**Міністерство освіти і науки України**

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

**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизованих систем обробки інформації**

**і управління**

**Звіт**

з лабораторної роботи № 2 з дисципліни

«Теорія алгоритмів»

„ **Проектування і аналіз алгоритмів пошуку**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

ІП-92 Медведєв Михайло Євгенович

**Перевірив**

(прізвище, ім'я, по батькові)

Головченко М.Н.

Київ 2020

**Зміст**

[1 мета лабораторної роботи 3](#_Toc509037345)

[2 Завдання 4](#_Toc509037346)

[3 Виконання 8](#_Toc509037347)

[3.1 Псевдокод алгоритму 8](#_Toc509037349)

[3.2 Аналіз часової складності 8](#_Toc509037350)

[3.3 Програмна реалізація алгоритму 8](#_Toc509037351)

[3.3.1 Вихідний код 8](#_Toc509037352)

[3.3.2 Приклади роботи 8](#_Toc509037353)

[3.4 тестування алгоритму 9](#_Toc509037354)

[3.4.1 Часові характеристики оцінювання 9](#_Toc509037355)

[3.4.2 Графіки залежності часових характеристик оцінюваняя від розміру структури 10](#_Toc509037356)

[Висновок 11](#_Toc509037357)

[Критерії оцінювання 12](#_Toc509037358)

# Мета лабораторної роботи

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

# Завдання

Згідно варіанту (таблиця 2.1), написати алгоритм пошуку за допомогою псевдокоду (чи іншого способу за вибором).

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

Виконати програмну реалізацію алгоритму на будь-якій мові програмування для пошуку індексу елемента по заданому ключу в масиві і двохзв'язному списку з фіксацією часових характеристик оцінювання (кількість порівнянь).

Для варіантів з **Хеш-функцією** замість масиву і двохзв'язного списку 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 взяти рівною 100.

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

Для проведення випробувань у варіантах з хешуванням рекомендується розробити генератор псевдовипадкових значень полів структури заданої розмірності.

Зробити висновок з лабораторної роботи.

Таблиця 2.1 – Варіанти алгоритмів

|  |  |
| --- | --- |
| **№** | **Алгоритм пошуку** |
| 1 | Однорідний бінарний пошук |
| 2 | Метод Шарра |
| 3 | Пошук Фібоначчі |
| 4 | Інтерполяційний пошук |
| 5 | Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом ланцюжків |
| 6 | Метод Хеш-функції (Хешування MurmurHash2), вирішення колізій методом ланцюжків |
| 7 | Метод Хеш-функції (Хешування MurmurHash2а), вирішення колізій методом ланцюжків |
| 8 | Метод Хеш-функції (Хешування PJW-32), вирішення колізій методом ланцюжків |
| 9 | Метод Хеш-функції (Хешування Пірсона), вирішення колізій методом ланцюжків |
| 10 | Метод Хеш-функції (Хешування Дженкінса), вирішення колізій методом ланцюжків |
| 11 | Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом відкритої адресації з лінійним пробуванням |
| 12 | Метод Хеш-функції (Хешування MurmurHash2), вирішення колізій методом відкритої адресації з лінійним пробуванням |
| 13 | Метод Хеш-функції (Хешування MurmurHash2а), вирішення колізій методом відкритої адресації з лінійним пробуванням |
| 14 | Метод Хеш-функції (Хешування PJW-32), вирішення колізій методом відкритої адресації з лінійним пробуванням |
| 15 | Метод Хеш-функції (Хешування Пірсона), вирішення колізій методом відкритої адресації з лінійним пробуванням |
| 16 | Метод Хеш-функції (Хешування Дженкінса), вирішення колізій методом відкритої адресації з лінійним пробуванням |
| 17 | Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом відкритої адресації з квадратичним пробуванням |
| 18 | Метод Хеш-функції (Хешування MurmurHash2), вирішення колізій методом відкритої адресації з квадратичним пробуванням |
| 19 | Метод Хеш-функції (Хешування MurmurHash2а), вирішення колізій методом відкритої адресації з квадратичним пробуванням |
| 20 | Метод Хеш-функції (Хешування PJW-32), вирішення колізій методом відкритої адресації з квадратичним пробуванням |
| 21 | Метод Хеш-функції (Хешування Пірсона), вирішення колізій методом відкритої адресації з квадратичним пробуванням |
| 22 | Метод Хеш-функції (Хешування Дженкінса), вирішення колізій методом відкритої адресації з квадратичним пробуванням |
| 23 | Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом відкритої адресації з подвійним хешуванням |
| 24 | Метод Хеш-функції (Хешування MurmurHash2), вирішення колізій методом відкритої адресації з подвійним хешуванням |
| 25 | Метод Хеш-функції (Хешування MurmurHash2а), вирішення колізій методом відкритої адресації з подвійним хешуванням |
| 26 | Метод Хеш-функції (Хешування PJW-32), вирішення колізій методом відкритої адресації з подвійним хешуванням |
| 27 | Метод Хеш-функції (Хешування Пірсона), вирішення колізій методом відкритої адресації з подвійним хешуванням |
| 28 | Метод Хеш-функції (Хешування Дженкінса), вирішення колізій методом відкритої адресації з подвійним хешуванням |
| 29 | Однорідний бінарний пошук |
| 30 | Метод Шарра |
| 31 | Пошук Фібоначчі |
| 32 | Інтерполяційний пошук |
| 33 | Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом ланцюжків |
| 34 | Метод Хеш-функції (Хешування MurmurHash2), вирішення колізій метоом ланцюжків |
| 35 | Метод Хеш-функції (Хешування MurmurHash2а), вирішення колізій методом ланцюжків |

