УДК 004.421.2

ПОРІВНЯННЯ НАЙВІДОМІШИХ АЛГОРИТМІВ ПОШУКУ

Р. Д. Матвійчук, О. М. Данильчук

Анотація. У даному дослідженні подана інформація про відомі алгоритми пошуку, їхню реалізацію, значення в різних сферах та порівняння ефективності цих алгоритмів. Методологічною основою роботи є наукова теорія пізнання основ поняття та аналізу властивостей алгоритму пошуку. Специфіка досліджуваної теми передбачає застосування прикладних знань в сфері інформаційних технологій та знання мов програмування для перевірки ефективності поданих у роботі алгоритмів пошуку.

Ключові слова: пошук, алгоритм, масив, функція, програма.

Навіть у звичайному житті іноді потрібно знайти конкретний елемент серед величезної кількості даних. Кожного дня ми шукаємо потрібну інформацію в глобальній мережі Інтернет. Цей процес займає лічені хвилини чи навіть секунди, але яким же чином ми так швидко знаходимо необхідні дані? Саме за допомогою алгоритмів пошуку, які постійно розвиваються та удосконалюються. Без них нам довелося би переглядати кожний елемент, кожний номер телефону чи адресу компанії. У великому наборі даних це займало би багато часу, тому розвиток алгоритмів пошуку дуже спрощує щоденне життя звичайних людей, а для фахівців, які працюють з базами даних гігантських масштабів, алгоритми є незамінними помічниками. Особливо важливу роль вони відіграють в IT сфері, оскільки допомагають розробникам програмного забезпечення створювати ефективні програми без помилок. Будь-яка програма чи програмне забезпечення зберігає величезну кількість даних. Щоб отримати доступ до них комп’ютеру потрібно для початку знайти їх, але, якщо програма містить величезну кількість даних, то лінійний пошук може зайняти багато часу, тому використовують альтернативні алгоритми пошуку. Найважливіше, що потрібно пам’ятати про них, це те, що для однієї і тієї ж задачі може бути багато різних алгоритмів, але деякі з них можуть бути ефективнішими за інші.

Найпростіший алгоритм для пошуку інформації – це лінійний (послідовний) алгоритм. Він полягає в тому, що ви переглядаєте весь список послідовно і намагаєтеся знайти відповідність для заданого елементу. Якщо знаходите, то програма повертає адресу відповідного елемента, якщо ні, то повертається нульове значення (NULL). Реалізацію даного алгоритму можна продемонструвати за допомогою мови програмування C++. Нехай задано рядок «Beautiful world», в якому потрібно знайти елемент ‘d’. Для пошуку використаємо програму (рис. 1).

