Основні вимоги до алгоритмів. Машини Тьюрінга. Обчислення числових функцій на машинах Тьюрінга.

Основні вимоги до алгоритмів

Алгоритм — послідовність, система, набір систематизованих правил виконання обчислювального процесу, що обов'язково приводить до розв'язання певного класу задач після скінченного числа операцій. При написанні комп'ютерних програм алгоритм описує логічну послідовність операцій. Для візуального зображення алгоритмів часто використовують блок-схеми.

Кожен алгоритм є списком добре визначених інструкцій для розв'язання задачі. Починаючи з початкового стану, інструкції алгоритму описують процес обчислення, які відбуваються через послідовність станів, які, зрештою, завершуються кінцевим станом. Перехід з одного стану до наступного не обов'язково детермінований — деякі алгоритми містять елементи випадковості.

Поняття алгоритму належить до первісних, основних, базисних понять математики, таких, як множина чи натуральне число. Обчислювальні процеси алгоритмічного характеру (арифметичні дії над цілими числами, знаходження найбільшого спільного дільника двох чисел тощо) відомі людству з глибокої давнини. Проте, в явному вигляді поняття алгоритму сформувалося лише на початку двадцятого століття.

Доведення коректності алгоритмів. Разом з поширенням інформаційних технологій збільшився ризик програмних збоїв. Одним зі способів уникнення помилок в алгоритмах та їхніх реалізаціях є доведення коректності систем математичними засобами.

Використання математичного апарату для аналізу алгоритмів та їхньої реалізацій називають формальними методами. Формальні методи передбачають застосування формальних специфікацій та, зазвичай, набору інструментів для синтаксичного аналізу та доведення властивостей специфікацій. Абстрагування від деталей реалізації дозволяє встановити властивості системи незалежно від її реалізації. Крім того, точність та однозначність математичних тверджень дозволяє уникнути багатозначності та неточності природних мов.

За гіпотезою Річарда Мейса «уникнення помилок краще за усунення помилок». За гіпотезою Гоара «доведення програм розв'язує проблему коректності, документації та сумісності». Доведення коректності програм дозволяє виявляти їхні властивості стосовно всього діапазону вхідних даних. Для доведення коректності програм, поняття коректності було розширене на два типи:

- *Часткова коректність* — програма дає коректний результат для тих випадків, коли вона завершується.
- *Повна коректність* — програма завершує роботу та видає коректний результат для всіх елементів з діапазону вхідних даних.

Під час доведення коректності порівнюють текст програми зі специфікацією бажаного співвідношення вхідних-вихідних даних. Для доведень типу Гоара ця специфікація має вигляд тверджень, які називають перед та післяумовами. В сукупності з самою програмою, їх ще називають трійками Гоара. Ці твердження записують так: $P\{Q\}R$

де P — це передумова, що має виконуватись перед запуском програми Q, а R — післяумова, правильна після завершення роботи програми.

Формальні методи було успішно застосовано до широкого кола задач, зокрема: розробка

електронних схем штучного інтелекту систем, чутливих до надійності, безпечності, автоматичних систем на залізниці, верифікації мікропроцесорів, специфікації стандартів та специфікації і верифікації програм.

На неформальному рівні розглянемо **деякі головні принципи**, на яких будують алгоритми, і виявимо, що саме слід уточнити в понятті алгоритму.

1. Будь-який алгоритм застосовують до початкових даних, і він видає результати. У звичних технічних термінах де означає, що алгоритм має входи й виходи. Отже, алгоритм застосовують для розв'язування цілого класу задач із різними початковими даними (цю властивість називають масовістю). Крім того, під час роботи алгоритму з'являються проміжні результати, які використовуються в подальшому. Звідси випливає, що, кожний алгоритм обробляє дані — вхідні, проміжні та вихідні. Оскільки ми збираємось уточнити поняття алгоритму, потрібно уточнити й поняття даних, тобто зазначити, яким вимогам мають задовольняти об'єкти, щоб алгоритми могли з ними працювати.

2. Для розміщення даних потрібна пам'ять. Її зазвичай уважають однорідною й дискретною, тобто такою, що складається з однакових комірок, причому кожна комірка може містити один символ алфавіту даних. Отже, одиниці виміру обсягу даних і пам'яті узгоджені, при цьому пам'ять може бути нескінченною. Питання про те, чи потрібна одна пам'ять, чи декілька й, зокрема, чи потрібна окрема пам'ять для кожного з трьох типів даних, вирішують по-різному.

3. Алгоритм складається з окремих елементарних кроків (або дій), причому множина різних кроків, з яких складено алгоритм, скінченна. Типовий приклад множини елементарних дій — система команд процесора. Зазвичай на елементарному кроці обробляється фіксована кількість символів, проте є команди, які працюють із полями пам'яті зі змінною довжиною.

