ЛЕКЦІЯ 2. АЛГОРИТМИ. СКЛАДНІСТЬ АЛГОРИТМІВ.

2.1. Зображення алгоритмів

Алгоритм – це формально описана обчислювальна процедура, яка отримує вихідні дані та видає результат обчислень на виході [3]. Вихідні дані також називаються входом алгоритму або його аргументом. Алгоритми будуються для розв‘язку тих або інших обчислювальних задач. Формулювання задачі описує, яким вимогам повинен задовольняти розв‘язок задачі. Алгоритм, що розв‘язує цю задачу, знаходить об‘єкт, який задовольняє даним вимогам.

Алгоритм вважається правильним, коректним, якщо для будь-якого входу він закінчує роботу та видає результат, що задовольняє вимогам задачі.. Неправильний алгоритм може зовсім не закінчити роботи або дати невірний результат.

Алгоритм відрізняється від системи та програми тим, що в ньому міститься тільки опис дій, що виконуються над даними, але повністю відсутні будь-які описи даних. Алгоритми можуть бути представлені за допомогою таблиць розв'язків, вербально з покроковим описом дій та псевдокодів. Алгоритми містять визначення покрокового процесу обробки даних з описом перетворень даних і функцій керування. Вони можуть бути записані природною мовою, на мові програмування, а також за допомогою математичної або іншої символічної нотації. Назва алгоритму може вказувати на його призначення (наприклад,алгоритм сортування, обертання матриць, гри в "хрестики й нулики" і т.п.) або визначати використовуваний у ньому метод розв'язку.

Зображення алгоритмів найкраще пояснити на прикладі. Розглянемоалгоритм знаходження елемента вектора з найбільшим алгебраїчним значенням.

АЛГОРИТМ GT визначає найбільший за значенням елемент вектора*A* , що містить *n* елементів, і присвоює його значення величині *MAX* . Символ *i*використовується у ролі індексу елемента вектора *A* .

*G*T1. Перевірка умови - чи вектор не порожній?

Якщо *n <1* , то - друк повідомлення; à G*T*6.

GT2. Початок: *MAX = A[1], i=2.*

GT3. Повторення кроків GT4 , G*T5* доти, доки *i<=n*.

GT4. Заміна значення MA*X* , якщо воно менше від значення наступного елемента: якщо*MAX <A[i]* , то *MAX =A[i]*

GT5. i= i + 1.

GT6. Кінець, вихід.

Алгоритму присвоєно ім'я **G*T***. Заним іде короткий опис задачі, яку розв'язуєалгоритм, і водночас описуються всі змінні, які тут використовуються. Післяцього наводиться сам алгоритм у виглядіпослідовності кроків. Кожний крок описується фразою, яка коротко пояснює дію, що виконується. Символ " à " замінює оператор переходу ***go to*** в мовах програмування . Крок GT3 є аналогом заголовку циклу в мовах програмування. Коли значення індексу *i* перевершить величину *n*, відбудеться перехід на крок GT6 (наступний після кінця циклу).

Виконання будь-якого алгоритму починається з кроку *i* та продовжується послідовно доти, доки цю послідовність не порушить оператор умовного або безумовного переходу.

В алгоритмах часто зустрічається слово "ініціалізація", що означає призначення деяких початкових значень змінним алгоритму. Слово "трасування" слід розуміти як запис послідовності виконуваних дій алгоритму і їх результатів для конкретних даних задачі.

2.2. Складність алгоритмів

Час роботи будь-якого алгоритму залежить від кількості вхідних даних. Так при роботі алгоритму сортування час залежить від розміру вхідного масиву. В загальному вивчають залежність часу роботи алгоритму від розміру входу. В одних задачах **розмір входу** це кількість елементів на вході (задача сортування, перетворення Фур'є). В інших випадках розмір входу це загальна кількість бітів, що необхідні для представлення всіх вхідних даних.

