Міністерство освіти і науки України

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

Факультет інформатики та обчислювальної техніки Кафедра ІПІ

Звіт

з лабораторної роботи №6
«Проектування і аналіз алгоритмів пошуку»

Виконав(ла) *ІП-22, Андреєва Уляна Андріївна*

(шифр, прізвище, ім'я, по батькові)

Перевірила *Халус Олена Андріївна*

(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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 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.

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

Зробити висновок з лабораторної роботи.

<u>Алгоритм пошуку:</u>

Метод Хеш-функції (Хешування FNV 32), вирішення колізій методом ланцюжків

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

HashTable get

```
function getNode(key):
    currentNode = head
    while currentNode is not null:
        if currentNode.getKey() equals key:
            getComparisons++
            return currentNode
        currentNode = currentNode.getNext()
        getComparisons++
    return null
```

LinkedList getNode

```
function get(key):
    index = abs(hash(key) % size)
    hashTableGetComparisons++
    node = table[index].getNode(key)
    if node is not null:
        return node.getValue()
    table[index].getComparisons++ // каунтер порівнянь для пошуку
    return null
```

3.2 Аналіз часової складності

*1 **Хешування ключа (функція hash):***

- Цикл залежить від довжини ключа **key.length()**, тому має часову складність O(key.length()).
- Операції порівняння та арифметичні операції мають константний часовий розгортання.
- Таким чином, часова складність функції **hash** - O(key.length()).

*2 **Функція put:***

- Хешування ключа має часову складність O(key.length()).
- Отримання вузла списку має часову складність, яка залежить від розміру списку. Нехай розмір списку в середньому складає k елементів, тоді часова складність буде O(k).
- Якщо вузол знайдений, часова складність присвоєння значення - константна.
- Якщо вузол не знайдений, часова складність додавання нового вузла залежить від розміру списку, тобто O(k).

- Операції порівняння та арифметичні операції мають константний часовий розгортання.
- Таким чином, загальна часова складність функції **put** - O(key.length() + k).

3 Функція get:

- Хешування ключа має часову складність O(key.length()).
- Отримання вузла списку має часову складність, яка залежить від розміру списку. Нехай розмір списку в середньому складає k елементів, тоді часова складність буде O(k).
- Якщо вузол знайдений, часова складність отримання значення - константна.
- Операції порівняння та арифметичні операції мають константний часовий розгортання.
- Таким чином, загальна часова складність функції **get** - O(key.length() + k).

Програмна реалізація алгоритму

3.3.1 Вихідний код

```
import java.util.Random;
import java.util.Scanner;

public class HashTable  {
    private final int size;
    private final LinkedList[] table;
    private int hashTableGetComparisons = 0;


    public HashTable(int size) {
        this.size = size;
        this.table = new LinkedList[size];
        for (int i = 0; i < size; i++) {
            table[i] = new LinkedList ();
        }
    }

    private int hash(String key) {
        final int p = 16777619;
        int hash = (int) 2166136261L;
        for (int i = 0; i < key.length (); i++) {
            hash = (hash ^ key.charAt ( i )) * p;
        }
        hash += hash << 13;
        hash ^= hash >> 7;
        hash += hash << 3;
        hash ^= hash >> 17;
        hash += hash << 5;
        return hash;
    }
```

```
    public void put(String key, String value) {
        int index = Math.abs ( hash ( key ) % size );
        Node node = table[index].getNode ( key );
        if (node != null) {
            node.setValue ( value );
        } else {
            table[index].addNode ( key, value );
        }
        table[index].putComparisons++; // каунтер порівнянь для додавання
    }

    public String get(String key) {
        int index = Math.abs ( hash ( key ) % size );
        hashTableGetComparisons++;
        Node node = table[index].getNode ( key );
        if (node != null) {
            return node.getValue ();
        }
        table[index].getComparisons++; // каунтер порівнянь для пошуку
        return null;
    }

    public void printTable() {
        for (int i = 0; i < size; i++) {
            System.out.print ( i + ": " );
            if (table[i].size () == 0) {
                System.out.println ( "Empty" );
            } else {
                Node currentNode = table[i].getHead ();
                while (currentNode != null) {
                    System.out.print ( currentNode.getKey () + "=" +
currentNode.getValue () + " " );
                    currentNode = currentNode.getNext ();
                }
                System.out.println ();
            }
        }
    }

    public int getGetComparisons() {
        return hashTableGetComparisons;
    }

    public void resetComparisons(){
        hashTableGetComparisons = 0;
    }

    private static class LinkedList {
        private HashTable.Node head;
        private int putComparisons; // каунтер порівнянь для додавання
        private int getComparisons; // каунтер порівнянь для пошуку

        public LinkedList() {
            this.head = null;
            this.putComparisons = 0; // ініціалізація каунтерів
            this.getComparisons = 0;
        }

        public HashTable.Node getHead() {
            return head;
        }
```

```
        public int size() {
            int count = 0;
            HashTable.Node currentNode = head;
            while (currentNode != null) {
                count++;
                currentNode = currentNode.getNext ();
            }
            return count;
        }

        public void addNode(String key, String value) {
            if (head == null) {
                head = new HashTable.Node ( key, value );
            } else {
                HashTable.Node currentNode = head;
                while (currentNode.getNext () != null) {
                    currentNode = currentNode.getNext ();
                }
                currentNode.setNext ( new HashTable.Node ( key, value ) );
            }
        }

        public HashTable.Node getNode(String key) {
            HashTable.Node currentNode = head;
            while (currentNode != null) {
                if (currentNode.getKey ().equals ( key )) {
                    getComparisons++; // збільшуємо каунтер порівнянь для
пошуку
                    return currentNode;
                }
                currentNode = currentNode.getNext ();
                getComparisons++; // збільшуємо каунтер порівнянь для
пошуку
            }
            return null;
        }

        public int getPutComparisons() {
            return putComparisons;
        }

        public int getGetComparisons() {
            return getComparisons;
        }

        public void resetComparisons(){
            getComparisons = 0;
            putComparisons = 0;
        }

    }private static class Node {
        private final String key;
        private String value;
        private Node next;

        public Node(String key, String value) {
            this.key = key;
            this.value = value;
            this.next = null;
        }
```

