підлога.)

Друга функція – її позначають як $\lceil x \rceil$ – найменше ціле, яке не менше ніж x; графік цієї функції подано на рис. 6б. (Стеля.)

Приклад. $\lfloor 0.5 \rfloor = 0$, $\lceil 0.5 \rceil = 1$, $\lfloor -0.5 \rfloor = -1$, $\lceil -0{,}5 \rceil = 0$, $\lfloor 3.1 \rfloor = 3$, $\lceil 3.1 \rceil = 4$, $\lfloor -3.1 \rfloor = -4$, $\lceil -3.1 \rceil = -3$.

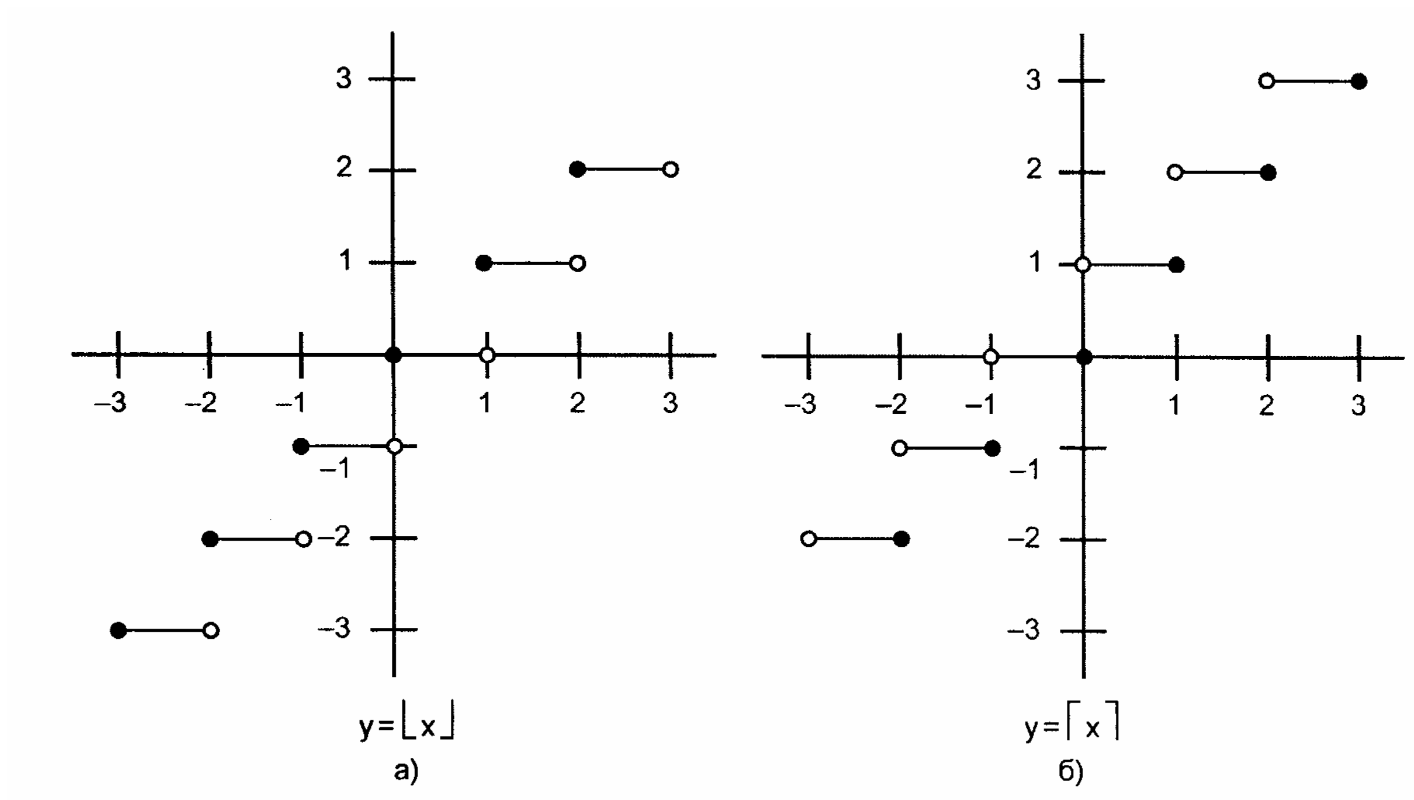


Рис. 6. Підлога і стеля

У курсі дискретної математики використовуються й інші типи функцій – поліноми, логарифмічні та експоненціальні функції, які відображають R в R.

Зростання функцій. Оцінки складності алгоритмів: *O*-велике нотація

Характеристики зростання функцій використовують у комп'ютерних науках для оцінювання часової складності алгоритмів. Вивченню алгоритмів присвячено останню тему нашого курсу; тут ми лише зазначимо, що *алгоритм – це сукупність правил для розв'язування певного класу однотипних задач, які відрізняються початковими даними* (говорять ще – «масова задача»; якщо задати конкретні початкові дані, то одержимо «індивідуальну задачу»). Початкові дані – це вхідна інформація або вхідні дані для алгоритму. Оскільки час виконання алгоритму залежить від розміру вхідних даних, його можна визначити як функцію від обсягу вхідної інформації. Тут добрим прикладом є задачі сортування. У цих задачах вхідні дані – це список елементів, які підлягають сортуванню, а результат – ті самі елементи, відсортовані в порядку зростання або спадання. Наприклад, вхідний список 3, 1, 2, 6, 1, 9, 5 буде перетворено у вихідний