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 та аналіз алгоритмів пошуку»

Варіант 5

Виконав студент ІП-15, Буяло Дмитро Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірив Соколовський Владислав Володимирович
(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота 7

Проектування та аналіз алгоритмів пошуку

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

Індивідуальне завдання

Варіант 5

Завдання

№	Алгоритм пошуку
5	Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом ланцюжків

Провести аналіз часової складності пошуку в гіршому, кращому і середньому випадках та записати часову складність в асимптотичних оцінках.

Використати безіндексну структуру даних розмірності n, що містить пару ключ-значення рядкового типу. Ключ – унікальне рядкове поле до 20 символів, значення – рядкове поле до 200 символів. Виконати пошук значення по заданому ключу. Розмірність хеш-таблиці регулювати відповідно потребам, а початкову її розмірність обрати самостійно.

Провести ряд випробувань алгоритму на структурах різної розмірності (100, 1000, 5000, 10000, 20000 елементів) і побудувати графіки залежності часових характеристик оцінювання від розмірності структури.

Зробити висновок з лабораторної роботи.

Виконання

1. Псевдокод алгоритму

Початок

init32 = 0x811c9dc5

prime32 = 0x01000193

mod32=2^32

hash = init32

Повторити для i від 0 до arr.length

hash ^= arr[i]

hash *= prime32

hash %= mod32

Все повторити

Кінець

2. Аналіз часової складності

Найкращий випадок: O(1)

Середній випадок: O(1)

Найгірший випадок: O(n)