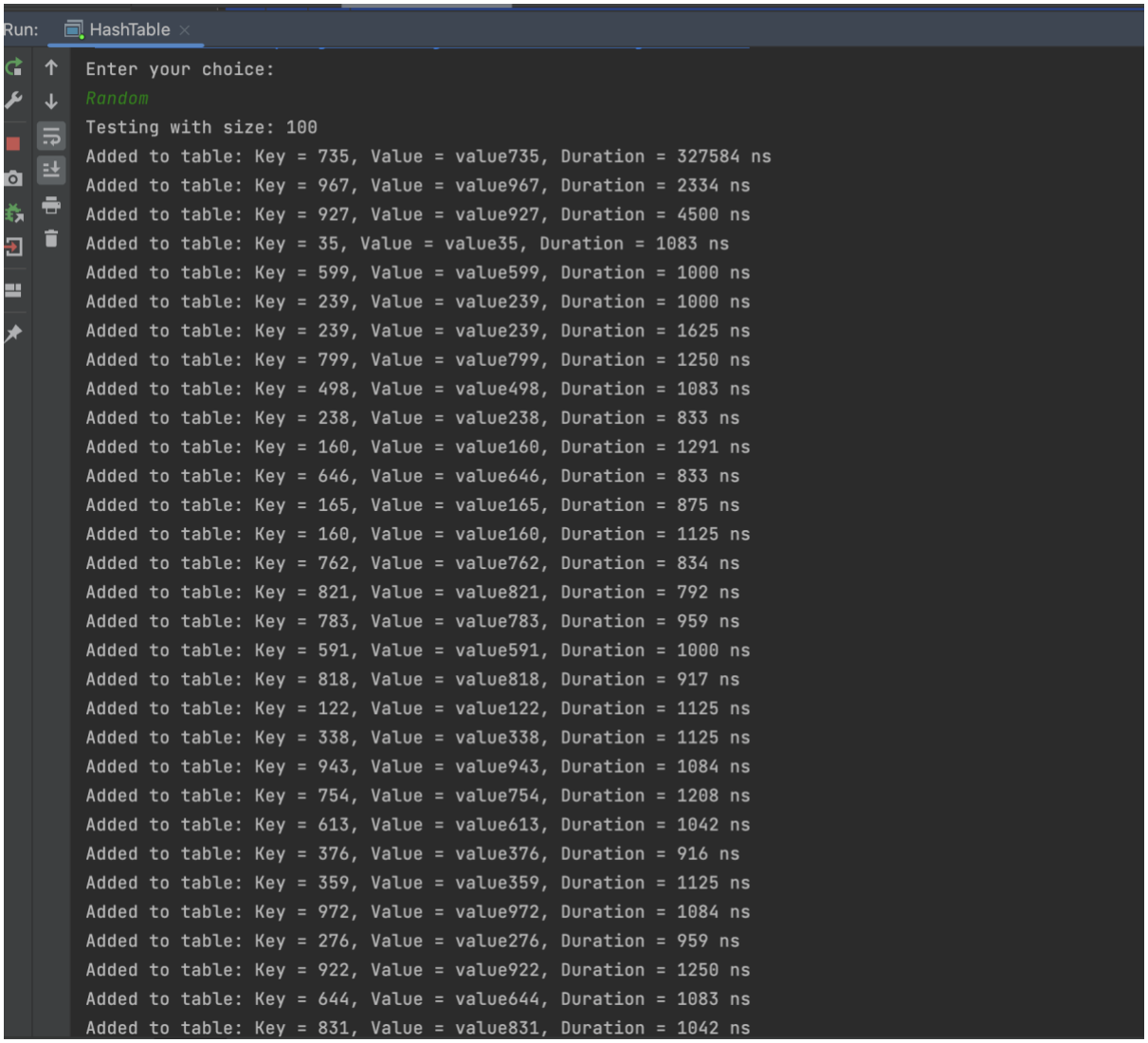
```
        public String getKey() {
            return key;
        }

        public String getValue() {
            return value;
        }

        public void setValue(String value) {
            this.value = value;
        }

        public Node getNext() {
            return next;
        }

        public void setNext(Node next) {
            this.next = next;
        }
    }

    private static final int[] TEST_SIZES = { 100, 1000, 5000, 10000,
20000};

    public static void main(String[] args) {
        Scanner scanner2 = new Scanner ( System.in );
        System.out.println ( "Enter your choice: " );
        String choiceSTR = scanner2.nextLine ();
        HashTable table = new HashTable ( 10 );
        Scanner scanner = new Scanner ( System.in );
        Random random = new Random ();
        try {

        switch (choiceSTR) {
            case "Manual":
                System.out.print ( "Enter the number of elements to ADD to
the table: " );
                int numElements = scanner.nextInt ();

                for (int i = 0; i < numElements; i++) {
                    int key = random.nextInt ( 1000 );
                    String value = "value" + key;
                    long startTime = System.nanoTime ();
                    table.put ( Integer.toString ( key ), value );
                    long endTime = System.nanoTime ();
                    long duration = (endTime - startTime);
                    System.out.println ( "Added to table: Key = " + key +
", Value = " + value + ", Duration = " + duration + " ns" );
                }

                table.printTable();
                System.out.println();

                System.out.print("Enter key to find: ");
                int key = scanner.nextInt();
                long startTime = System.nanoTime();
                String value = table.get(Integer.toString(key));
                long endTime = System.nanoTime();
                long duration = (endTime - startTime);
                System.out.println("Searched for the key: " + key +
```