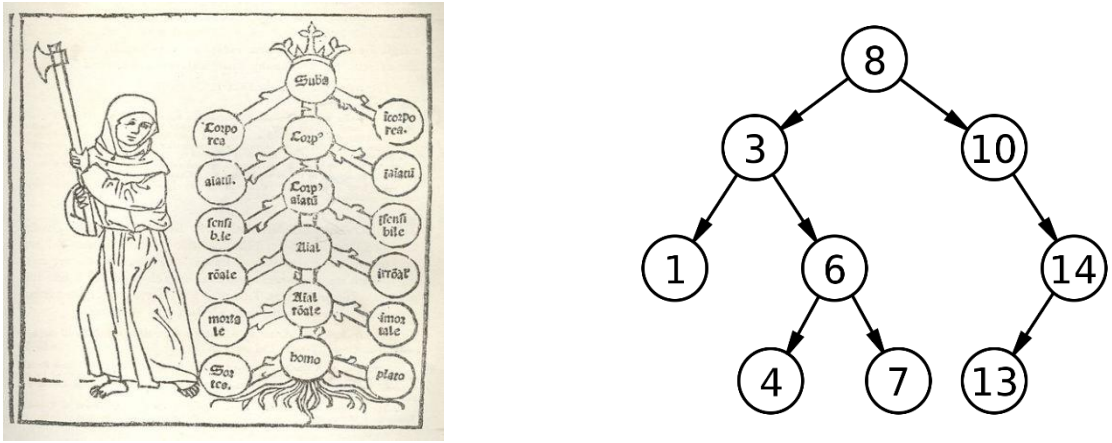
```
#include <iostream> //для виводу на екран
#include <string.h> //функція strlen
using namespace std; //використовуємо простір імен std

void FindPos(int pos) //оголошена функція обробки входження з параметром позиції
{
    cout << "Find position: " << pos << endl; //вивід позиції шуканого елемента
}

void SearchAlgorithm(char* str, //масив даних у якому відбуватиметься пошук
char c, /*шуканий елемент масиву*/ void (*FindPos) (int) /*функція обробки входження*/){

     int N = strlen(str); //дізнаємося довжину масиву

for (int i = 0; i < N; ++i) //цикл пошуку
{
    if (str[i] == c) //перевірка елементу
    {
        FindPos(i); //якщо справджується умова - виконати обробку
    }
}
}

const char* phrase = "Beautiful world"; //створений і заповнений масив

int main(int argc, char** argv) //головна функція програми
{
    char* str = (char*)phrase;
    char c = 'd'; //перевірка елементу

    SearchAlgorithm(str, c, &FindPos); //передаємо управління функції пошуку

    return 0; //вихід з програми
}
```

Рис. 1. Реалізація лінійного пошуку на мові C++

Рис. 2. Результат виконання програми

Програма послідовно проходиться по рядку, порівнюючи кожний елемент з шуканим. Для знаходження заданої літери програмі знадобилося 14 ітерацій, адже вона була останньою (рис. 2). Зрозуміло, чому найгірший випадок продуктивності алгоритму O(n), адже для знаходження елементу потрібно стільки ітерацій, який порядковий номер елементу у масиві, а, як наприклад в нашому випадку, цей елемент може бути останнім. Найшвидший варіант – O(1), коли шуканий елемент є першим, а в середньому для знаходження елементу потрібно $(n+1)/2$ ітерацій. Лінійний пошук доволі простий у реалізації, але тоді, коли масив містить лише декілька елементів, або коли масив є невпорядкований. Як результат, хоча теоретично інші алгоритми пошуку можуть бути швидшими, ніж лінійний пошук, наприклад бінарний, на практиці навіть на масивах середнього розміру (близько 100 елементів або менше) може бути неможливим використовувати щось інше. У більших масивах має сенс використовувати швидші методи пошуку лише, якщо дані досить великі, оскільки початковий час підготовки (сортування) даних порівняно більший.

Попередній варіант пошуку часто порівнюють з бінарним пошуком. Бінарний (двійковий) пошук – це алгоритм пошуку, який використовується в відсортованому масиві шляхом багаторазового ділення інтервалу навпіл. Його суть полягає в тому, щоб використовувати інформацію про те, що масив є відсортованим та зменшити складність алгоритму (в порівнянні з лінійним). Вперше він згадується Джоном Моклі в 1946 році в його праці «Теорія та методика проектування електронних цифрових комп'ютерів». Його робота можливо була першим опублікованим обговоренням методів нечисельного програмування. Хоча не можна достовірно знати, коли алгоритми пошуку були винайдені, адже навіть у роботах Платона та Аристотеля можна знайти побудови класифікацій, які були засновані на бінарних дихотоміях (розбиття цілого (чи множини) на дві частини (підмножини)), що пізніше були зображені як бінарні порфірові дерева (рис. 3а) і перевтіленні в інформатиці у відсортовані бінарні дерева (рис. 3б), які використовуються для бінарного пошуку.



