**ЛАБОРАТОРНА РОБОТА №6. «МЕТОДИ ПОШУКУ»**

Мета роботи: Виконати аналіз емпіричної ефективності (практичної складності) методів пошуку: послідовного, швидкого послідовного, бінарного.

**Завдання на лабораторну роботу**

1) Вивчення послідовного, швидкого послідовного та бінарного методів пошуку.

2) Програмна реалізація цих методів пошуку.

3) Генерація вихідних даних відповідно до індивідуального завдання.

4) Проведення обчислювальних експериментів за допомогою розробленої програми з метою виміру часу роботи кожного методу.

5) Побудова графіків залежностей часу роботи методу пошуку від розміру вихідної послідовності. Графіки повинні включати залежності для вдалого та невдалого пошуку.

6) Порівняльний аналіз одержаних результатів.

**Теоретичні відомості**

Пошук – це одна з найпоширеніших у програмуванні дій.

Існує безліч варіацій задачі пошуку. У цій роботі розглянемо тільки пошуку у фіксованій послідовності елементів. На вхід алгоритму пошуку подається послідовність n елементів і деякий елемент. Завдання полягає в тому, щоб отримати на виході індекс i-го елемента послідовності, що дорівнює заданому елементу Key, або з'ясувати, що такого елемента в даній послідовності не існує.

1. Послідовний пошук

Найпростішим алгоритмом для вирішення поставленого завдання є послідовний чи лінійний пошук. Він полягає в тому, що по черзі порівнюється шуканий елемент Key із елементами вихідної послідовності. Якщо буде виявлено збіг, повертається індекс знайденого елемента. Якщо після закінчення порівняння з елементами послідовності шуканий елемент Key не знайдено, повертається інформація про це, наприклад, -1. Псевдокод даного алгоритму має такий вигляд:

SequentialSearch(A[1..n],b)

// Input data: array A[1..n], element b

//Output data: index of element array A,

// coincident with element Key,

// else -1, if this element not find

i=1

while i <= n and A[i] ≠ Key do i = i+1

if A[i] = Key then return i

else return n

Очевидно, що при пошуку елемента в масиві доводиться перевірити в середньому половину елементів масиву у разі успішного пошуку, і весь масив – у разі невдалого. В обох випадках часова ефективність дорівнює О(n).

Оформимо описаний алгоритм як фрагмента програми мовою С++. При цьому не важливим є тип елементів послідовності, а важлива лише перевірка рівності елемента вихідної послідовності шуканому елементу Key.

i=1;

while ((i<=n)&&(A[i]!=Key)) i++;

if (A[i]==Key) <елемент знайдений>;

else <елемент не знайдений>;

У середньому цей алгоритм потребує n/2 ітерацій.

2. Швидкий послідовний пошук

Цей спосіб пошуку є невеликим удосконаленням попереднього. У будь-якій програмі, що має цикли, найбільший інтерес представляє оптимізація саме циклів, тобто скорочення кількості дій у них.

Порівняємо алгоритм лінійного пошуку (програму розділу 1). У циклі while проводиться два порівняння: (i<=n) та (A[i] != Key). Позбавимося одного з них (від першого), поклавши A[n+1] = Key. Тоді програма пошуку буде виглядати так:

A[n+1] = Key;

i = 1;

while (A[i] != Key) i++;

if (i<=n) < element found >;

else < element not found >;

Даний фрагмент коду працюватиме швидше, але його часова ефективність залишається такою самою – О(n). Набагато більший інтерес представляють методи, які не тільки працюють швидко, але й дають високу ефективність. Один із таких методів – дихотомічний пошук.

3. Бінарний (дихотомічний) пошук

Цей метод пошуку припускає, що множина зберігається як деяка відсортована (наприклад, за зростанням) послідовність елементів, для яких можна отримати прямий доступ у вигляді індексу. Таким чином, множина зберігається в масиві та цей масив відсортовано.

Областю пошуку (l, r) назвемо частину масиву з індексами від l до r, в якій імовірно знаходиться елемент, що шукається. Спочатку областю пошуку буде частина масиву (l, r), де l = 1, а r = n, тобто вся заповнена елементами множини частина масиву. Тепер знайдемо індекс середнього елемента шляхом цілісних поділів

int m = (l+r)/2.

Якщо Key>A[m], можна стверджувати (оскільки масив відсортований), що якщо Key є в масиві, він знаходиться в одному з елементів з індексами від m + l до r, отже, можна присвоїти l = m+1, скоротивши область пошуку. В іншому випадку можна покласти r = m. На цьому закінчується перший крок методу. Інші кроки аналогічні.

На кожному кроці методу область пошуку скорочуватиметься вдвічі. Як тільки l стане r, тобто область пошуку скоротиться до одного елемента, можна буде перевірити цей елемент на рівність шуканому і зробити висновок про результат пошуку.

Фрагмент алгоритму бінарного пошуку представлений нижче.