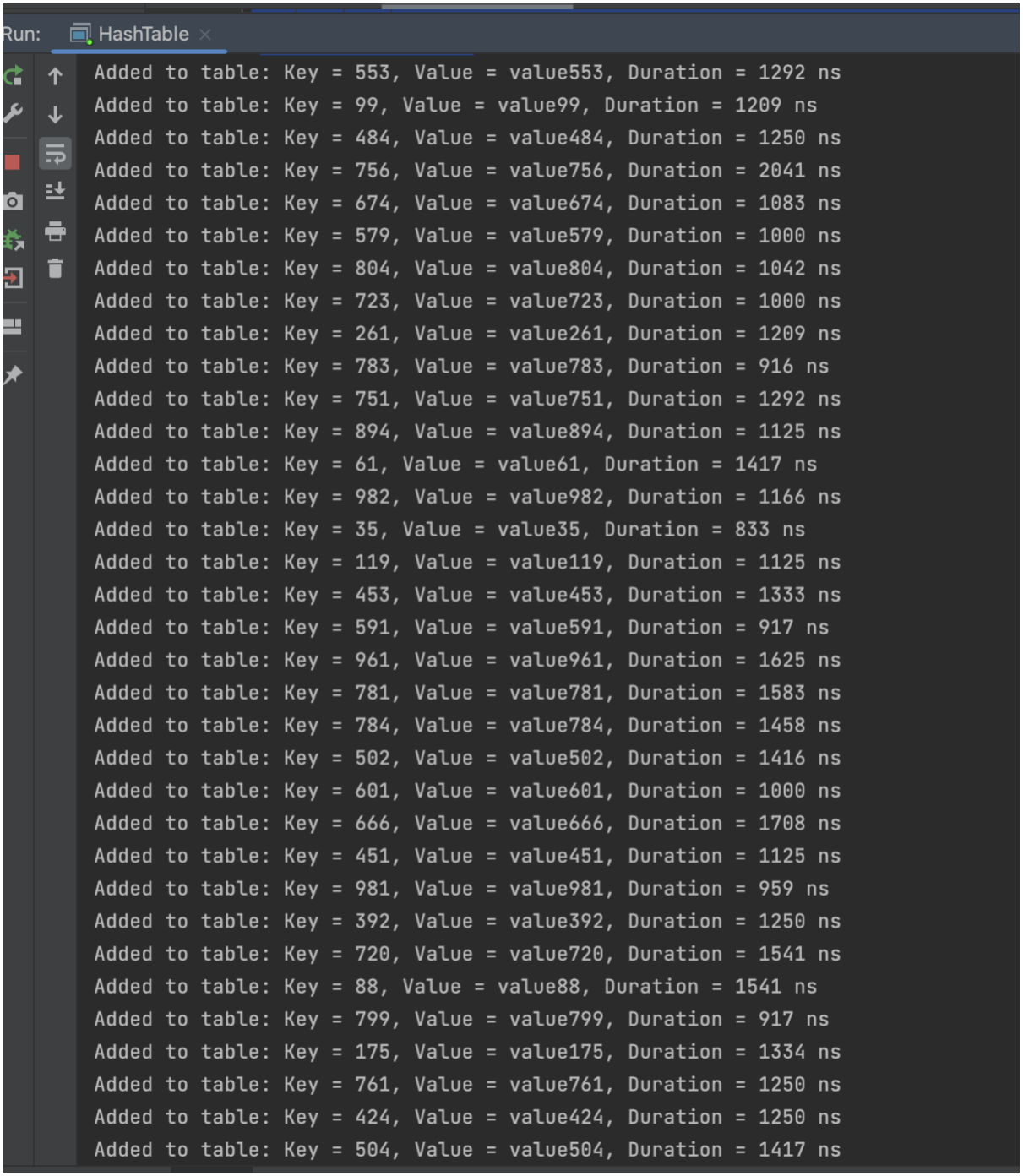
```
",duration of the searching is - " + duration + " ns");
                if (value != null) {
                    System.out.println("Found: " + value);
                } else {
                    System.out.println("key " + key + " with value " +
value + " not found");
                }

                LinkedList linkedList = table.table[0];
                System.out.println ( "Number of comparisons for adding to
linked list: " + linkedList.getPutComparisons () );
                System.out.println ( "Number of comparisons for searching
in linked list : " + linkedList.getGetComparisons () );
                System.out.println ( "Number of comparisons for searching
in hash table: " + table.getGetComparisons () );

                break;
            case "Random":

                for (int size : TEST_SIZES) {
                    System.out.println("Testing with size: " + size);
                    for (int i = 0; i < size; i++) {
                        int key2 = random.nextInt(1000);
                        String value2 = "value" + key2;
                        long startTime2 = System.nanoTime();
                        table.put(Integer.toString(key2), value2);
                        long endTime2 = System.nanoTime();
                        long duration2 = (endTime2 - startTime2);
                        System.out.println("Added to table: Key = " + key2
+ ", Value = " + value2 + ", Duration = " + duration2 + " ns");
                    }


                    table.printTable();
                    System.out.println();

                    System.out.print("Enter key to find: ");
                    int key3 = scanner.nextInt();
                    long startTime3 = System.nanoTime();
                    String value3 = table.get(Integer.toString(key3));
                    long endTime3 = System.nanoTime();
                    long duration3 = (endTime3 - startTime3);
                    System.out.println("Searched for the key: " + key3 +
",duration of the searching is - " + duration3 + " ns");
                    if (value3 != null) {
                        System.out.println("Found: " + value3);
                    } else {
                        System.out.println("key " + key3 + " with value " +
value3 + " not found");
                    }

                    LinkedList linkedList2 = table.table[0];
                    System.out.println ( "Number of comparisons for adding
to linked list: " + linkedList2.getPutComparisons () + ", the size is " +
size);
                    System.out.println ( "Number of comparisons for
searching in linked list : " + linkedList2.getGetComparisons ()  + ", the
size is " + size);
                    System.out.println ( "Number of comparisons for
searching in hash table: " + table.getGetComparisons ()  + ", the size is "
+ size);
```