# ВИконання

## Псевдокод алгоритму

hash = hash(key);  
index = hash mod(length)

startIndex = index;

i=0  
 **while** table[index] != **null** and !table[index].mKey.equals(key) {   
 index = (hash(key) + i\*temp)mod(length)

i+=1

**if** (index == startIndex)  
 **break**;  
 }  
  
 **if** (table[index] == **null**)  
 **return null**;  
  
 **if** (index == startIndex && table[index] != **null** and !table[index].mKey.equals(key))  
 **return null**;  
  
 **return** table[index].mValue;

## Аналіз часової складності

Найгірший випадок – О(n)

Найкращий випадок – O(1)

Середній випадок:

При невдалому пошуку 1/(1-a)

При вдалому пошуку 1/a \* ln(1/1-a)

Де а – коефіціент заповненості таблиці

## Програмна реалізація алгоритму

### Вихідний код

**public static long** hash32(**final byte**[] data, **int** length, **int** seed) {  
 **final int** m = 0x5bd1e995;  
 **final int** r = 24;  
 **int** h = seed ^ length;  
 **int** length4 = length / 4;  
  
 **for** (**int** i = 0; i < length4; i++) {  
 **final int** i4 = i \* 4;  
 **int** k = (data[i4 + 0] & 0xff) + ((data[i4 + 1] & 0xff) << 8)  
 + ((data[i4 + 2] & 0xff) << 16) + ((data[i4 + 3] & 0xff) << 24);  
 k \*= m;  
 k ^= k >>> r;  
 k \*= m;  
 h \*= m;  
 h ^= k;  
 }  
 **switch** (length % 4) {  
 **case** 3:  
 h ^= (data[(length & ~3) + 2] & 0xff) << 16;  
 **case** 2:  
 h ^= (data[(length & ~3) + 1] & 0xff) << 8;  
 **case** 1:  
 h ^= (data[length & ~3] & 0xff);  
 h \*= m;  
 }  
 h ^= h >>> 13;  
 h \*= m;  
 h ^= h >>> 15;  
  
 **return** Integer.*toUnsignedLong*(h);  
}

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми для пошуку індекса елемента за ключем для хеш-таблиці на 100 і на 1000 елементів.

Рисунок 3.1 – Пошук елемента в хеш-таблиці на 100 елементів

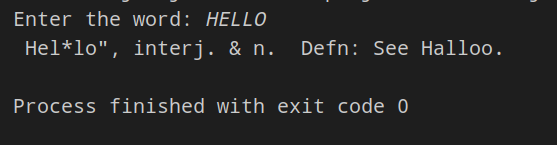
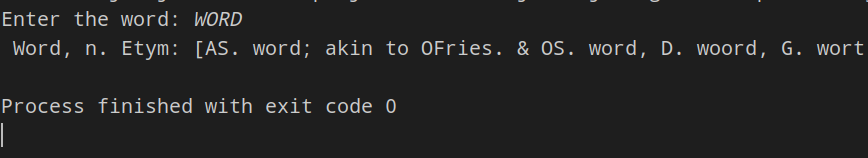


Рисунок 3.2 – Пошук елемента в хеш-таблиці на 100 елементів



## Тестування алгоритму

### Часові характеристики оцінювання

В таблиці 3.1 наведені характеристики оцінювання числа порівнянь при пошуку елемента для масивів різної розмірності і двохзв'язних списків різної розмірності.

Таблиця 3.1 Характеристики оцінювання методу хеш-функції (хешування MurmurHash2)

|  |  |
| --- | --- |
| Розмірність масиву/списку/  структури | Число порівнянь в масиві/двохзв'язному списку/хеш-таблиці |
| 100 | 4 |
| 1000 | 20 |
| 5000 | 29 |
| 10000 | 72 |
| 20000 | 80 |

### Графіки залежності часових характеристик оцінювання від розмірності структури

На рисунку 3.3 показані графіки залежності часових характеристик оцінювання від розмірності хеш-таблиці.

Рисунок 3.3 – Графіки залежності часових характеристик оцінювання

Висновок

В рамках виконання даної лабораторної роботи мною було проведено експериментальне дослідження методу хеш-функції (Хешування MurmurHash2) і вирішення колізій методом відкритої адресації з лінійним пробуванням. Всі досліди були проведені на прикладі хеш-таблиці у вигляді массиву данних «ключ - значення».

Хешування MurmurHash2 показало стабільну роботу як для великих, так і для малих ключів, достатню простоту роботу та відносно невелику кількість колізій у хеш-таблиці. Однак при деяких спеціальних ключах дане хешування видає надто великі колізії порівняно з іншими, більш новими хеш-функціями.

Залежність кількості порівнянь у хеш-таблиці при пошуку елемента до розмірності структури є майже лінійною, що є доволі гарним показником для ключів малого та середнього розмірів. Для більших ключів доцільно було б використовувати інші хеш-функції.

Критерії оцінювання

За умови здачі лабораторної роботи до 5.04.2020 включно надається можливість виправити помилки без втрати балів, за бажанням. Після 5.04.2020 оцінка за лабораторну роботу ставиться по факту здачі. Максимальний бал дорівнює 5.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 20%;
* аналіз часової складності– 20%;
* програмна реалізація алгоритму– 25%;
* тестування алгоритму– 20%;
* висновки – 5%.