Міністерство освіти і науки України

Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"

Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

**Звіт**

з лабораторної роботи № 7 з дисципліни

Алгоритми та структури даних 2. Структури даних»

«Проектування і аналіз алгоритмів пошуку»

Варіант **3**

**Виконав студент:** ІП-з21 Гавриленко Даяна Юріївна

**Перевірив:** Халус Олена Андріївна

Київ 2023

**7.1 Мета роботи**  
 Мета роботи – вивчити основні підходи аналізу обчислювальної складності алгоритмів пошуку оцінити їх ефективність на різних структурах даних.

**7.2 Постановка задачі**

Алгоритм пошуку: Пошук Фібоначчі

**7.3 Псевдокод алгоритму**

Запитати користувача, чи використовувати масив чи двозв'язний список (ввести 'a' або 'l')

Запитати користувача про розмір структури (n)

Створити порожній масив або двозв'язний список (залежно від вибору користувача)

Якщо користувач обрав масив:

Заповнити масив випадковими числами

Сортувати масив

Інакше, якщо користувач обрав двозв'язний список:

Заповнити список випадковими числами

Створити новий впорядкований список на основі першого списку

Знайти індекс елемента з заданим ключем, використовуючи пошук Фібоначчі

Встановити змінні для порівнянь, доступів до масиву або списку

Запустити таймер

Якщо користувач обрав масив:

Знайти індекс за допомогою пошуку Фібоначчі у масиві, записати кількість порівнянь і доступів до масиву

Інакше:

Знайти індекс за допомогою пошуку Фібоначчі у списку, записати кількість порівнянь і доступів до списку

Зупинити таймер

Вивести кількість порівнянь, доступів до масиву або списку

Вивести знайдений індекс елемента з ключем

Вивести час, що пройшов у тактах

Функція пошуку Фібоначчі у масиві:

Ініціалізувати числа Фібоначчі

Знайти найменше число Фібоначчі, яке більше або рівне довжині масиву

Ініціалізувати змінну зсуву

Поки є елементи для перевірки:

Перевірити, чи елемент менше ключа

Якщо так, зменшити діапазон пошуку

Перевірити, чи елемент більше ключа

Якщо так, зменшити діапазон пошуку

Якщо елемент дорівнює ключу, повернути індекс елемента

Порівняти останній елемент з ключем

Якщо так, повернути індекс останнього елемента

Повернути -1

Функція пошуку Фібоначчі у списку:

Ініціалізувати числа Фібоначчі

Знайти найменше число Фібоначчі, яке більше або рівне довжині списку

Ініціалізувати змінну зсуву

Поки є елементи для перевірки:

Отримати значення елемента за індексом

Перевірити, чи елемент менше ключа

Якщо так, зменшити діапазон пошуку

Перевірити, чи елемент більше ключа

Якщо так, зменшити діапазон пошуку

Якщо елемент дорівнює ключу, повернути індекс елемента

Отримати значення останнього елемента

Порівняти останній елемент з ключем

Якщо так, повернути індекс останнього елемента

Повернути -1

Функція для отримання значення за індексом у списку:

Ініціалізувати лічильник доступів

Знайти початковий вузол

Пройтися по списку до вузла з вказаним індексом

Збільшити лічильник доступів

Повернути значення вузла і лічильник доступів

**7.4 Аналіз часової складності** Алгоритмічна складність програми: O(n)

|  |  |  |
| --- | --- | --- |
| Розмірність масиву/списку/ структури | Час витрачений на елементи масиву(мс) | Час витрачений на елементи двохзв'язного списку(мс) |
| 100 | 0,7869 | 1,2449 |
| 1000 | 0,8835 | 1,4442 |
| 5000 | 0,9529 | 1,5492 |
| 10000 | 0,9913 | 1,7819 |
| 20000 | 1,0692 | 1,8878 |

**7.4 Програмна реалізація алгоритму**

**7.4.1 Вихідний код на мові C#**

using System;  
using System.Collections.Generic;  
using System.Diagnostics;  
  
class Program  
{  
 static void Main(string[] args)  
 {  
 Console.Write("Enter 'a' to use an array or 'l' to use a doubly-linked list: ");  
 char choice = Convert.ToChar(Console.ReadLine());  
  
 Console.Write("Enter the size of the structure: ");  
 int n = Convert.ToInt32(Console.ReadLine());  
  
 Random random = new Random();  
 int[] array = null;  
 LinkedList<int> list = null;  
 if (choice == 'a')  
 {  
 array = new int[n];  
 for (int i = 0; i < n; i++)  
 {  
 array[i] = random.Next(100);  
 }  
 Array.Sort(array);  
 }  
 else if (choice == 'l')  
 {  
 list = new LinkedList<int>();  
 for (int i = 0; i < n; i++)  
 {  
 list.AddLast(random.Next(100));  
 }  
 List<int> sortedList = new List<int>(list);  
 sortedList.Sort();  
 list = new LinkedList<int>(sortedList);  
 }  
  
 int key = 50;  
 int comparisons = 0;  
 int arrayAccesses = 0;  
 int listAccesses = 0;  
 Stopwatch stopwatch = new Stopwatch();  
   
 stopwatch.Start();  
 int index;  
 if (choice == 'a')  
 {  
 index = FibonacciSearch.Search(array, key, out comparisons, out arrayAccesses);  
 Console.WriteLine($"Number of comparisons in array: {comparisons}");  
 Console.WriteLine($"Number of array accesses: {arrayAccesses}");  
 }  
 else  
 {  
 index = FibonacciSearch.Search(list, key, out comparisons, out listAccesses);  
 Console.WriteLine($"Number of comparisons in list: {comparisons}");  
 Console.WriteLine($"Number of list accesses: {listAccesses}");  
 }  
 stopwatch.Stop();  
   