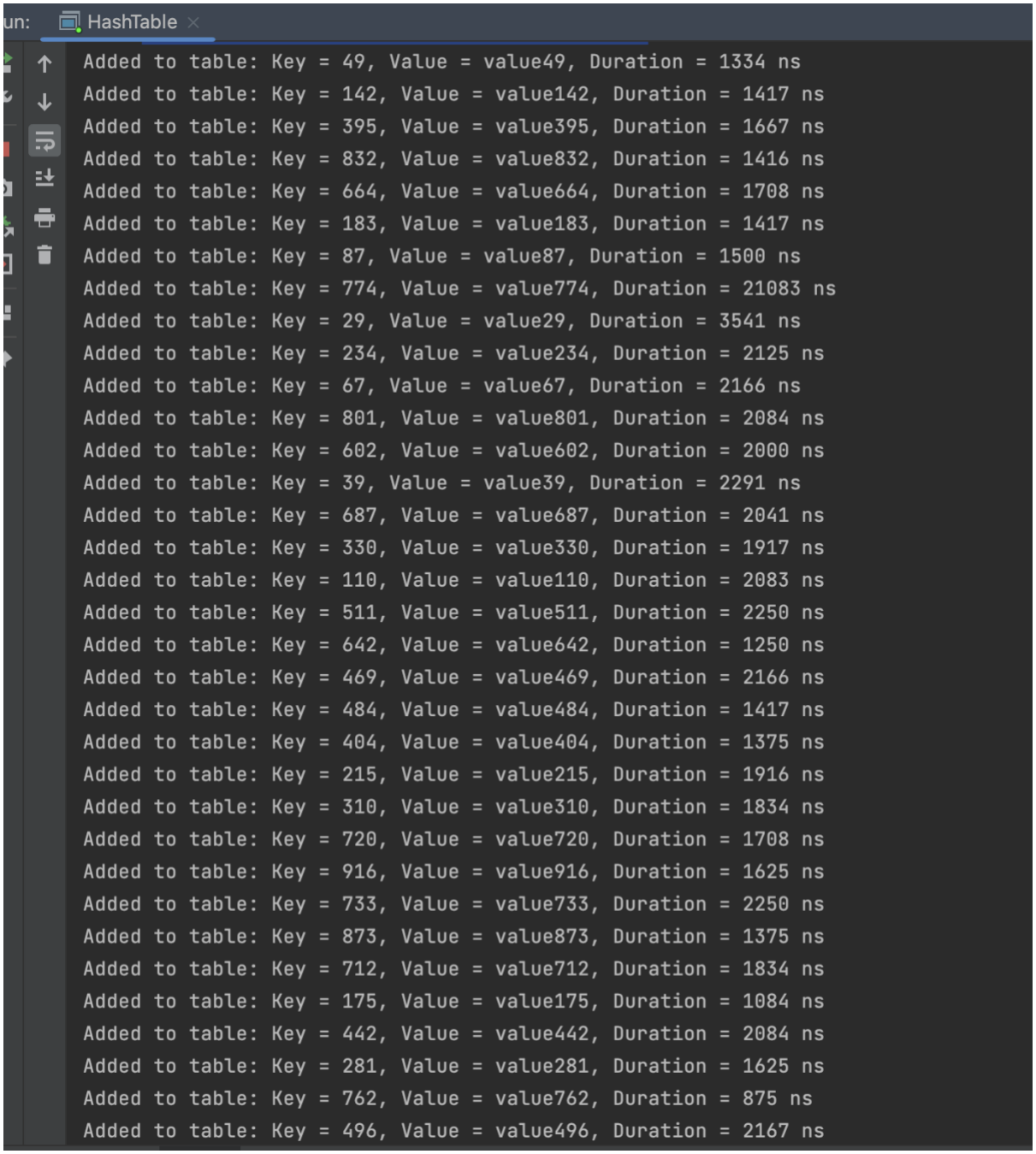
```
                    linkedList2.resetComparisons();
                    table.resetComparisons();

                    System.out.println("Continue?");
                    scanner2.nextLine ();
                }
                break;
            default:
                throw new IllegalArgumentException("Invalid choice: " +
choiceSTR);
            }
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
            }

}
```