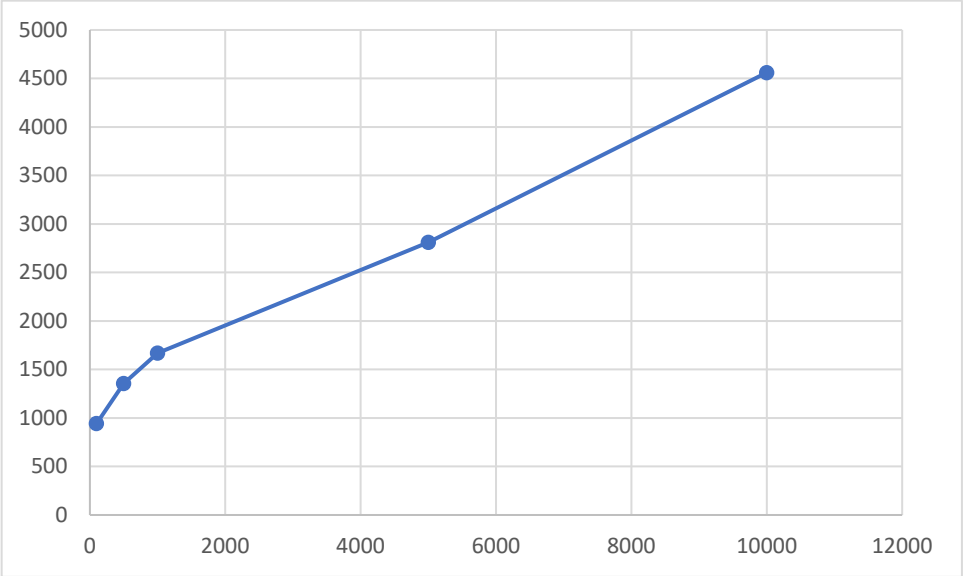

Рисунок 2.1 – Найгірший випадок

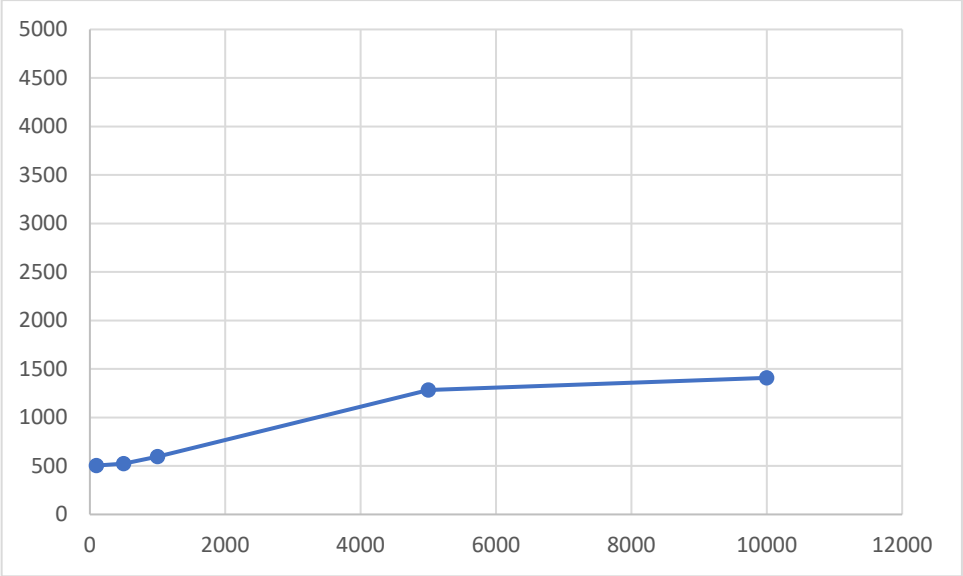

Рисунок 2.2 – Середній випадок

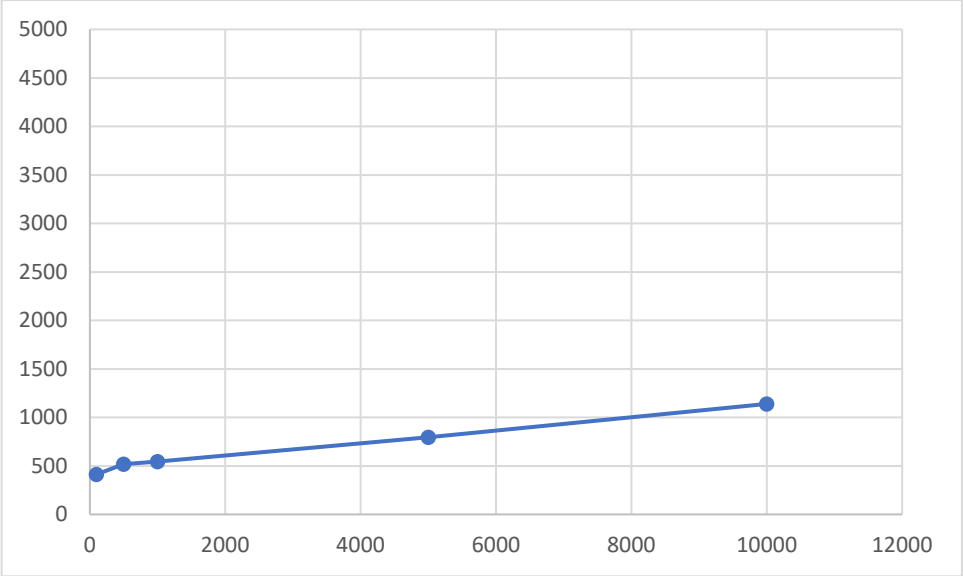

Рисунок 2.3 – Найкращий випадок

Щоб перевірити асимптотичну оцінку, наведемо рисунки 2.1, 2.2 та 2.3.

Графіки були побудовані за середніми результатами тестувань. Для кожної розмірності хеш-таблиці було зроблено 7 випробувань та обрано середнє.

3. Програмна реалізація алгоритму

3.1.Вихідний код

Main.java Fnv.java

```java
package com.company;

import java.io.File;
import java.io.FileNotFoundException;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) throws FileNotFoundException {
        HashMap<String, ArrayList<String>> h = new HashMap<>();

        Scanner in = new Scanner(System.in);
        System.out.print("Generate automatically? [Y/N]: ");
        String yn = in.nextLine();
        char yes = yn.charAt(0);

        if (yes == 'y' || yes == 'Y' || yes == '+' || yes == '1') {
            System.out.println("You chose automatic generation!");
            System.out.print("Enter the number of strings to hash: ");
            int count = in.nextInt();
            rand(count, h);
            in.nextLine();
        } else {
            System.out.println("You chose manual input!");
            File file = new File( pathname: "Hash.txt");
            Scanner scan = new Scanner(file);
            String input;
            while (scan.hasNextLine()) {
                input = scan.nextLine();
                hashing(input, h);
            }
        }
        System.out.println("\nHash table:");
        output(h);


        System.out.print("\nEnter a string to search: ");
        String toFind = in.nextLine();
        System.out.println(h.get(search(toFind)));
    }

    public static void hashing(String str, HashMap<String, ArrayList<String>> h) {
        Fnv f = new Fnv();

        byte[] byteArray = str.getBytes(StandardCharsets.UTF_8);

        String hash = f.fnv1a_32(byteArray) + "";
        if (h.containsKey(hash)) {
            h.get(hash).add(str);
```

Main.java Fnv.java

```
53          } else {
54              ArrayList<String> arr = new ArrayList<>();
55              arr.add(str);
56              h.put(hash, arr);
57          }
58      }
59
60  @   public static void output(HashMap<String, ArrayList<String>> h) {
61          for (String key : h.keySet()) {
62              System.out.println(h.get(key) + " " + key);
63          }
64      }
65
66      public static void rand(int counter, HashMap<String, ArrayList<String>> h) {
67          int randomLen, random;
68          String word;
69          for (int i = 0; i < counter; i++) {
70              word = "";
71              randomLen = (int) (Math.random() * 200 + 1);
72              for (int j = 0; j < randomLen; j++) {
73                  random = (int) (Math.random() * 94 + 33);
74                  word += (char) random;
75              }
76              hashing(word, h);
77          }
78      }
79
80  @   public static String search(String str) {
81          Fnv f = new Fnv();
82
83          byte[] byteArray = str.getBytes(StandardCharsets.UTF_8);
84
85          return f.fnv1a_32(byteArray)+"";
86      }
87  }
```