Часом роботи алгоритмуназивається число елементарних кроків, які він виконує [3]. Вважається, що один рядок псевдокоду вимагає не більше ніж фіксованого числа операцій.

Розрізняють також виклик функції, на який іде фіксована кількість операцій та виконання функції, яке може бути довгим. Так для алгоритму пошуку максимального за значенням числа в масиві розміром *n*необхідно виконати *n* кроків. Тоді час роботи такого алгоритму позначимо *T(n).*Для кожного алгоритму проводиться оцінка кожної операції, скільки разів ця операція виконується. Всі оцінки додаються і виводиться залежність часу роботиалгоритму від розміру входу.

Якщо алгоритм призначений для обробки рівних за об'ємом даних, тривалість його роботи щораз залишається незмінною й складність алгоритму є постійною. Час роботи алгоритму обробки яких-небудь масивів даних залежить від розмірів цих масивів. Час роботи алгоритму, що виконує тільки операції читання й занесення даних в оперативну пам'ять, визначається формулою $an+b$, де *a*- час, необхідний для читання й занесення в пам'ять однієї інформаційної одиниці, і *b* - час, затрачуваний на виконання допоміжних функцій. Оскільки ця формула виражає лінійну залежність від *n*, складність відповідного алгоритму називають лінійною.

Якщо алгоритм передбачає виконання двох незалежних циклів, перший від *1* до *k,* і другий від *1* до *t*, то загальна кількість кроків буде $k+t$, а час роботи $T(k+t)$. Якщо алгоритм передбачає виконання двох вкладених циклів, від *1* до*k* і від *1* до *t*, то загальна кількість кроків буде $k*t$, а час роботи $T(k*t)$. Це є лінійна функція. І вона є найбільш сприятливою для оцінки швидкодії алгоритму. Чим більша степінь складності алгоритму в залежності від розміру входу (n^2, n^3, n^n…) тим ефективність алгоритму нижча.

Час роботи алгоритму також залежить від подачі вхідних даних. Так дляалгоритму сортування «бульбашкою» суттєво як розміщені елементи вхідногомасиву. Якщо вони частково відсортовані, то вихід за прапорцем відбудеться швидше. Якщо елементи стоять в різнобій, тоалгоритм відпрацює всі цикли повністю.

Складність можна оцінити як стосовно задачі, так і стосовно алгоритму. Оцінками часу роботи алгоритму є максимальний, мінімальний і середній час його виконання. Залежно від використовуваних евристик ці оцінки можуть збігатися або не збігатися.

Складність задачі повинна оцінюватися за реалізаціями правильнихалгоритмів. Вона, являє собою верхню межу для часу роботи алгоритму. Але часто можна знайти також і нижню межу. Очевидно, що для виконання якого-небудь виду обробки *n* елементів потрібен час, принаймні пропорційний *n*.

Час роботи в гіршому випадку є більш цікавим для дослідження ефективностіалгоритму, оскільки:

1) це гарантує, що виконанняалгоритму закінчиться за деякий час, незалежно від розміру входу;

2) на практиці «погані» входи трапляються частіше;

3) час роботи в середньому є достатньо близьким до часу роботи в гіршому випадку.

Розглянемо три позначення асимптотики росту часу роботи алгоритму].

***Q* - позначення (тета)**

Наприклад, якщо час роботи *T(n)*деякого алгоритму становить залежність n^2від розміру входу, то знайдуться такі константи c_1, $c_2 > 0$ і таке число n_0 що$c_1 n^2 \le T(n) \le c_2 n^2$ при всіх $n \ge 0$. Взагалі, якщо *g(n)* - деяка функція, то запис $f(n) = Q(g(n))$ (тета) означає, що знайдуться такіс$_1$, $c_2 > 0$ і таке n_0, що $0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ для всіх $n \ge n_0$. Тобто складністьалгоритму геометрично розміщена в межах умовної смуги між функціями $c_1 g(n)$і $c_2 g(n)$ (рис.2.1.а).