3.3.1 Приклади роботи

```
/Users/mac/Desktop/Algorithm_Projects-/Labwork6/target/classes HashTable
Enter your choice:
Manual
Enter the number of elements to ADD to the table: 10
Added to table: Key = 474, Value = value474, Duration = 1255375 ns
Added to table: Key = 225, Value = value225, Duration = 4625 ns
Added to table: Key = 174, Value = value174, Duration = 2750 ns
Added to table: Key = 299, Value = value299, Duration = 2083 ns
Added to table: Key = 931, Value = value931, Duration = 1625 ns
Added to table: Key = 804, Value = value804, Duration = 8541 ns
Added to table: Key = 557, Value = value557, Duration = 3083 ns
Added to table: Key = 626, Value = value626, Duration = 2792 ns
Added to table: Key = 701, Value = value701, Duration = 1875 ns
Added to table: Key = 934, Value = value934, Duration = 2292 ns
0: Empty
1: 174=value174 557=value557
2: 701=value701
3: 474=value474
4: 299=value299 804=value804 626=value626
5: 225=value225
6: Empty
7: 931=value931 934=value934
8: Empty
9: Empty

Enter key to find: 174
Searched for the key: 174,duration of the searching is - 81583 ns
Found: value174
Number of comparisons for adding to linked list: 0
Number of comparisons for searching in linked list : 0
Number of comparisons for searching in hash table: 1

Process finished with exit code 0
```