4. Послідовність кроків алгоритму детермінована, тобто після кожного кроку зазначено, який крок робити далі, або виконується команда зупинки, після чого робота алгоритму вважається закінченою.

5. Алгоритм має бути результативним, тобто зупинятися після скінченної кількості кроків (залежно від початкових даних) із зазначенням того, що вважати результатом. Зокрема, алгоритм для обчислення функції f{x) має зупинятися після скінченної кількості кроків для будь-якого x із області визначення функції *f*. У такому разі говорять, що алгоритм збігається. Проте перевірити результативність (збіжність) значно важче, ніж вимоги, викладені в пп. 1-4. На відміну від них збіжність переважно неможливо виявити простим переглядом опису алгоритму. Загального ж методу перевірки збіжності, придатного для довільного алгоритму А й довільних початкових даних а, узагалі не існує .

Потрібно розрізняти:

♦ опис алгоритму (інструкцію чи програму);

♦ механізм реалізації алгоритму (наприклад, персональний комп'ютер), який містить засоби запуску, зупинки, реалізації елементарних кроків, видачі результатів і забезпечення детермінованості, тобто керування перебігом обчислення;

♦ процес реалізації алгоритму, тобто послідовність кроків, яка буде породжена в разі застосування алгоритму до конкретних даних.

Уважатимемо, що опис алгоритму й механізм його реалізації скінченні (пам'ять, як ми вже зазначали, може бути нескінченною, але вона не входить у механізм). Вимога скінченності процесу реалізації збігається з вимогою результативності.

Ми сформулювали головні вимоги до алгоритмів. Прості поняття, використані в них, інтуїтивні. Тому в теорії алгоритмів застосовують інший підхід: вибирають скінченний набір основних об'єктів, які оголошують елементарними, і скінченний набір способів побудови з них

нових об'єктів. Уточненням поняття „дані" вважатимемо множини слів (ланцюжків) у скінченних алфавітах. Для уточнення детермінізму використовуватимемо опис механізму реалізації алгоритму. Крім того, потрібно зафіксувати набір елементарних кроків і домовитися про організацію пам'яті. Коли це буде зроблено, отримаємо конкретну алгоритмічну модель. Алгоритмічні моделі, які ми розглянемо в цьому розділі, можна вважати формалізацією поняття „алгоритм". Це означає, що вони мають бути універсальними, тобто за їх допомогою можна описати будь-які алгоритми. Тому може виникнути природне заперечення проти запропонованого підходу: чи не призведе вибір конкретних засобів до втрати загальності формалізації? Маючи на увазі головні цілі, поставлені під час розробки теорії алгоритмів, — універсальність і пов'язану з нею можливість говорити в рамках якоїсь моделі про властивості алгоритмів узагалі, — це заперечення можна зняти так. По-перше, доводять звідність одних моделей до інших, тобто те, що будь-який алгоритм, описаний засобами однієї моделі, можна описати й засобами іншої. По-друге, завдяки взаємній звідності моделей у теорії алгоритмів удалося виробити інваріантну щодо моделей систему понять, яка дає змогу говорити про властивості алгоритмів незалежно від того, яку формалізацію алгоритму вибрано. Ця система ґрунтується на понятті обчислюваної функції, тобто такої, для обчислення якої існує алгоритм.

Однак, хоча загальність формалізації в конкретній моделі не втрачається, унаслідок різного вибору початкових засобів виникають різні моделі. Можна виділити три основні типи універсальних алгоритмічних моделей, які різняться початковими евристичними міркуваннями відносно того, що таке алгоритм.

У *першому типі* поняття алгоритму пов'язане з найтрадиційнішими поняттями математики — обчисленнями та числовими функціями (числовою називають функцію, значення якої та значення її аргументів — невід'ємні цілі числа). Найпопулярніша модель цього типу — рекурсивні функції — історично перша формалізація поняття алгоритму. Рекурсивні функції — клас функцій, введений як уточнення класу обчислюваних функцій.

В математиці загальноприйнятою є теза про те, що клас функцій, для обчислення яких існують алгоритми, при найширшому розумінні алгоритму, збігається з класом рекурсивних функцій. У зв'язку з цим, рекурсивні функції грають важливу роль в математиці та її застосуваннях, в першу чергу, в математичній логіці, основах математики та кібернетиці, як ефективно обчислювані функції. Тільки такі функції можна обчислювати на електронних обчислювальних машинах та інших цифрових пристроях.

Другий тип ґрунтується на уявленні про алгоритм як детермінований пристрій, здатний виконувати в кожний окремий момент лише дуже примітивні операції. У такому разі не залишається сумнівів у однозначності алгоритму й елементарності його кроків. Окрім того, евристика цих моделей близька до комп'ютерів. Основна теоретична модель цього типу (створена в 30-х роках XX століття — раніше комп'ютерів) — машина Тьюрінга. Основна ідея, що лежить в основі машини Тьюрінга, дуже проста. Машина Тьюрінга — це абстрактна машина, що працює зі стрічкою окремих комірок, в яких записано символи.