а б

Рис. 3: а) Порфірове дерево, б) Бінарне дерево

Реалізацію даного алгоритму також можна продемонструвати за допомогою мови програмування C++. Нехай маємо відсортований масив цілих чисел, в якому потрібно знайти число 15. Для пошуку використаємо програму (рис. 4).

```cpp
#include <iostream>
#include <string.h>
using namespace std;

int main(int argc, char** argv)
{
    //відсортований масив даних
    int M[] = { 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 };

    int N = sizeof(M) / sizeof(int); //визначаємо розмір масиву
    int L = 0;  // ліва межа
    int R = N - 1;  // права межа - обрахунок починається з нуля
    int m = 0;  // змінна збереження середини відрізку
    int x = 15;  // шуканий елемент
    int counter = 0;  // позиція елементу

    do //цикл
    {
        m = R - ((R - L) / 2); //визначаємо середину відрізка

        if (M[m] < x)
        {
            L = m + 1;
        } //посуваємо праву межу вліво від елементу масиву
        else
        {
            R = m - 1;
        } // посуваємо ліву межу вправо від елементу масиву

        counter++; // переходимо на наступну позицію
    } while ((L <= R) && (M[m] != x));

    // виводимо кількість ітерацій, яка знадобилася для знаходження елементу
    cout << "Count of iterations:" << counter << endl;

    // виводимо позицію, на якій знаходиться шуканий елемент
    cout << "Find position " << m << endl;
    return 0;
}
```

Рис. 4. Реалізація бінарного пошуку на мові C++

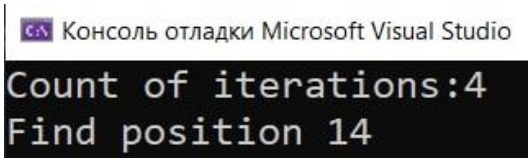


Рис. 5. Результат виконання програми

Схожа ситуація, як із задачею для лінійного пошуку, шуканий елемент є останнім, але у випадку з бінарним пошуком нам знадобилося лише 4 кроки (рис. 5) для того, щоб знайти його. Складність алгоритму – O ($\log_2 n$). Для порівняння при десяти елементів масиву, щоб знайти потрібний елемент за допомогою лінійного пошуку, знадобиться найбільше – це 10 ітерацій, а за допомогою бінарного пошуку близько 3–4 ітерацій.

Бінарний пошук знайшов застосування в математиці та інформатиці. Але його можна застосовувати лише у тому випадку, якщо елементи масиву впорядковано (відсортовано), тому цей алгоритм пошуку не використовується для знаходження максимального або мінімального елементу, так як у відсортованому масиві ці елементи містяться на початку та в кінці масиву відповідно, залежно від того, як відсортований масив, за зростанням чи спаданням. Тому хоча бінарний пошук ефективніше лінійного, але його не завжди можна застосувати.

Пошук за допомогою золотого перерізу – це покращення реалізації потрійного пошуку, що служить для знаходження мінімуму та максимуму функції. При простому потрійному пошуку кожної ітерації функція обчислюється у двох точках (рис. 6). Метод золотого перерізу вимагає обчислення лише в одній точці (за винятком першої ітерації). Метод був продемонстрований у 1953 році Джеком Кіфером.

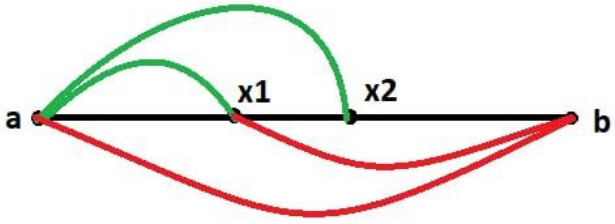


Рис. 6. Схема алгоритму методу золотого перерізу

В теорії метод виглядає так: нехай маємо функцію $f(x)$ та відрізок $[a; b]$, на якому потрібно знайти максимальне значення. Відрізок ділиться в двох напрямках точками x_1 та x_2 у відношенні золотого перерізу. Тобто $\frac{b-a}{b-x_1} = \frac{b-a}{x_2-a} = \varphi$, де φ – це пропорція золотого перерізу. Отже, координати x_1 та x_2 можемо знайти за формулами $x_1 = b - \frac{b-a}{\varphi}$, $x_2 = a + \frac{b-a}{\varphi}$.