Run:
HashTable
Enter your choice:
Random
Testing with size: 100
Added to table: Key = 735, Value = value735, Duration = 327584 ns
Added to table: Key = 967, Value = value967, Duration = 2334 ns
Added to table: Key = 927, Value = value927, Duration = 4500 ns
Added to table: Key = 35, Value = value35, Duration = 1083 ns
Added to table: Key = 599, Value = value599, Duration = 1000 ns
Added to table: Key = 239, Value = value239, Duration = 1000 ns
Added to table: Key = 239, Value = value239, Duration = 1625 ns
Added to table: Key = 799, Value = value799, Duration = 1250 ns
Added to table: Key = 498, Value = value498, Duration = 1083 ns
Added to table: Key = 238, Value = value238, Duration = 833 ns
Added to table: Key = 160, Value = value160, Duration = 1291 ns
Added to table: Key = 646, Value = value646, Duration = 833 ns
Added to table: Key = 165, Value = value165, Duration = 875 ns
Added to table: Key = 160, Value = value160, Duration = 1125 ns
Added to table: Key = 762, Value = value762, Duration = 834 ns
Added to table: Key = 821, Value = value821, Duration = 792 ns
Added to table: Key = 783, Value = value783, Duration = 959 ns
Added to table: Key = 591, Value = value591, Duration = 1000 ns
Added to table: Key = 818, Value = value818, Duration = 917 ns
Added to table: Key = 122, Value = value122, Duration = 1125 ns
Added to table: Key = 338, Value = value338, Duration = 1125 ns
Added to table: Key = 943, Value = value943, Duration = 1084 ns
Added to table: Key = 754, Value = value754, Duration = 1208 ns
Added to table: Key = 613, Value = value613, Duration = 1042 ns
Added to table: Key = 376, Value = value376, Duration = 916 ns
Added to table: Key = 359, Value = value359, Duration = 1125 ns
Added to table: Key = 972, Value = value972, Duration = 1084 ns
Added to table: Key = 276, Value = value276, Duration = 959 ns
Added to table: Key = 922, Value = value922, Duration = 1250 ns
Added to table: Key = 644, Value = value644, Duration = 1083 ns
Added to table: Key = 831, Value = value831, Duration = 1042 ns