***O*- позначення (о)**

Запис $f(n) = Q(g(n))$включає дві оцінки: верхню і нижню. Їх можна розділити. Говорять, що $f(n) = O(g(n))$, якщо знайдеться така константа $c > 0$ і таке число n_0, що $0 < f(n) \le cg(n)$ для всіх $n \ge n_0$ – верхня оцінка, тобто складністьалгоритму не перевищує деякої функції$cg(n)$ і геометрично знаходиться нижче її графіку (рис.2.1.б).

***Ω* – позначення (омега)**

Говорять, що $f(n) = \Omega(g(n))$, якщо знайдеться така константа $c > 0$ і таке число n_0, що $0 \le cg(n) \le f(n)$ для всіх $n \ge n_0$– нижня оцінка, тобто складністьалгоритму не менша деякої функції $cg(n)$ і геометрично знаходиться вище її графіку (рис.2.1.в).

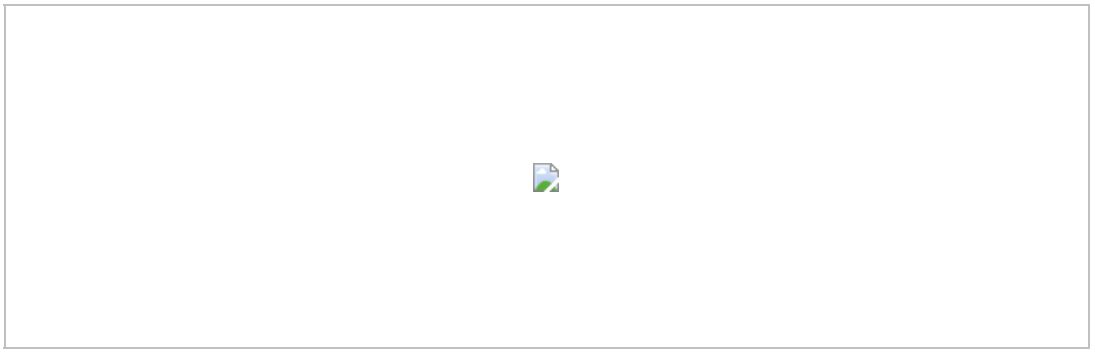

Рис. 2.1. Визначення оцінок складності :

а) для f(n) = Q (g(n)) ; б) для f(n) = O(g(n)); в) для f(n) = Ω(g(n)).

Введені позначення володіють властивостями транзитивності, рефлективності та симетричності.

Транзитивність:

f(n) = Q (g(n)) і g(n) = Q (h(n)) то f(n) = Q (h(n))

f(n) = O(g(n)) і g(n) = O (h(n)) то f(n) = O (h(n))

f(n) = Ω (g(n)) і g(n) = Ω (h(n)) то f(n) = Ω (h(n))

Рефлексивність:

f(n) = Q (f(n)), f(n) = O(f(n)), f(n) = Ω (f(n)) .

Симетричність:

f(n) = Q (g(n)) якщо і тільки якщо g(n) = Q (f(n)).

Наприклад, алгоритм зчитування й занесення в пам‘ять множини даних має оцінку $O(n)$. Алгоритм двійкового пошуку в таблиці з упорядкованими елементами оцінюється як $O(\log_2 n)$. Просте обмінне сортування має оцінку $O(n^2)$, а множення матриць - $O(n^3)$. Коли говорять, що складність сортування рівна $O(n^2)$, під цим мається на увазі, що сортування інформації буде виконано в найкращому випадку за час, пропорційний n^2.

Складність розв'язку задачі можна зменшити, якщо знайти більш вигідний метод (як, наприклад, при заміні лінійного пошуку двійковим) або використовувати оптимізуючі евристики.