BinarySearch(A[1..n],Key)

// Input data: array A[1..n]order by ascending, element Key

// Output data: index of element array A,

// coincident with element Key,

// else -1, if this element not find

l=1;

r=n;

while (l!=r)

{

m=(l+r)/2;

if (Key>A[m]) l=m+1;

else r=m;

}

if (A[l]==Key) <елемент знайдений>;

else <елемент не знайдений>;

Область пошуку на кожному кроці скорочується вдвічі, а це означає складність T(log(n))

4. Методичні вказівки щодо виконання роботи

Для виконання роботи необхідно:

1. Побудувати блок-схему алгоритмів пошуку.

2. Виконати програмну реалізацію алгоритмів пошуку. Правильність роботи програм підтвердити скріншотами.

3. Згенерувати вихідні послідовності залежно від заданого розміру вибірки та зберегти у файлі.

− Для генерації вихідної числової послідовності можна використовувати датчик випадкових чисел.

– Для чистоти експерименту дані мають бути з однієї згенерованої вибірки, а також повинен здійснюватися пошук одного й того самого елемента, наприклад останнього елемента вибірки чи середнього.

4. Виконати експерименти, під час яких визначити час роботи кожного алгоритму залежно від розміру вибірки, використовуючи функцію clock(), яка описана в заголовному файлі <time.h> у такому вигляді:

clock\_t clock (void);

Тип даних clock\_t визначається у файлі time.h. Функція clock\_t повертає процесорний час у одиницях, які залежать від мови. (Якщо процесорний час недоступний або не може бути представлено, функція повертає значення -1.)

Наприклад, якщо необхідно отримати час виконання будь-яких дій, то потрібно вставити в програму наступний код:

#include <time.h >

......

clock\_t time;

time = clock();

/\* якісь дії \*/

time = clock() - time;

cout << (double)time; // час виконання якихось дій

Результати експериментів для вдалого та невдалого пошуків представити у вигляді таблиці

Таблиця 1 – Результати вдалого пошуку

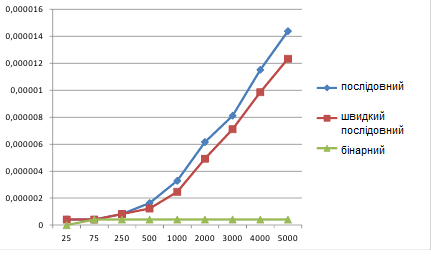
|  |  |  |  |
| --- | --- | --- | --- |
| Обсяг вибірки,  N | Час | | |
| При послідовному пошуку | При швидкому послідовному пошуку | При бінарному пошуку |
| 100 |  |  |  |
| 500 |  |  |  |
| 1000 |  |  |  |
| 2500 |  |  |  |
| 5000 |  |  |  |
| 10000 |  |  |  |
| 50000 |  |  |  |
| 100000 |  |  |  |

Таблиця 2 – Результати невдалого пошуку

|  |  |  |  |
| --- | --- | --- | --- |
| Обсяг вибірки,  N | Час | | |
| При послідовному пошуку | При швидкому послідовному пошуку | При бінарному пошуку |
| 100 |  |  |  |
| 500 |  |  |  |
| 1000 |  |  |  |
| 2500 |  |  |  |
| 5000 |  |  |  |
| 10000 |  |  |  |
| 50000 |  |  |  |
| 100000 |  |  |  |

5. Побудувати графіки залежностей часу роботи алгоритмів від обсягу вибірки для вдалого та невдалого пошуків

Приклад графіка



Додаткове завдання (для отримання максимального бала).

Номер задачі k обирається за формулою

k=mod(N/12)+1 – номер

mod2 – залишок від ділення

N – номер за списком

Задача 1.

Знайти другий найбільший елемент в послідовності

Задача 2

Перевірити, чи є в масиві елементи, що повторюються

Задача 3

Знайти елемент, який з'являється в послідовності найбільше число разів

Задача 4

Заданий масив цілих чисел від 1 до n. Знайти пропущене число

(Наприклад, вх.масив [1,2,3,4,6,7,8,9] – пропущене число 5

Задача 5

Задано масив з n чисел.

Знайти два елементи в масиві, чия сума дорівнює заданому числу К

Задача 6

Заданий масив з цілих додатних та від'ємних чисел. Знайти такі два елементи, чия сума є найближчою до 0.

Задача 7

Заданий масив, що складається з 1 на початку та 0 в кінці. Знайти індекс першого 0

Задача 8.

Заданий масив, що складається з цілих чисел, які можуть повторюватись. Знайти індекс останнього входження заданого числа.

Задача 9

Елемент є більшістю, якщо він з’являється більше ніж n/2 разів. В заданому масиві знайти більшість (якщо вона існує)

Задача 10

Заданий масив з 101 елемента. З них 25 елементів зустрічається двічі, 12 елементів 4 рази, один елемент зустрічається 3 рази. Знайти елемент. Який зустрічається тричі.

Задача 11

Заданий масив А, що містить цілі числа з інтервалу від Х до Y. В масиві є один елемент, який знаходиться за межами цього діапазону. Знайти даний елемент.

Задача 12

Задано 3 відсортовані масиви. Знайти всі спільні елементи в них