Застосування методу золотого перетину досить широке, адже воно зустрічається в різних сферах. Його застосовують в фотографії, в живописі, в веб-розробці, в архітектурі, а також в програмуванні. Але метод має свої недоліки, наприклад обов'язковим є те, що функція має бути унімодальною (дійсна функція $f(x)$, якщо існує єдина точка мінімуму x^*, і що $f(x)$ строго спадає для $x \leq x^*$ і строго зростає для $x \geq x^*$) в межах заданого інтервалу невизначеності. Адже, якщо функція не унімодальна, а мультимодальна (функція більш ніж з однією «модою» або оптимумом), тоді існує кілька максимумів, а локальні максимуми та мінімуми знаходяться в межах інтервалу невизначеності, і потрібно зменшити заданий інтервал для використання методу. Після зменшення ми можемо застосовувати метод золотого перетину для отримання оптимального рішення чи то знаходження локального максимуму, чи мінімуму, а потім вже отримати глобальне рішення, виходячи з локальних даних.

Метод золотого перерізу схожий ще на один відомий метод – метод Фібоначчі – метод пошуку відсортованого масиву за допомогою алгоритму «розділяй та владарюй», який звужує можливі місця за допомогою чисел Фібоначчі. Його використовують не часто, адже для нього потрібно завчасно задати число обчислень значень функцій. І метод золотого перерізу, і метод Фібоначчі найбільше підходять для вирішення одновимірних унімодальних задач для оптимізації. Але метод Фібоначчі вважається досить повільним, адже в ньому виникають проблеми пов'язані з рекурсією, тому деякі обчислення повторюються по декілька разів. Навіть для невеликих чисел, таких як 100, рекурсивний алгоритм не дасть швидкий результат. Пояснення такої повільної роботи представлено на діаграмі (рис. 7).

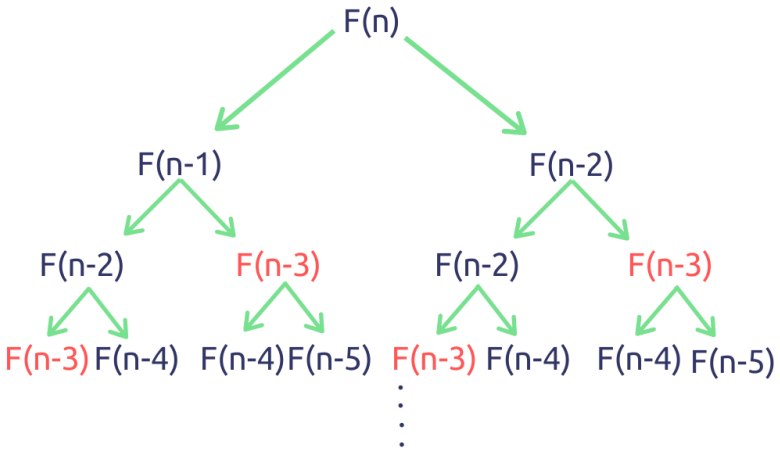


Рис. 7. Схема методу Фібоначчі

Якщо порівнювати згадані вище методи, то при зіставленні бінарного пошуку з алгоритмом пошуку Фібоначчі можемо бачити, що в першому методі відсортований масив ділиться на дві однакові за розміром частини, одна з яких розглядається далі, а в другому – алгоритм ділить масив на дві частини, розмірами яких є числа Фібоначчі. Така різниця у цих методах в середньому дає на 4 % більшу кількість порівнянь у методі Фібоначчі, але його перевага в тому, що для обчислення індексів елементів масиву потрібно лише дії додавання та віднімання. В той же час бінарний пошук, хоча і не потребує такої кількості порівнянь, але метод потребує бітового зсуву, ділення або множення. Ряд Фібоначчі має таку властивість, що кожне його число є сумою двох попередників, тому його можна легко обчислити шляхом повторного додавання. Якщо обчислити співвідношення двох його послідовних чисел, то помітимо, що значення близьке до «золотого перетину» близько 1.618… Як було згадано вище, бінарний пошук ділить масив навпіл, тобто у відношенні 1:1,