список 1, 1, 2, 3, 5, 6, 9 (сортування за зростанням). Як міру обсягу вхідної інформації для алгоритму сортування природно вибрати кількість елементів, які підлягають сортуванню, або, інакше кажучи, довжину вхідного списку. Загалом довжина вхідних даних – придатна міра їхнього обсягу.

Як синонім терміну «*вхідні дані*» часто використовують термін «*входи*». Означимо *часову складність* (у подальшому – *складність*) алгоритму як функцію f, яка ставить у відповідність кожному невід'ємному цілому числу n час роботи $f(n)$ алгоритму в найгіршому випадку на входах довжиною n. Іншими словами, $f(n)$ – максимальний час роботи алгоритму по всіх входах довжиною n. Довжина входу, як уже було сказано, характеризує розмір задачі. Час роботи алгоритму вимірюють у кроках (операціях), що виконуються на ідеалізованому комп'ютері.

Аналіз ефективності алгоритмів полягає в з'ясуванні питання: як швидко зростає функція *f*(*n*) зі збільшенням *n*? Для порівняння швидкості зростання двох функцій *f*(*n*) та *g*(*n*) використовують таке поняття.

Нехай f та g – функції з множини Z цілих чисел або з множини R дійсних чисел в множину R дійсних чисел. Говорять, що $f(x)=O(g(x))$ якщо існують константи C і k такі, що

$$|f(x)|\le C|g(x)|$$

для всіх $x>k$. Говорять також, що $f(x)$ є $O(g(x))$ (читають «O-велике від $g(x)$» і називають O*-велике оцінкою*).

У подальших докладних обговореннях ми майже завжди матимемо справу з функціями, які набувають лише додатні значення. Всі позначення абсолютних величин в оцінках для таких функцій можна усунути.

