Федеральное агенство связи
Ордена Трудового Красного Знамени федеральное государственное
бюджетное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Математической кибернетики и
информационных технологий

Лабораторная работа №2

по дисциплине: «Методы поиска.»

Выполнил студент
группы БФИ1902
Рахимов Е.К.
Проверила:
Мосева М.С.

Москва, 2021 г.

Оглавление

1. Цель лабораторной работы 3
2. Задание на лабораторную работу 3
3. Ход лабораторной работы 3
3.1 Листинг программы 3
3.2 Результат выполнения программы 34
Список использованных источников 35

1. Цель лабораторной работы

Цель данной лабораторной работы — научиться использовать методы поиска.

2. Задание на лабораторную работу

Задание №1:

Бинарный поиск	**Бинарное дерево**	**Фибоначчиев**	**Интерполяционный**

Задание №2:

Простое рехэширование	**Рехэширование с помощью псевдослучайных чисел**	**Метод цепочек**

Задание № 3:

Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого». Подразумевается, что ферзь бьёт все клетки, расположенные по вертикалям, горизонталям и обеим диагоналям

Написать программу, которая находит хотя бы один способ решения задач.

3. Ход лабораторной работы

3.1 Листинг программы

```
package com.company;

public class Ferzi {
```

```
    static int[] chessboard = {0,0,0,0,0,0,0,0};
    static int index = 0;
    static int version = 1;

    public static void startFerzi(){

        do {
            if (checking()){
                if (index == 7) {
                    System.out.println(version++ + " [a]=" + chessboard[0] + "
[b]=" + chessboard[1] + " [c]=" + chessboard[2] + " [d]=" + chessboard[3] + "
[e]=" + chessboard[4] + " [f]=" + chessboard[5] + " [g]=" + chessboard[6] + "
[h]=" + chessboard[7]);
                    chessboard[index]++;
                }
                else {
                    index++;
                }
            }
            else {
                chessboard[index]++;
            }
        } while (chessboard[0]<8);
    }

    static boolean checking() {
        int i;

        if (index == 0) {
            return true;
        }

        if (chessboard[index]>7){
            chessboard[index] = 0;
            index--;
            return false;
        }

        for (i = 0; i < index; i++){
            if ((chessboard[index] ==
chessboard[i])|((Math.abs(chessboard[index] - chessboard[i])) == (index-i))){
                return false;
            }
        }

        return true;
    }
}
package com.company;

public class Fibonacci {
    // Сервисная функция для поиска минимума

    // из двух элементов

    public static int min(int x, int y)
    { return (x <= y)? x : y; }
```

```
    /* Возвращает индекс x, если присутствует, иначе возвращает -1 */
    public static int fibMonaccianSearch(int arr[], int x, int n)
    {

        /* Инициализировать числа Фибоначчи */

        int fibMMm2 = 0; // (м-2) -ый номер Фибоначчи

        int fibMMm1 = 1; // (m-1) '-ый номер Фибоначчи

        int fibM = fibMMm2 + fibMMm1; // м Фибоначчи

        /* fibM собирается хранить самые маленькие

        Число Фибоначчи, большее или равное n */

        while (fibM < n)

        {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;

        }
        // Отмечает удаленный диапазон спереди
        int offset = -1;
        /* пока есть элементы для проверки.

            Обратите внимание, что мы сравниваем arr [fibMm2] с x.

        Когда fibM становится 1, fibMm2 становится 0 */

        while (fibM > 1)

        {
            // Проверяем, является ли fibMm2 действительным местоположением

            int i = min(offset+fibMMm2, n-1);
            /* Если x больше значения в

            индекс fibMm2, вырезать массив подмассива

            от смещения до i */

            if (arr[i] < x)

            {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
            }

            /* Если x больше, чем значение в индексе

            fibMm2, вырезать подрешетку после i + 1 */

            else if (arr[i] > x)
```

```
        {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;

        }
            /* элемент найден. индекс возврата */

            else return i;

        }
        /* сравнение последнего элемента с x */
        if(fibMMm1 == 1 && arr[offset+1] == x)
            return offset+1;
        /* элемент не найден. возврат -1 */
        return -1;

    }

}
package com.company;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

////////////////////////////////////////////////////////////////
class Link {                                    // (could be other items)
    private int iData;                  // data item
    public Link next;                   // next link in list

    // -------------------------------------------------------------
    public Link(int it)                 // constructor
    {
        iData = it;
    }

    // -------------------------------------------------------------
    public int getKey() {
        return iData;
    }

    // -------------------------------------------------------------
    public void displayLink()           // display this link
    {
        System.out.print(iData + " ");
    }
}  // end class Link

////////////////////////////////////////////////////////////////
class SortedList {
    private Link first;                 // ref to first list item

    // -------------------------------------------------------------