а пошук Фібоначчі може розділити 1:1.618, використовуючи простіші операції. Пошук Фібоначчі може мати перевагу перед бінарним пошуком, дещо зменшивши середній час, необхідний для доступу до місця зберігання. Також варто зазначити, що даний пошук має середню та найвищу складність O ($\log_2 n$).

На сьогоднішній день алгоритми застосовуються в багатьох сферах, адже їх розвиток досягнув величезних масштабів. Не можна точно сказати, що один алгоритм буде ефективніше іншого, поки перед нами не стоїть умова задачі, яку потрібно вирішити. Для різних типів прикладних задач, для різних масивів та сфер даних підходять різні алгоритми пошуку. Найважливіше вміти грамотно використовувати знання про них.

Abstract. In this study presents the information about famous searching algorithms, their implementation, meanings in different spaces and comparison of efficiency. The methodological basis of the work is the scientific theory of cognition of the basics of the concept and the analysis of the properties of the search algorithm. Specificity of research topic envisages using of applied knowledge in IT sector and knowledge of programming languages for assesses the effectiveness of searching algorithms.

Keywords: searching, algorithm, array, function, program.

СПИСОК ЛІТЕРАТУРИ

1. Kagan E. and Ben-Gal I. A Group-Testing Algorithm with Online Informational Learning. 2014. 46:2. P. 164–184.
2. Ali Yalcin, Autar Kaw. Textbook notes for the golden search method. 2021.
3. URL: http://numericalmethods.eng.usf.edu
4. Chang, Shi-Kuo. Data structures and algorithms. *Software Engineering and Knowledge Engineering*. Vol. 13. Singapore: World Scientific. 2003.
5. URL: https://www.geeksforgeeks.org/binary-search/

УДК 378.015.31:174.7.

ОСОБЛИВОСТІ ДОТРИМАННЯ ПРИНЦИПІВ АКАДЕМІЧНОЇ ДОБРОЧЕСНОСТІ: ЗАРУБІЖНИЙ ДОСВІД

А. В. Недипіч, В. Ю. Василенко

Анотація. У даному дослідженні подана інформація про світові тенденції дотримання академічної доброчесності у таких країнах, як: Франція, Сполучені Штати Америки (США), Польща, Швеція. Визначено, що окрім фундаментальних нормативних документів зарубіжні заклади вищої освіти (ЗВО) поширюють та популяризують культуру академічної доброчесності, зокрема, за допомогою консультацій, діалогів, вебінарів, курсів, окремих навчальних компонентів в освітніх програмах здобувачів. Основними методами, які були використані у процесі дослідження, стали: теоретичний аналіз наукової та методичної літератури, порівняння, пояснювальний, методи пізнання та узагальнення.

Ключові слова: зарубіжний досвід, академічна доброчесність, плагіат.

Вступ. Дотримання принципів академічної доброчесності в освітньо-науковому середовищі є вкрай важливим та актуальним питанням сьогодення. Заклади, які реалізують освітню діяльність, знаходяться у постійному пошуку та реалізації механізмів та можливих способів протидії порушенням академічної доброчесності. У зв’язку з цим при організації досліджень у сфері академічної доброчесності в Україні доречно спиратися на доробок інших країн світу.

Основний розділ. Особливості реалізації академічної доброчесності у **Франції.**

Особливості розвитку академічної доброчесності в університетах Франції можна згрупувати за ознаками:

- на законодавчому рівні – університети Європи враховують академічну доброчесність як складову якості освітнього процесу, що регламентується в Бухарестській декларації етичних цінностей і принципів вищої освіти в Європі та різними міжнародними проектами;