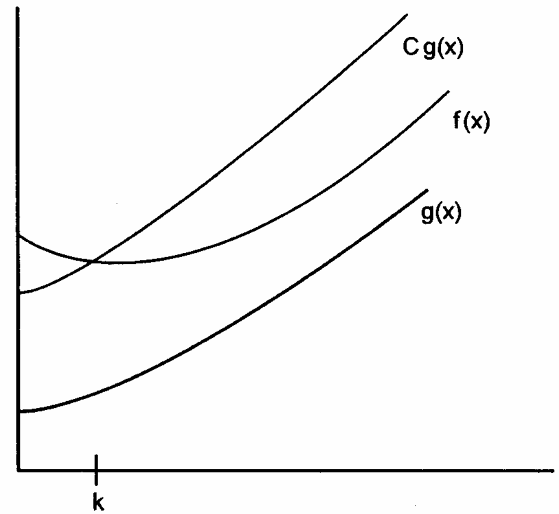


Рис. 7

Рис. 7 ілюструє співвідношення $f(x)=O(g(x))$. Тут $f(x)<Cg(x)$ для $x>k$

Приклад. Покажемо, що $f(x) = x^2 + 2x + 1$ є $O(x^2)$. Зазначимо, що ми можемо оцінити зростання $f(x)$ при $x > 1$, оскільки $x < x^2$ і $1 < x^2$ при $x > 1$. Звідси випливає, що

$$0 \le x^2 + 2x + 1 \le x^2 + 2x^2 + x^2 = 4x^2$$

при $x > 1$, як це показано на **рис. 8**.

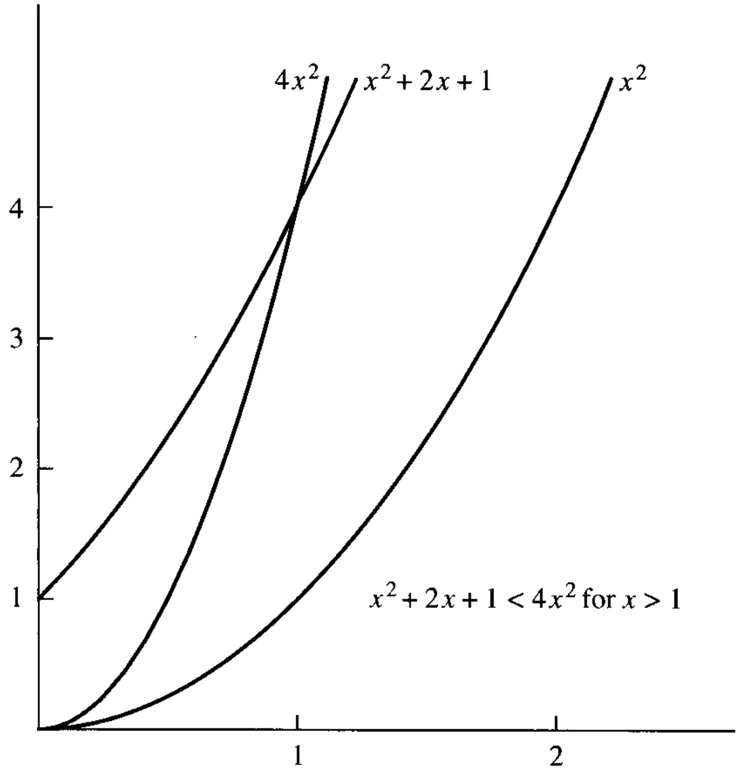


Рис. 8

Як альтернативу, ми можемо оцінити $f(x)$ коли $x > 2$. Коли $x > 2$, матимемо $2x < x^2$ та $1 < x^2$. Отже, якщо $x > 2$, матимемо

$$0 \le x^2 + 2x + 1 \le x^2 + x^2 + x^2 = 3x^2$$

Приклад. Знайдемо оцінку для функцій $f(n)=n!$ та $f(n)=\log n!$

Послідовно маємо:

$$n!=1\cdot 2\cdot 3\cdot\ldots\cdot n\le n\cdot n\cdot n\cdot\ldots\cdot n\le n^n.$$

Ця нерівність доводить, що $n!=O\left(n^n\right)$. Беручи логарифм від обох частин цієї нерівності, дістанемо $\log n!\le\log n^n$, звідки $\log n!=O(n\log n)$.