Нарешті, *третій тип* алгоритмічних моделей — це перетворення слів у довільних алфавітах; у цих моделях елементарні операції — це підстановки, тобто заміна частини слова (підслова) іншим словом. Приклади моделей цього типу — канонічні системи Поста й нормальні алгоритми Маркова. Нормальні алгоритми Маркова — це система послідовних застосувань підстановок, які реалізують певні процедури отримання нових слів з базових, побудованих із символів деякого алфавіту. Як і машини Тьюрінга, нормальні алгоритми не виконують самих обчислень: вони лише виконують перетворення слів шляхом заміни літер за заданими правилами. Нормально обчислюваною називають функцію, яку можна реалізувати

нормальним алгоритмом. Тобто, алгоритмом, який кожне слово з множини припустимих даних функції перетворює на її вихідні значення. Творець теорії нормальних алгоритмів А.А. Марков висунув гіпотезу, яка отримала назву принцип нормалізації Маркова:

"Для знаходження значень функції, заданої в деякому алфавіті, тоді і лише тоді існує деякий алгоритм, коли функція нормально обчислювана."

Машини Тьюрінга

Машина Тюрінга — це абстрактна машина (автомат), що працює зі стрічкою, що складається із окремих комірок, в яких записано символи. Машина також має голівку для запису та читання символів із комірок і яка може рухатись вздовж стрічки.

На кожному кроці машина зчитує символ із комірки, на яку вказує голівка та, на основі зчитаного символу та внутрішнього стану, робиться наступний крок. При цьому, машина може змінити свій стан, записати інший символ в комірку або пересунути голівку на одну комірку ліворуч або праворуч.

1)

0	1	0	1	0	0	1	1	1	**1**	0	0	1	0	1	1	0	0	0

2)

0	1	0	1	0	0	1	1	**1**	1	0	0	1	0	1	1	0	0	0

3)

0	1	0	1	0	0	1	1	**1**	0	0	0	1	0	1	1	0	0	0

4)

0	1	0	1	0	0	1	1	**0**	0	0	0	1	0	1	1	0	0	0

5)

0	1	0	1	0	0	1	1	**0**	0	0	0	1	0	1	1	0	0	0

6)

0	1	0	1	0	0	1	1	**0**	0	0	0	1	0	1	1	0	0	0

7)

0	1	0	1	0	0	1	1	**1**	0	0	0	1	0	1	1	0	0	0

8)

0	1	0	1	0	0	1	1	1	**0**	0	0	1	0	1	1	0	0	0

9)

0	1	0	1	0	0	1	1	1	0	**0**	0	1	0	1	1	0	0	0

10) 0	1	0	1	0	0	1	1	1	0	**0**	0	1	0	1	1	0	0	0

Приклад 1. Машина може бути виконана, щоб обчислити послідовність 0 1 0 1 0 1 ..." (0 <пробіл> 1 <пробіл> 0 ...)

Конфігурація		**Поведінка**	
Стан	**Символ стрічки**	**Операції стрічки**	**Остаточний стан**
b	blank(пробіл)	P0, R	c

c	blank	R	e
e	blank	P1, R	f
f	blank	R	b

Обчислення числових функцій на машинах Тюрінга

Числовою називають функцію $f(x_1,\dots,x_n)$, значення якої та значення її аргументів — невід'ємні цілі числа. Розглянемо часткові числові функції, визначені, узагалі кажучи, не для всіх значень аргументів. Для обчислення числових функцій на машинах Тюрінга застосовують спеціальне кодування чисел. Наприклад [ЗО], невід'ємне ціле число m можна задати набором з $(m+1)$одиниць, який позначатимемо 1^{n+1}: 0—як 1, 1 —як 11, 2 —як 111 тощо.

Числову функцію$f(x_1,\dots,x_n)$ називають *обчислюваною за Тюрінгом*, якщо існує така машина Тюрінга *T,* що для довільних цілих невід'ємних чисел $m_1, m_2, \dots, m_n$, якщо$f(m_1, m_2, \dots, m_n) = m$, то машина *T* застосовна до слова $1^{m_1+1}\Lambda 1^{m_2+1}\Lambda\dots\Lambda 1^{m_n+1}$ і в заключній конфігурації на якомусь відрізку стрічки буде записано слово 1^{m+1}, а решта комірок (якщо такі є) виявляться порожніми. Якщо ж значення $f(m_1, m_2, \dots, m_n)$ не визначено, то машина *T* незастосовна до слова$1^{m_1+1}\Lambda 1^{m_2+1}\Lambda\dots\Lambda 1^{m_n+1}$.

