УДК 004.422.5

Житник Є. Т., здобувачка 2 курсу спеціальності
122 Комп'ютерні науки,
Фриз І. В., канд. фіз.-мат. наук,
старший викладач кафедри інформаційних технологій

ОПТИМІЗАЦІЯ АЛГОРИТМІВ: МЕТОДИ, ПІДХОДИ ТА ЇХ ПОРІВНЯННЯ

Донецький національний університет імені Василя Стуса, м. Вінниця

Оптимізація алгоритмів – це процес удосконалення алгоритмічних рішень з метою зменшення витрат ресурсів під час їх виконання. Адекватний аналіз ефективності алгоритму є критично важливим етапом розробки: правильний вибір алгоритму часто має вирішальне значення, адже навіть «теоретично оптимальний» алгоритм може виявитися на практиці надто повільним. Аналіз алгоритмів передбачає оцінку часових та пам'ятних витрат як функції розміру вхідних даних. Метою оптимізації є змінити алгоритм чи структури даних так, щоб зменшити цю складність чи покращити константи за тієї ж асимптики [1].

Визначають 3 підходи до оптимізації:

1. Зменшення асимптотичної складності. Часто це досягається застосуванням кращих алгоритмічних технік або структур даних. Наприклад, заміна квадратичного алгоритму на лінійно-логарифмічний $O(n^2) \to O(n \log n)$ дає різке підвищення продуктивності за великих n [1]. Додатково, важливою є мінімізація використання пам'яті (оптимізація пам'яті).

2. Переходи від рекурсії до ітерації. Рекурсивні алгоритми витрачають час на виклики функцій і виділення стекових кадрів. Часто можна переписати рекурсивні процедури у вигляді ітераційних циклів, що усуває накладні витрати викликів. Динамічне програмування демонструє цю ідею: рекурсивний алгоритм із мемоїзацією можна перетворити на еквівалентний bottom-up варіант. Водночас підхід «знизу вгору» працює швидше, бо відсутній накладний час рекурсії [2]. За такого переписування принцип роботи алгоритму не змінюється, але константи під час виконання коду значно зменшуються.

3. Мемоїзації і кешування результатів дає змогу уникнути повторних обчислень. Динамічне програмування – класичний приклад такого підходу: алгоритми зберігають проміжні результати в таблиці, щоб під час повторного виникнення підзадачі не виконувати обчислення заново. Динамічне програмування використовує властивість перекриття підзадач – кожну підзадачу обчислюють лише раз і потім повертають із таблиці за $O(1)$-час [2]. Так, мемоїзований (рекурсивний) та ітеративний DP-алгоритм для задачі матричного перемноження обчислюють оптимальну вартість за $O(n^3)$ однаково асимптотично, але ітераційний варіант працює швидше завдяки відсутності накладних витрат на рекурсію [2]. Порівняння методів оптимізації алгоритмів наведено у табл. 1.

Таблиця 1 – Порівняння методів оптимізації алгоритмів

Метод	Сутність	Застосування	Переваги	Недоліки
Хешування	Перетворення даних у хеш-код для швидкого доступу	Пошук, перевірка приналежності, кешування	Дуже висока швидкість ($O(1)$). Ефективне використання пам'яті за гарного хешу	Колізії. Залежність від хорошої хеш-функції
Ефективність структур даних	Підбір структури з оптимальними операціями	Сортування, пошук, маршрути, обчислення пріоритетів	Зменшення складності алгоритмів. Гнучкість	Необхідність глибокого розуміння структур. Складність реалізації
«Розділяй і володарюй»	Рекурсивний поділ задачі на дрібніші підзадачі	Сортування (MergeSort, QuickSort), пошук, обчислення	Значне прискорення обчислень. Природна паралелізація	Залежність від рекурсії. Накладні витрати на злиття та поділ
Динамічне програмування	Запам'ятовування вже обчислених результатів	Оптимізаційні задачі, графи, комбінаторика	Максимальне скорочення повторних обчислень. Мінімізує складність експоненційних задач	Вимагає багато пам'яті. Потрібно правильно визначити підзадачі
Евристики та жадібні алгоритми	Вибір часткового найкращого рішення на кожному кроці	Задачі маршрутизації, наближених рішень, великий обсяг даних	Дуже швидкі. Підходять для великих задач	Не гарантують оптимального результату. Можуть застрягати в локальних мінімумах

Оптимізація алгоритму може здійснюватися різноманітними техніками:

1. Хешування прискорює пошук у великих множинах даних: хеш-таблиця дає змогу виконувати операції вставки чи пошуку в амортизований час $O(1)$. Як зазначено у підручнику, за допомогою хешування можна побудувати структуру даних із постійним часом пошуку [1]. Використання добре спроєктованої хеш-функції та механізму розв'язання колізій забезпечує швидкий доступ до елементів.