Поліноми часто використовують для оцінок зростання функцій. Наведемо результат, який завжди може бути використаний для оцінки зростання полінома.

Теорема. Нехай $p(x)=a_m x^m+a_{m-1}x^{m-1}+\ldots+a_1 x+a_0$, де a_m, a_{m-1}, …, a_1, a_0 – дійсні числа. Тоді $p(x)=O\left(x^m\right)$.

Доведення. Використовуючи нерівність трикутника, одержимо (для $x>1$):

$$|p(x)|=\left|a_m x^m+a_{m-1}x^{m-1}+\ldots+a_1 x+a_0\right|\le|a_m|x^m+|a_{m-1}|x^{m-1}+\ldots+|a_1|x+|a_0|=$$

$$=x^m\left(|a_m|+|a_{m-1}|/x+\ldots+|a_1|/x^{m-1}+|a_0|/x^m\right)\le x^m\left(|a_m|+|a_{m-1}|+\ldots+|a_1|+|a_0|\right).$$

Із цього випливає, що

$$|p(x)|\le Cx^m,$$

де $C=|a_m|+|a_{m-1}|+\ldots+|a_1|+|a_0|$, $x>1$. Отже, $p(x)=O\left(x^m\right)$.

У разі використання нотації $O(g(x))$ функцію g у співвідношенні $f(x) = O(g(x))$ вибирають настільки малою, наскільки це можливо (здебільшого із множини функцій, які вважають еталонними). Вираз «складність алгоритму є (дорівнює, складає) $O(g(n))$» означає, що функція $f(n)$, яка визначає складність алгоритму, є $O(g(n))$. Тут n – довжина входу. Як еталони для оцінок складності алгоритмів використовують такі функції (записані в порядку зростання складності):

$$1,\ \ \log n,\ n,\ n\log n,\ n^2,\ n^3,\ n^4,\ 2^n,\ n!.$$

На рис. 9 зображено графіки цих функцій. *Зверніть увагу, що шкала на вісі ординат – логарифмічна* (кожна поділка дорівнює подвоєній попередній, тому графік функції 2^n – пряма).

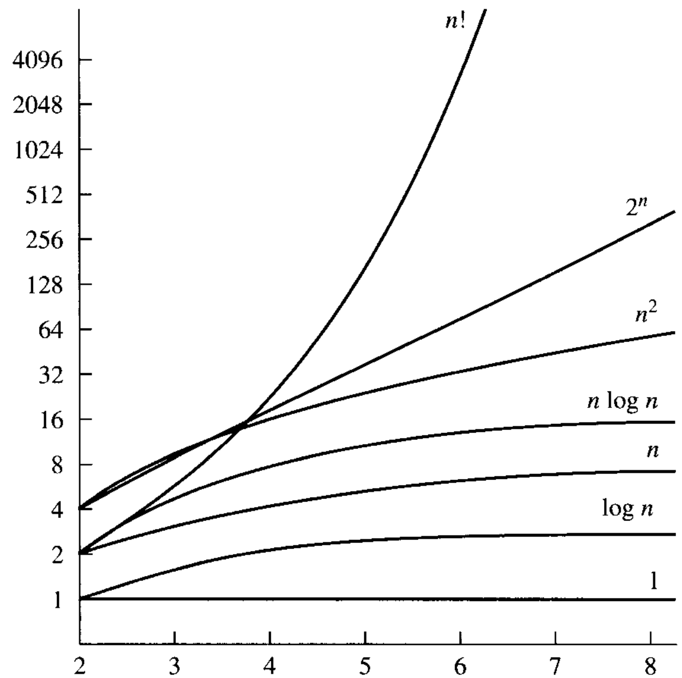


Рис. 9

Зокрема, складність $O(1)$ означає, що час роботи відповідного алгоритму *не залежить* від довжини входу. Алгоритм зі складністю $O(n)$ називають *лінійним*. Такий алгоритм для переважної більшості задач є найліпшим (за порядком) щодо складності.

