Розділ I. Алгоритми

Тема 1. Алгоритми та обчислення

1.1. Що таке алгоритм?

Поняття алгоритму інтуїтивно зрозуміло та часто використовується в математиці та комп'ютерних науках. Говорячи неформально, **алгоритм** – це довільна коректно визначена обчислювальна процедура, на **вхід** якої подається деяка величина або набір величин, а результатом виконання якої є **вихідна** величина або набір значень. Таким чином, алгоритм є послідовністю обчислювальних кроків, які перетворюють вхідні величини у вихідні.

Алгоритм можна також розглядати як інструмент, який призначений для вирішення коректно поставленої обчислювальної задачі. У постановці задачі в загальних рисах визначаються відношення між входом та виходом. В алгоритмі описується конкретна обчислювальна процедура, за допомого якої можна досягнути виконання вказаних відношень.

Можна навести загальні риси алгоритму:

a. *Дискретність інформації*. Кожний алгоритм має справу з даними: вхідними, проміжними, вихідними. Ці дані представляються у вигляді скінченних слів деякого алфавіту.
b. *Дискретність роботи алгоритму*. Алгоритм виконується по кроках та при цьому на кожному кроці виконується тільки одна операція.
c. *Детермінованість алгоритму*. Система величин, які отримуються в кожний (не початковий) момент часу, однозначно визначається системою величини, які були отримані в попередні моменти часу.
d. *Елементарність кроків алгоритму*. Закон отримання наступної системи величин з попередньої повинен бути простим та локальним.
e. *Виконуваність операцій*. В алгоритмі не має бути не виконуваних операцій. Наприклад, неможна в програмі призначити значення змінній «нескінченність», така операція була би не виконуваною. Кожна операція опрацьовує певну ділянку у слові, яке обробляється.
f. *Скінченність алгоритму*. Опис алгоритму повинен бути скінченним.
g. *Спрямованість алгоритму*. Якщо спосіб отримання наступної величини з деякої заданої величини не дає результату, то має бути вказано, що треба вважати результатом алгоритму.
h. *Масовість алгоритму*. Початкова система величин може обиратись з деякої потенційно нескінченної множини.

Розглянемо для прикладу задачу сортування послідовності чисел у зростаючому порядку. Ця задача часто виникає на практиці і, фактично, буде центральною проблемою першого розділу даного курсу. **Задача сортування** визначається формально наступним чином.

Вхід: послідовність n чисел $\langle a_1, a_2, \ldots, a_n \rangle$

Вихід: перестановка $\langle a_1', a_2', \ldots, a_n' \rangle$ вхідної послідовності таким чином, що для всіх її членів виконується співвідношення $a_1' \le a_2' \le \ldots \le a_n'$.

Наприклад, якщо на вхід подається послідовність ⟨31, 41, 59, 26, 11, 58⟩, то вивід алгоритму сортування повинен бути таким: ⟨26, 31, 41, 41, 58, 59⟩. Подібна вихідна послідовність називається **екземпляром задачі** сортування. Взагалі, екземпляр задачі складається із входу, який необхідний для розв'язання задачі та який задовольняє усім обмеженням, які присутні в постановці задачі.

В комп'ютерних науках сортування є основною операцією (у багатьох програмах вона використовується в якості проміжного кроку), в результаті чого з'явилось багато якісних алгоритмів сортування. Вибір найбільш адекватного алгоритму залежить від багатьох факторів, в тому числі й від кількості елементів для сортування, від їх порядку у вхідній послідовності, від можливих обмежень, які накладаються на членів послідовності.

Кажуть, що алгоритм є **коректним**, якщо для кожного входу результатом його роботи є коректний вивід. Тоді коректний алгоритм **розв'язує** дану обчислювальну задачу. Якщо алгоритм некоректний, то для деяких входів він може взагалі не завершити свою роботу або видати відповідь, яка відрізняється від очікуваної.

1.2. Для чого вивчати алгоритми?

По-перше, алгоритми є життєво необхідними складовими для рішення будь-яких задач з різноманітних напрямків комп'ютерних наук. Алгоритми відіграють ключову роль у сучасному розвитку технологій. Тут можна згадати такі розповсюджені задачі, як:

- розв'язання математичних рівнянь різної складності, знаходження добутку матриць, обернених матриць;
- знаходження оптимальних шляхів транспортування товарів та людей;
- знаходження оптимальних варіантів розподілення ресурсів між різними вузлами (виробниками, верстатами, працівниками, процесорами тощо);
- знаходження в геномі послідовностей, які співпадають;
- пошук інформації в глобальній мережі Інтернет;
- прийняття фінансових рішень в електронній комерції;
- обробка та аналіз аудіо та відео інформації.