У таблиці 2.1 показана залежність різних видів складності алгоритмів від зростання n. Логарифмічна залежність більш прийнятна, ніж лінійна, а лінійна залежність переважніше поліноміальної й експонентної [1].

Таблиця 2.1.

Складність алгоритмів

n	$O(\log_2 n)$	$O(n)$	$O(n1\log_2 n)$	$O(n^2)$	$O(2^n)$	$O(n!)$	$O(n^n)$
1	0	1	0	1	2	1	1
5	2,3219	5	11,6069	25	32	120	3,125
10	3,3219	10	33,2193	100	1,024	$3,6*10^6$	10^{10}
20	4,3219	20	86,4386	400	10^6	$2,4*10^{18}$	10^{26}
30	4,9069	30	147,2067	900	10^9	$2,6*10^{32}$	$2*10^{44}$
100	6,6439	100	664,3856	10000	10^{30}	10^{51}	10^{200}

2.3. Класи алгоритмів

Кожна наукова дисципліна має свої методи одержання результатів. Щодо цього програмування не є винятком. Основна відмінність між задачами полягає в тому, що для одних існують прямі методи розв'язку, а інші не можуть бути вирішені без додаткової інформації, отриманої з відповідей на деякі питання. Варіанти відповідей заздалегідь передбачені.

Якщо задача може бути вирішена прямим способом, говорять, що вона має*детермінований* метод розв'язку [1]. Детерміновані алгоритми завжди забезпечують регулярні розв'язки. В них відсутні елементи, що вносять невизначеність, крім того для них неможлива довільність у виборі рішень, що визначають послідовність дій. Для побудови детермінованих алгоритмів неприпустиме застосування методів проб і помилок. До задач, що мають детермінований розв'язок відносяться математичні рівняння, перевірка даних, друк звітів.

Якщо розв'язок задачі не може бути отриманий прямим методом, а вибирається із заздалегідь визначеної множини варіантів, така задача має*недетермінований* розв'язок [1]. Алгоритмнедетермінований, якщо для його реалізації використовуються методи проб і помилок, повторів або випадкового вибору. До подібних задач належать такі, як знаходження дільників числа, пошук сторінки, а також класичні задачі про комівояжера, про вісім ферзів і задачі знаходження найкоротшого шляху через лабіринт.

Третій, основний тип алгоритмів, призначений не для пошуку відповіді на поставлену задачу, а для моделювання фізичних систем з використанням комп‘ютера.

2.4. Документація алгоритмів

Документація алгоритмів є частиною документації програм і модулів. Якщо використовуваний алгоритм добре відомий або його опис можна знайти в спеціальному джерелі, документація повинна містити ім'я алгоритму або посилання на джерело і авторів. Так, наприклад, для розв'язку задачі про комівояжера можна скористатисяалгоритмом Дейкстри. Для знаходження квадратного кореня в основному використовується метод Ньютона - Рафсона, а для обчислення sin(x) - поліноми Чебишева. Упорядкування списків виконується за допомогою сортування Шелла, а також різних видів бульбашкового сортування.

Якщо в алгоритмі використаний спеціальний математичний апарат (як, наприклад, у випадку біноміальних коефіцієнтів), у документацію повинне бути включене математичне обґрунтування методу.

Якщо для спрощення задачі або збільшення швидкості розрахунку її на комп‘ютері застосовуються які-небудь евристичні методи, вони повинні бути зазначені в документації. У ній повинен бути включений загальний опис використовуваного методу й підходи до застосовуваної моделі, якщо це допоможе краще зрозуміти алгоритм. Також повинні бути зазначені особливості реалізаціїалгоритму у випадку застосування рекурсії або паралельної обробки. Добре відоміалгоритми не мають потреби в детальному описі. Якщо ж використовуються менш відомі алгоритми або вони були розроблені програмістом, описи повинні бути складені докладно, можливо із застосуванням псевдокодів.