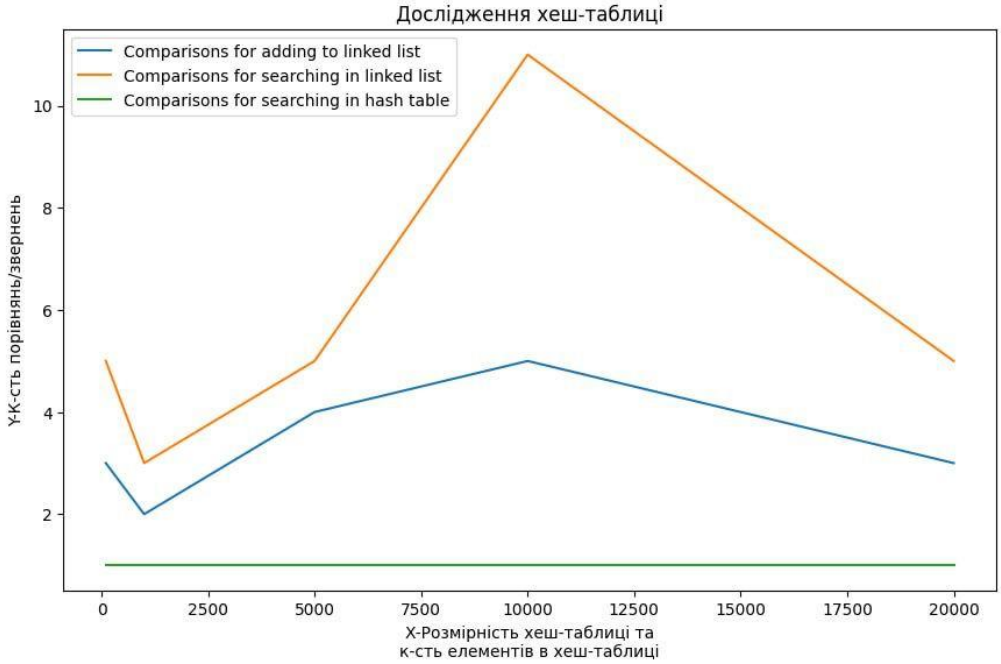

Run:
HashTable
Added to table: Key = 553, Value = value553, Duration = 1292 ns
Added to table: Key = 99, Value = value99, Duration = 1209 ns
Added to table: Key = 484, Value = value484, Duration = 1250 ns
Added to table: Key = 756, Value = value756, Duration = 2041 ns
Added to table: Key = 674, Value = value674, Duration = 1083 ns
Added to table: Key = 579, Value = value579, Duration = 1000 ns
Added to table: Key = 804, Value = value804, Duration = 1042 ns
Added to table: Key = 723, Value = value723, Duration = 1000 ns
Added to table: Key = 261, Value = value261, Duration = 1209 ns
Added to table: Key = 783, Value = value783, Duration = 916 ns
Added to table: Key = 751, Value = value751, Duration = 1292 ns
Added to table: Key = 894, Value = value894, Duration = 1125 ns
Added to table: Key = 61, Value = value61, Duration = 1417 ns
Added to table: Key = 982, Value = value982, Duration = 1166 ns
Added to table: Key = 35, Value = value35, Duration = 833 ns
Added to table: Key = 119, Value = value119, Duration = 1125 ns
Added to table: Key = 453, Value = value453, Duration = 1333 ns
Added to table: Key = 591, Value = value591, Duration = 917 ns
Added to table: Key = 961, Value = value961, Duration = 1625 ns
Added to table: Key = 781, Value = value781, Duration = 1583 ns
Added to table: Key = 784, Value = value784, Duration = 1458 ns
Added to table: Key = 502, Value = value502, Duration = 1416 ns
Added to table: Key = 601, Value = value601, Duration = 1000 ns
Added to table: Key = 666, Value = value666, Duration = 1708 ns
Added to table: Key = 451, Value = value451, Duration = 1125 ns
Added to table: Key = 981, Value = value981, Duration = 959 ns
Added to table: Key = 392, Value = value392, Duration = 1250 ns
Added to table: Key = 720, Value = value720, Duration = 1541 ns
Added to table: Key = 88, Value = value88, Duration = 1541 ns
Added to table: Key = 799, Value = value799, Duration = 917 ns
Added to table: Key = 175, Value = value175, Duration = 1334 ns
Added to table: Key = 761, Value = value761, Duration = 1250 ns
Added to table: Key = 424, Value = value424, Duration = 1250 ns
Added to table: Key = 504, Value = value504, Duration = 1417 ns