Алгоритм, складність якого дорівнює $O(p(n))$, де $p(n)$ – деякий поліном, називають *поліноміальним*. Часто замість $O(p(n))$ пишуть $O(n^m)$, де m – константа (наприклад, $m = 2,3$ або 4, тут m – степінь полінома). Особливу роль поліноміальні алгоритми відіграватимуть при вивченні теми «Моделі обчислень». Тут лише зазначимо, що всі задачі дискретної математики, *які вважають важкими для алгоритмічного розв'язування*, нині не мають поліноміальних алгоритмів. Крім того, поняття «поліноміальний алгоритм» зараз є найпоширенішою формалізацією поняття «ефективний алгоритм».

Алгоритми, часова складність яких не піддається подібній оцінці, називають *експоненціальними*. Більшість експоненціальних алгоритмів – це просто варіанти «повного перебору», тоді як поліноміальні алгоритми здебільшого можна побудувати лише тоді, коли вдається заглибитись в суть задачі.

Задачу називають *важкорозв'язною*, якщо для її розв'язування не існує поліноміального алгоритму.

ДОДАТОК 1

У цій таблиці наведено загальну термінологію , яку використовують для опису (часової) складності алгоритмів.

Commonly Used Terminology for the Complexity of Algorithms.	
Complexity	***Terminology***
$O(1)$	constant complexity
$O(\log n)$	logarithmic complexity
$O(n)$	linear complexity
$O(n \log n)$	$n \log n$ complexity
$O(n^b)$	polynomial complexity
$O(b^n)$, where $b > 1$	exponential complexity
$O(n!)$	factorial complexity

ДОДАТОК 2

У цій таблиці подано час, необхідний для розв'язування задач різних розмірів алгоритмами із зазначеною кількістю бітових операцій. Час, більший 10^{100}, позначено зірочкою. Ця таблиця побудована із розрахунку, що бітова операція займає 10^{-9} сек., що відповідало найшвидшим комп'ютерам 1988 року.

The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-9} sec	10^{-8} sec	3×10^{-8} sec	10^{-7} sec	10^{-6} sec	3×10^{-3} sec
10^2	7×10^{-9} sec	10^{-7} sec	7×10^{-7} sec	10^{-5} sec	4×10^{13} yr	*
10^3	1.0×10^{-8} sec	10^{-6} sec	1×10^{-5} sec	10^{-3} sec	*	*
10^4	1.3×10^{-8} sec	10^{-5} sec	1×10^{-4} sec	10^{-1} sec	*	*
10^5	1.7×10^{-8} sec	10^{-4} sec	2×10^{-3} sec	10 sec	*	*
10^6	2×10^{-8} sec	10^{-3} sec	2×10^{-2} sec	17 min	*	*

Ця сама таблиця, побудована із розрахунку, що бітова операція займає 10^{-11} сек., що відповідало найшвидшим комп'ютерам 2012 року. Як можна побачити, для важкорозв'язних задач за чверть століття нічого не змінилося!

The Computer Time Used by Algorithms.

Problem Size	*Bit Operations Used*					
n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
10	3×10^{-11} s	10^{-10} s	3×10^{-10} s	10^{-9} s	10^{-8} s	3×10^{-7} s
10^2	7×10^{-11} s	10^{-9} s	7×10^{-9} s	10^{-7} s	4×10^{11} yr	*
10^3	1.0×10^{-10} s	10^{-8} s	1×10^{-7} s	10^{-5} s	*	*
10^4	1.3×10^{-10} s	10^{-7} s	1×10^{-6} s	10^{-3} s	*	*
10^5	1.7×10^{-10} s	10^{-6} s	2×10^{-5} s	0.1 s	*	*
10^6	2×10^{-10} s	10^{-5} s	2×10^{-4} s	0.17 min	*	*