    public void SortedList()            // constructor
    {
        first = null;
```

```
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert link, in order
    {
        int key = theLink.getKey();
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key > current.getKey()) {
// or current > key,
            previous = current;
            current = current.next;     // go to next item
        }
        if (previous == null)             // if beginning of list,
            first = theLink;              //    first --> new link
        else                              // not at beginning,
            previous.next = theLink;      //    prev --> new link
        theLink.next = current;           // new link --> current
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete link
    {                                 // (assumes non-empty list)
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key != current.getKey()) {
// or key == current,
            previous = current;
            current = current.next;     // go to next link
        }
        // disconnect link
        if (previous == null)             //   if beginning of list
            first = first.next;           //      delete first link
        else                              //   not at beginning
            previous.next = current.next; //    delete current link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)         // find link
    {
        Link current = first;          // start at first
        // until end of list,
        while (current != null && current.getKey() <= key) {
// or key too small,
            if (current.getKey() == key)    // is this the link?
                return current;           // found it, return link
            current = current.next;     // go to next item
        }
        return null;                   // didn't find it
    }  // end find()

    // -------------------------------------------------------------
    public void displayList() {
        System.out.print("List (first-->last): ");
        Link current = first;       // start at beginning of list
        while (current != null)      // until end of list,
```

```
        {
            current.displayLink();   // print data
            current = current.next;  // move to next link
        }
        System.out.println("");
    }
}  // end class SortedList

////////////////////////////////////////////////////////////////
class HashTable {
    private SortedList[] hashArray;   // array of lists
    private int arraySize;

    // -------------------------------------------------------------
    public HashTable(int size)        // constructor
    {
        arraySize = size;
        hashArray = new SortedList[arraySize];  // create array
        for (int j = 0; j < arraySize; j++)          // fill array
            hashArray[j] = new SortedList();     // with lists
    }

    // -------------------------------------------------------------
    public void displayTable() {
        for (int j = 0; j < arraySize; j++) // for each cell,
        {
            System.out.print(j + ". "); // display cell number
            hashArray[j].displayList(); // display list
        }
    }

    // -------------------------------------------------------------
    public int hashFunc(int key)      // hash function
    {
        return key % arraySize;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert a link
    {
        int key = theLink.getKey();
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].insert(theLink); // insert at hashVal
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete a link
    {
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].delete(key); // delete link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)         // find link
    {
        int hashVal = hashFunc(key);   // hash the key
        Link theLink = hashArray[hashVal].find(key);  // get link
        return theLink;                // return link
```

```
    }
// --------------------------------------------------------------
}  // end class HashTable

////////////////////////////////////////////////////////////////
class HashChainApp {
    public static void main(String[] args) throws IOException {
        int aKey;
        Link aDataItem;
        int size, n, keysPerCell = 100;
        // get sizes
        System.out.print("Enter size of hash table: ");
        size = getInt();
        System.out.print("Enter initial number of items: ");
        n = getInt();
        // make table
        HashTable theHashTable = new HashTable(size);

        for (int j = 0; j < n; j++)          // insert data
        {
            aKey = (int) (java.lang.Math.random() *
                    keysPerCell * size);
            aDataItem = new Link(aKey);
            theHashTable.insert(aDataItem);
        }
        while (true)                    // interact with user
        {
            System.out.print("Enter first letter of ");
            System.out.print("show, insert, delete, or find: ");
            char choice = getChar();
            switch (choice) {
                case 's':
                    theHashTable.displayTable();
                    break;
                case 'i':
                    System.out.print("Enter key value to insert: ");
                    aKey = getInt();
                    aDataItem = new Link(aKey);
                    theHashTable.insert(aDataItem);