Цей список можна продовжувати й продовжувати і, власно кажучи, майже неможливо знайти таку галузь комп'ютерних наук та інформатики, де б не використовувались ті або інші алгоритми.

По-друге, якісні та ефективні алгоритми можуть бути каталізаторами проривів у галузях, які є на перший погляд далекими від комп'ютерних наук (квантова механіка, економіка та фінанси, теорія еволюції).

І, по-третє, вивчення алгоритмів це також неймовірно цікавий процес, який розвиває наші математичні здібності та логічне мислення.

1.3. Ефективність алгоритмів

Припустимо, швидкодія комп'ютеру та об'єм його пам'яті можна збільшувати до нескінченності. Чи була би тоді необхідність у вивченні алгоритмів? Так, але тільки для того, щоб продемонструвати, що метод розв'язку має скінченний час роботи і що він дає правильну відповідь. Якщо б комп'ютери були необмежено швидкими, підійшов би довільний коректний метод рішення задачі. Звісно, тоді найчастіше обирався би метод, який найлегше реалізувати.

Сьогодні є дуже потужні комп'ютери, але їх швидкодія не є нескінченно великою, як і пам'ять. Таким чином, час обчислення – це такий само обмежений ресурс, як і об'єм необхідної пам'яті. Цими ресурсами слід користуватись розумно, чому й сприяє застосування алгоритмів, які ефективні в плані використання ресурсів часу та пам'яті.

Алгоритми, які розроблені для розв'язання однієї та тієї самої задачі, часто можуть дуже сильно відрізнятись за ефективністю. Ці відмінності можуть бути набагато більше помітними, чим ті, які викликані застосуванням різного апаратного та програмного забезпечення.

Як зазначалось вище, в цьому розділі центральну роль буде присвячено задачі сортування. Перший алгоритм, який буде розглядатись – сортування включенням, для своєї роботи вимагає часу, кількість якого оцінюється як $c_1 n^2$, де n – розмір вхідних даних (кількість елементів у послідовності для сортування), c_1 – деяка стала. Цей вираз вказує на те, як залежить час роботи алгоритму від об'єму вхідних даних. У випадку сортування

включенням ця залежність є квадратичною. Другий алгоритм – сортування злиттям – потребує часу, кількість якого оцінюється як $c_2 n\log_2 n$. Зазвичай константа сортування включенням менше константи сортування злиттям, тобто $c_1<c_2$, але як ми пересвідчимось у наступних темах, ці константи не відіграють ролі у порівнянні швидкодії різних алгоритмів. Адже зрозуміло, що функція n^2 зростає швидше зі збільшенням n, аніж функція $n\log_2 n$. І для деякого значення $n = n_0$ буде досягнуто такий момент, коли вплив різниці констант перестане мати значення і надалі функція $c_2 n\log_2 n$ буде менша за $c_1 n^2$ для будь-яких $n > n_0$.

Для демонстрації цього розглянемо два комп'ютери – А та Б. Комп'ютер А більш швидкий і на ньому працює алгоритм сортування, а комп'ютер Б більш повільний і на ньому працює алгоритм сортування методом злиття. Обидва комп'ютери повинні виконати сортування множини, яка складається з мільйона чисел. Припустимо, що комп'ютер А виконує мільярд операцій в секунду, а комп'ютер Б – лише десять мільйонів, тобто А працює в 100 разів швидше за Б. Щоб різниця стала більш відчутною, припустимо що код методу включення написаний найкращим програмістом в світі із використанням команд процесору, і для сортування n чисел за цим алгоритмом потрібно виконати $2n^2$ операцій (тобто $c_1=2$). Сортування методом злиття на комп'ютері Б написано програмістом початківцем із використанням мови високого рівня і отриманий код потребує $50n\log_2 n$ операцій (тобто $c_2=50$). Таким чином для сортування мільйона чисел комп'ютеру А буде потрібно

$$\frac{2\cdot\left(10^6\right)^2 \text{ команд}}{10^9 \text{ команд / с}} = 2000 \text{ с},$$

а комп'ютеру Б –

$$\frac{50\cdot 10^6\cdot\log_2 10^6 \text{ команд}}{10^7 \text{ команд / с}} \approx 100 \text{ с}.$$

Тож, використання коду, час роботи якого зростає повільніше, навіть при поганому комп'ютері та поганому компіляторі потребує на порядок менше процесорного часу! Для сортування 10 мільйонів чисел перевага сортування злиттям стає ще більш відчутною: якщо сортування включенням потребує для такої задачі приблизно 2,3 дня, то для сортування злиттям – менше 20 хвилин. Загальне правило таке: чим більша кількість елементів для сортування, тим помітніше перевага сортування злиттям.