Приклад 2. Побудуємо машину Тюрінга, яка обчислює числову функцію $s(x) - x + 1$. Зовнішній алфавіт $A = [\Lambda, 1]$, множина станів $Q = [q_0, q_1]$, а команди можна задати так: $q_1 1 \to 1\Pi q_1, q_1\Lambda \to 1Hq_0$. Конфігурації, що виникають у разі обчислення *s(1),* зображено на рис. 1.

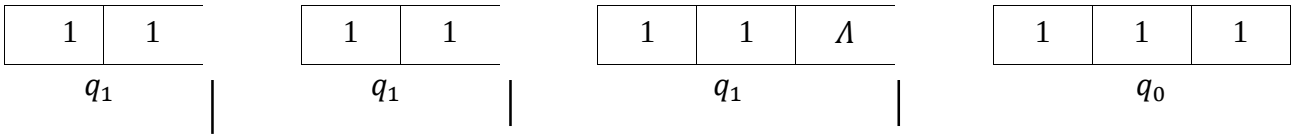
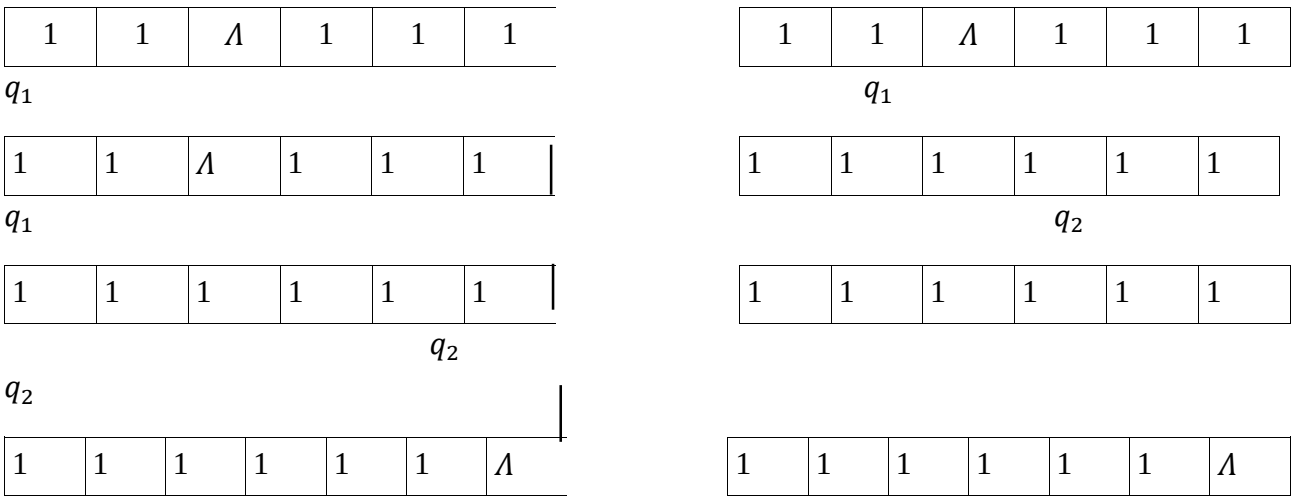


Рис. 1

Приклад 3. Побудуємо машину Тюрінга, яка обчислює числову функцію $f(x,y) = x + y$. Зовнішній алфавіт $A = [\Lambda, 1]$, множина станів $Q = [q_0, q_1, q_2, q_3, q_4]$. У процесі роботи машини головка рухається вправо, і символ Λміж аргументами замінюється на 1. Далі продовжується рух праворуч до першої комірки із символомΛ , після чого головка починає рухатись уліво та стирає дві останні 1 поспіль. Команди можна задати так: $q_1 1 \to 1\Pi q_1$, , $q_2 1 \to 1\Pi q_2$, $q_2\Lambda \to \Lambda\Pi q_3$, $q_3 1 \to \Lambda\Pi q_1$, $q_1 1 \to \Lambda H q_0$. Конфігурації, що виникають під час обчислення $f(1.2)$, зображено на рис. 2.



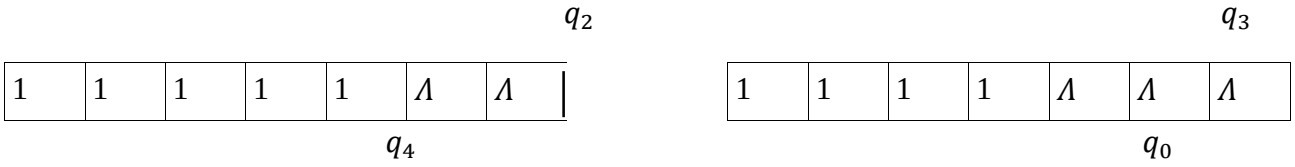


Рис. 2