Main.java | Fnv.java

```java
package com.company;

import java.math.BigInteger;

public class Fnv {
    private static final BigInteger INIT32  = new BigInteger( val: "811c9dc5", radix: 16);
    private static final BigInteger PRIME32 = new BigInteger( val: "01000193", radix: 16);
    private static final BigInteger MOD32   = new BigInteger( val: "2").pow(32);

    public BigInteger fnv1a_32(byte[] data) {
        BigInteger hash = INIT32;

        for (byte b : data) {
            hash = hash.xor(BigInteger.valueOf((int) b & 0xff));
            hash = hash.multiply(PRIME32).mod(MOD32);
        }

        return hash;
    }
}
```

3.2.Приклад роботи

```
Generate automatically? [Y/N]: 2
You chose manual input!

Hash table:
[kkk] 1911768552
[абракадабра] 1981767587
[lol, lol] 877740032
[100] 1731450012
[pop] 1362321360

Enter a string to search: абракадабра
абракадабра

Process finished with exit code 0
```

```
Generate automatically? [Y/N]: no
You chose manual input!

Hash table:
[kkk] 1911768552
[абракадабра] 1981767587
[lol, lol] 877740032
[100] 1731450012
[pop] 1362321360

Enter a string to search: lol
lol
lol

Process finished with exit code 0
```

4. Тестування алгоритму

4.1.Часові характеристики оцінювання

У таблиці 4.1 наведені характеристики оцінювання часу при пошуку елемента за допомогою хеш-функції fnv32 для хеш таблиць різної розмірності.

Таблиця 4.1 – Характеристика оцінювання алгоритму пошуку хеш-функції

Розмірність структури	Середній час виконання, мс
100	944
500	1356
1000	1671
5000	2811
10000	4561

4.2.Графіки залежності часових характеристик оцінювання від розмірності структури

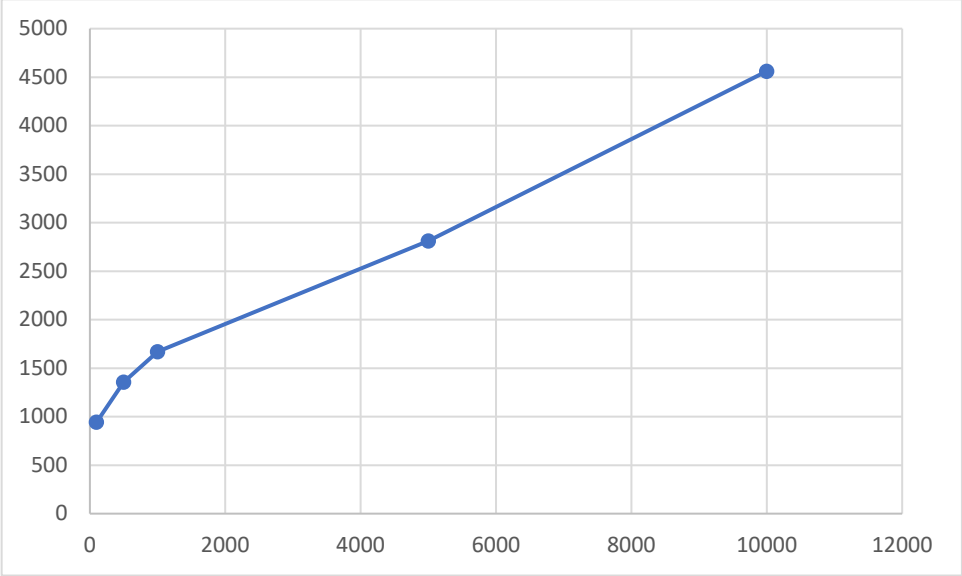

Рисунок 4.1 – Графік залежності часових характеристик оцінювання

Інші графіки були наведені у пункті 2 (Аналіз часової складності).

ВИСНОВОК

При виконанні сьомої лабораторної роботи, вивчили основні підходи аналізу обчислювальної складності алгоритмів пошуку, оцінили їх ефективність на різних структурах даних.

Розглянули алгоритми пошуку за допомогою модифікованої хеш-функції FNV32. Дослідили їх властивості, провели аналіз часової складності, провели ряд випробувань на різних наборах вхідних даних. За допомогою графіків порівняли часові характеристики оцінювання.

Отже, можемо прийти до висновку, що хеш-функція FNV 32 є ефективною. Проблему колізій вирішували методом ланцюжків, проте в наведеній функції хешування колізії дуже малоімовірні.