Наведений вище приклад демонструє, що алгоритми, як і програмне забезпечення комп'ютеру, являють собою **технологію**. Загальна продуктивність системи настільки ж залежить від ефективності алгоритму, як і від потужності апаратних засобів.

1.4. Золоте правило розробників алгоритмів

Тепер розглянемо для прикладу просту задачу, яка відома усім ще з початкової школи, а також метод розв'язання цієї задачі – множення двох цілих чисел. Цю задачу можна описати наступним чином:

Вхід: 2 цілих n-розрядних числа x та y

Вихід: добуток чисел $x\cdot y$

Розглянемо приклад для чисел x = 5678 та y = 1234. Результат відомого з дитинства методу множення в стовпчик буде виглядати наступним чином:

```
  5678
 ×1234
------
 22712
17034
11356
5678
------
7006652
```

Легко помітити, що елементарні операції, які тут використовуються, це – множення та додавання однорозрядних чисел. Припустивши, що операція множення займає більше часу аніж операція додавання для однієї пари чисел, оцінимо кількість таких елементарних операцій. Всі вони виконується в області, яка вище позначена сірим кольором. В даному прикладі кількість елементарних операцій добутку становитиме $16 = 4^2$, а в загальному випадку становитиме n^2. Тож, кількість операцій для добутку двох цілих n-розрядних чисел методом множення у стовпчик оцінюється як cn^2, де c – деяка стала.

Проте чи можемо ми покращити цей результат, отримавши метод добутку чисел, який буде працювати швидше? Щоб мотивувати себе для пошуку такого методу, наведемо цитату з книги «Розробка та аналіз комп'ютерних алгоритмів» (Аго, Гопкрофт, Ульман, 1974): «Можливо найбільш важливим принципом для гарного розробника алгоритмів є відмова від того, щоб бути задоволеним результатом». Слідуючи цьому правилу, розглянемо ще раз детальніше природу об'єктів задачі добутку чисел.

За умовою на вхід подається два n-розрядних числа. Припустимо ми розіб'ємо кожне число навпіл, отримавши, так звані, верхнє та нижнє півслова. Тобто, можна записати $x = 10^{n/2}a + b$ та $y = 10^{n/2}c + d$, де a, b, c, d – цілі $n/2$-розрядні числа. Тоді добуток $x \cdot y$ можна представити так:

$$xy = \left(10^{n/2}a + b\right)\cdot\left(10^{n/2}c + d\right) = 10^n ac + 10^{n/2}\left(ad + bc\right) + bd\,. \qquad (1.1)$$

Таким чином ми природно підійшли до рекурсивного методу обчислення добутку двох цілих чисел, який зводить обрахунок добутку двох n-розрядних чисел до обрахунку чотирьох добутків $n/2$-розрядних чисел. Спробуємо з'ясувати, чи покращиться таким чином швидкість добутку двох чисел. Кожне з чисел a, b, c, d мають $n/2$ розрядів, а відтак добуток будь-якої їх пари (якщо використовувати для нього старий алгоритм множення у стовпчик) займатиме $c(n/2)^2$ операцій, тобто $cn^2/4$. Чотири таких добутки в сумі знову дадуть початковий результат: $4 \cdot cn^2/4 = cn^2$. Отже, виграш за часом не було отримано.

Невже не можливо покращити результат роботи методу множення чисел у стовпчик? Насправді, можливо і відповіддю на це питання є метод множення Карацуби (1960). Якщо подивитись на формулу (1.1), то помітимо, що насправді важливими є не чотири добутки, а три: ac, bd та $(ad + bc)$, тобто елементи ad та bc нас не цікавлять самі по собі, а лише їх сума. Чи можна отримати їх суму перемноживши лише два числа? Так:

$$(a + b)(c + d) = ac + ad + bc + bd = (ad + bc) + ac + bd$$
$$ad + bc = (a + b)(c + d) - ac - bd \qquad (1.2)$$

Таким чином, суму $(ad + bc)$ можна отримати з добутку двох $n/2$-розрядних числа (можливо, $n/2 + 1$) $(a+b)$ та $(c+d)$, а також добутків ac та bd, які ми вже маємо. І, отже, кількість рекурсивних викликів скоротились з чотирьох до трьох. Аналіз швидкості методу множення Карацуби приводить до оцінки $3n^{\log_2 3} \approx 3n^{1,585}$.

Цей приклад красномовно свідчить, що простір для руху розробника алгоритмів є набагато більшим ніж може видаватись на початку.

Література

Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford (2001) [1990]. Introduction to Algorithms (2nd ed.). MIT Press and McGraw-Hill. ISBN 0-262-03293-7. Глава 1.