2. Підбір ефективних структур даних – наприклад, збалансованих дерев, бінарних куп та пріоритетних черг, дає змогу оптимізувати операції вставки / видалення / пошуку. Наприклад, пріоритетна черга на основі кучи дає змогу за $O(\log n)$ дістати мінімальний елемент, тоді як із простої лінійної черги цей час був би $O(1)$ лише для вставки, але $O(n)$ для вибору мінімального.

3. «Розділяй і володарюй». Розбиття задачі на підзадачі зазвичай зменшує складність. Класичний приклад – алгоритм сортування злиттям: він розбиває масив навпіл, сортує кожну половину й потім об'єднує результати за лінійний час. Завдяки такому розбиттю та конструюванню рішення за допомогою двох часткових розв'язків, злиття забезпечує асимптотичну швидкість $O(n \log n)$. Інші приклади D&C – швидке сортування, бінарний пошук тощо – зазвичай мають кращу складність, порівняно з наївними алгоритмами.

4. Динамічне програмування та мемоїзація для задач із оптимальною підструктурою. Якщо задача дає змогу розбити її на перекриваючі підзадачі з властивістю optimal substructure, DP-алгоритм зберігає результати підзадач у таблиці,

що допомагає уникнути повторних обчислень [2]. Так складність може зменшитися з експоненційної до поліноміальної.

5. Жадібні алгоритми та евристики. Жадібні алгоритми приймають локально оптимальний вибір у кожному кроці, сподіваючись, що це приведе до глобально оптимального рішення. Евристичні підходи застосовують наближені методи, щоб швидко отримати достатньо хороший розв'язок, коли екзогенно оптимальний алгоритм є занадто повільним. Такі методи особливо важливі для задач NP-складності, де точний розв'язок може вимагати надто тривалого часу.

Отже, кожен із розглянутих методів оптимізації виконує унікальну функцію та застосовується для різних типів алгоритмічних задач і вибір конкретного методу оптимізації залежить від структури задачі, обмежень ресурсів та вимог до точності. Найвищу ефективність забезпечує комбінація декількох підходів, що дає змогу збалансувати швидкість, складність реалізації й кінцеву якість результату.

Список використаних джерел

1. Креневич А. П. Алгоритми і структури даних: підручник. Київ: ВПЦ «Київський Університет», 2021. 200 с.

2. Introduction to algorithms / T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. 4th ed. Cambridge, MA: The MIT Press, 2022. 1312 p. URL: https://books.google.com.ua/books?id=NLngYyWFl_YC&printsec=copyright&hl=uk#v=onepage&q&f=false (дата звернення: 15.11.2025).

3. Skiena S. S. The algorithm design manual. 3rd ed. Cham: Springer, 2020. 810 p.

УДК 004.422.5

Горчук К. В., здобувач 2 курсу
спеціальності 122 Комп'ютерні науки,
Потапова Н. А., канд. екон. наук, доцент,
доцент кафедри інформаційних технологій

ОСОБЛИВОСТІ ТА ПЕРЕВАГИ ВИКОРИСТАННЯ АЛГОРИТМУ БІНАРНОГО ПОШУКУ

Донецький національний університет імені Василя Стуса, м. Вінниця

Бінарний пошук – це алгоритм пошуку індексу елементу у впорядкованому масиві. На кожній ітерації відбувається поділ масиву на дві частини, через що алгоритм називають методом ділення навпіл. Цей спосіб пошуку даних достатньо простий для розуміння, водночас він дуже ефективний, оскільки на кожному кроці кількість елементів у робочій області масиву зменшується вдвічі [1].

Алгоритм бінарного пошуку полягає у визначенні значення елементу посередині робочої області масиву даних і порівнянні його із шуканим. У разі рівності елементів повертається індекс, а якщо значення елементу посередині масиву більше від шуканого, то пошук продовжується в лівій частині масиву, якщо інакше – в