un:
HashTable
Added to table: Key = 49, Value = value49, Duration = 1334 ns
Added to table: Key = 142, Value = value142, Duration = 1417 ns
Added to table: Key = 395, Value = value395, Duration = 1667 ns
Added to table: Key = 832, Value = value832, Duration = 1416 ns
Added to table: Key = 664, Value = value664, Duration = 1708 ns
Added to table: Key = 183, Value = value183, Duration = 1417 ns
Added to table: Key = 87, Value = value87, Duration = 1500 ns
Added to table: Key = 774, Value = value774, Duration = 21083 ns
Added to table: Key = 29, Value = value29, Duration = 3541 ns
Added to table: Key = 234, Value = value234, Duration = 2125 ns
Added to table: Key = 67, Value = value67, Duration = 2166 ns
Added to table: Key = 801, Value = value801, Duration = 2084 ns
Added to table: Key = 602, Value = value602, Duration = 2000 ns
Added to table: Key = 39, Value = value39, Duration = 2291 ns
Added to table: Key = 687, Value = value687, Duration = 2041 ns
Added to table: Key = 330, Value = value330, Duration = 1917 ns
Added to table: Key = 110, Value = value110, Duration = 2083 ns
Added to table: Key = 511, Value = value511, Duration = 2250 ns
Added to table: Key = 642, Value = value642, Duration = 1250 ns
Added to table: Key = 469, Value = value469, Duration = 2166 ns
Added to table: Key = 484, Value = value484, Duration = 1417 ns
Added to table: Key = 404, Value = value404, Duration = 1375 ns
Added to table: Key = 215, Value = value215, Duration = 1916 ns
Added to table: Key = 310, Value = value310, Duration = 1834 ns
Added to table: Key = 720, Value = value720, Duration = 1708 ns
Added to table: Key = 916, Value = value916, Duration = 1625 ns
Added to table: Key = 733, Value = value733, Duration = 2250 ns
Added to table: Key = 873, Value = value873, Duration = 1375 ns
Added to table: Key = 712, Value = value712, Duration = 1834 ns
Added to table: Key = 175, Value = value175, Duration = 1084 ns
Added to table: Key = 442, Value = value442, Duration = 2084 ns
Added to table: Key = 281, Value = value281, Duration = 1625 ns
Added to table: Key = 762, Value = value762, Duration = 875 ns
Added to table: Key = 496, Value = value496, Duration = 2167 ns

```
HashTable
Talking: Дмитро М'яч
Added to table: Key = 404, Value = value404, Duration = 1375 ns
Added to table: Key = 215, Value = value215, Duration = 1916 ns
Added to table: Key = 310, Value = value310, Duration = 1834 ns
Added to table: Key = 720, Value = value720, Duration = 1708 ns
Added to table: Key = 916, Value = value916, Duration = 1625 ns
Added to table: Key = 733, Value = value733, Duration = 2250 ns
Added to table: Key = 873, Value = value873, Duration = 1375 ns
Added to table: Key = 712, Value = value712, Duration = 1834 ns
Added to table: Key = 175, Value = value175, Duration = 1084 ns
Added to table: Key = 442, Value = value442, Duration = 2084 ns
Added to table: Key = 281, Value = value281, Duration = 1625 ns
Added to table: Key = 762, Value = value762, Duration = 875 ns
Added to table: Key = 496, Value = value496, Duration = 2167 ns
Added to table: Key = 635, Value = value635, Duration = 2291 ns
0: 967=value967 981=value981 234=value234 642=value642 404=value404 873=value873
1: 821=value821 276=value276 831=value831 756=value756 392=value392 664=value664 183=value183 39=value39 687=value687 442=value442
2: 165=value165 783=value783 338=value338 644=value644 99=value99 751=value751 961=value961 801=value801 602=value602 215=value215
3: 35=value35 498=value498 122=value122 754=value754 922=value922 553=value553 61=value61 781=value781 395=value395 29=value29 511=value511 469=value469 733=value733
  635=value635
4: 762=value762 804=value804 723=value723 119=value119 666=value666 49=value49 832=value832 67=value67 110=value110
5: 599=value599 799=value799 613=value613 674=value674 982=value982 88=value88 504=value504 774=value774 310=value310
6: 646=value646 376=value376 579=value579 894=value894 761=value761 424=value424 87=value87 330=value330
7: 239=value239 818=value818 943=value943 359=value359 484=value484 453=value453 784=value784 502=value502 720=value720 712=value712 496=value496
8: 735=value735 927=value927 160=value160 972=value972 261=value261 916=value916
9: 238=value238 591=value591 601=value601 451=value451 175=value175 142=value142 281=value281

Enter key to find: 943
Searched for the key: 943,duration of the searching is - 57000 ns
Found: value943
Number of comparisons for adding to linked list: 6, the size is 100
Number of comparisons for searching in linked list : 15, the size is 100
Number of comparisons for searching in hash table: 1, the size is 100
Continue?
```

Тестування алгоритму

Графіки залежності часових характеристик оцінювання від розміру графіків структури



Загальною тенденцією є збільшення кількості порівнянь при зростанні розміру хеш-таблиці та кількості елементів для додавання та пошуку в зв'язаному списку. Однак, пошук в хеш-таблиці має стабільну кількість порівнянь незалежно від розміру таблиці та кількості елементів, що підкреслює перевагу використання хеш-таблиці для операцій пошуку.

4 ВИСНОВОК

Аналіз обчислювальної складності алгоритмів пошуку допоміг мені оцінити, як швидко та ефективно алгоритми можуть знаходити елементи в різних структурах даних. Я вивчила різні види алгоритмів пошуку, такі як лінійний пошук, бінарний пошук та хеш-таблиці. Розуміння того, як ці алгоритми працюють та їхній рівень ефективності, дозволило мені вибрати найбільш підходящий алгоритм для конкретного сценарію використання.
Оцінка ефективності алгоритмів на різних структурах даних була особливо цікавою, оскільки вона показала, як вибір правильної структури даних може суттєво покращити швидкість пошуку.