 Console.WriteLine($"Index of {key}: {index}");  
 Console.WriteLine($"Time elapsed: {stopwatch.Elapsed.TotalMilliseconds} milliseconds");  
 }  
}  
  
class FibonacciSearch  
{  
 public static int Search(int[] array, int key, out int comparisons, out int arrayAccesses)  
 {  
 comparisons = 0;  
 arrayAccesses = 0;  
  
 int fib2 = 0;  
 int fib1 = 1;  
 int fib = fib2 + fib1;  
   
 while (fib < array.Length)  
 {  
 fib2 = fib1;  
 fib1 = fib;  
 fib = fib2 + fib1;  
 }  
  
 int offset = -1;  
  
 while (fib > 1)  
 {  
 int i = Math.Min(offset + fib2, array.Length - 1);  
  
 if (array[i] < key)  
 {  
 comparisons++;  
 arrayAccesses++;  
 fib = fib1;  
 fib1 = fib2;  
 fib2 = fib - fib1;  
 offset = i;  
 }  
 else if (array[i] > key)  
 {  
 comparisons++;  
 arrayAccesses++;  
 fib = fib2;  
 fib1 = fib1 - fib2;  
 fib2 = fib - fib1;  
 }  
 else  
 {  
 comparisons++;  
 arrayAccesses++;  
 return i;  
 }  
 }  
   
 if (fib1 == 1 && array[offset + 1] == key)  
 {  
 comparisons++;  
 arrayAccesses++;  
 return offset + 1;  
 }  
  
 return -1;  
 }  
  
 public static int Search(LinkedList<int> list, int key, out int comparisons, out int listAccesses)  
 {  
 comparisons = 0;  
 listAccesses = 0;  
  
 int fib2 = 0;  
 int fib1 = 1;  
 int fib = fib2 + fib1;  
  
 while (fib < list.Count)  
 {  
 fib2 = fib1;  
 fib1 = fib;  
 fib = fib2 + fib1;  
 }  
  
 int offset = -1;  
  
 while (fib > 1)  
 {  
 int i = Math.Min(offset + fib2, list.Count - 1);  
  
 int value = GetValueAt(list, i, out int myAccesses);  
 listAccesses += myAccesses;  
   
 if (value < key)  
 {  
 comparisons++;  
 fib = fib1;  
 fib1 = fib2;  
 fib2 = fib - fib1;  
 offset = i;  
 }  
 else if (value > key)  
 {  
 comparisons++;  
 fib = fib2;  
 fib1 = fib1 - fib2;  
 fib2 = fib - fib1;  
 }  
 else  
 {  
 comparisons++;  
 return i;  
 }  
 }  
  
 if (fib1 == 1 && GetValueAt(list, offset + 1, out int accesses) == key)  
 {  
 comparisons++;  
 listAccesses += accesses;  
 return offset + 1;  
 }  
  
 return -1;  
 }  
  
 private static int GetValueAt(LinkedList<int> list, int index, out int accesses)  
 {  
 accesses = 0;  
 LinkedListNode<int> node = list.First;  
 for (int i = 0; i < index; i++)  
 {  
 node = node.Next;  
 accesses++;  
 }  
 accesses++;  
 return node.Value;  
 }  
}

**7.4.2 Приклади роботи**

На рисунках 7.1 і 7.2 показані приклади роботи програми для пошуку індекса елемента за ключем для масиву на 100 елементів і двохзв'язного списку на 1000 елементів.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 7.1 Пошук елемента в масиві на 100 елементів

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рис. 7.2 Пошук елемента в двохзв'язному списку на 1000 елементів

**7.5 Тестування алгоритму**

**7.5.1 Часові характеристики оцінювання**

В таблиці 7.1 наведені характеристики оцінювання числа порівнянь при пошуку елемента і числа звертань при «двійковому пошуку» для масивів різної розмірності і двохзв'язних списків різної розмірності.

Таблиця 7.1 –Характеристики оцінювання алгоритму двійкового пошуку

|  |  |  |  |
| --- | --- | --- | --- |
| Розмірність масиву/списку/ структури | Число порівнянь в масиві/двохзв'язному списку/хеш-таблиці | Число звертань до елементів масиву | Число звертань до елементів двохзв'язного списку |
| 100 | 6 | 6 | 387 |
| 1000 | 7 | 7 | 3702 |
| 5000 | 8 | 8 | 19164 |
| 10000 | 9 | 9 | 30398 |
| 20000 | 9 | 9 | 70001 |

**7.5.2 Графіки залежності часових характеристик оцінювання від розмірності структури**

На рисунку 7.3 показані графіки залежності часових характеристик оцінювання від розмірності масиву і двохзв'язного списку.

Изображение выглядит как текст, линия, снимок экрана, График

Автоматически созданное описаниеРис.7.3 – Графіки залежності часових характеристик оцінювання

**7.6 Висновок**

В рамках виконання даної лабораторної роботи було розглянуто різні алгоритми пошуку, такі як лінійний пошук, бінарний пошук, пошук Фібоначчі та інші. Для кожного алгоритму було проаналізовано його часову складність, що дозволяє оцінити, як швидко алгоритм працює при збільшенні розміру вхідних даних.