                    break;
                case 'd':
                    System.out.print("Enter key value to delete: ");
                    aKey = getInt();
                    theHashTable.delete(aKey);
                    break;
                case 'f':
                    System.out.print("Enter key value to find: ");
                    aKey = getInt();
                    aDataItem = theHashTable.find(aKey);
                    if (aDataItem != null)
                        System.out.println("Found " + aKey);
                    else
                        System.out.println("Could not find " + aKey);
                    break;
                default:
                    System.out.print("Invalid entry\n");
            }  // end switch
        }  // end while
```

```
    }  // end main()

    //--------------------------------------------------------------
    public static String getString() throws IOException {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String s = br.readLine();
        return s;
    }

    //--------------------------------------------------------------
    public static char getChar() throws IOException {
        String s = getString();
        return s.charAt(0);
    }

    //--------------------------------------------------------------
    public static int getInt() throws IOException {
        String s = getString();
        return Integer.parseInt(s);
    }
//--------------------------------------------------------------
}  // end class HashChainApp
////////////////////////////////////////////////////////////////
package com.company;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

////////////////////////////////////////////////////////////////
class Link {                                    // (could be other items)
    private int iData;                  // data item
    public Link next;                   // next link in list

    // -------------------------------------------------------------
    public Link(int it)                 // constructor
    {
        iData = it;
    }

    // -------------------------------------------------------------
    public int getKey() {
        return iData;
    }

    // -------------------------------------------------------------
    public void displayLink()           // display this link
    {
        System.out.print(iData + " ");
    }
}  // end class Link

////////////////////////////////////////////////////////////////
class SortedList {
    private Link first;                // ref to first list item

    // -------------------------------------------------------------
```

```
    public void SortedList()           // constructor
    {
        first = null;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert link, in order
    {
        int key = theLink.getKey();
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key > current.getKey()) {
// or current > key,
            previous = current;
            current = current.next;     // go to next item
        }
        if (previous == null)             // if beginning of list,
            first = theLink;             //    first --> new link
        else                             // not at beginning,
            previous.next = theLink;     //    prev --> new link
        theLink.next = current;          // new link --> current
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)        // delete link
    {                                 // (assumes non-empty list)
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key != current.getKey()) {
// or key == current,
            previous = current;
            current = current.next;     // go to next link
        }
        // disconnect link
        if (previous == null)             //   if beginning of list
            first = first.next;          //      delete first link
        else                             //   not at beginning
            previous.next = current.next; //    delete current link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)          // find link
    {
        Link current = first;          // start at first
        // until end of list,
        while (current != null && current.getKey() <= key) {
// or key too small,
            if (current.getKey() == key)     // is this the link?
                return current;           // found it, return link
            current = current.next;     // go to next item
        }
        return null;                     // didn't find it
    }  // end find()

    // -------------------------------------------------------------
    public void displayList() {
```

```
        System.out.print("List (first-->last): ");
        Link current = first;       // start at beginning of list
        while (current != null)     // until end of list,
        {
            current.displayLink();  // print data
            current = current.next; // move to next link
        }
        System.out.println("");
    }
}  // end class SortedList

////////////////////////////////////////////////////////////////
class HashTable {
    private SortedList[] hashArray;   // array of lists
    private int arraySize;

    // -------------------------------------------------------------
    public HashTable(int size)        // constructor
    {
        arraySize = size;
        hashArray = new SortedList[arraySize];  // create array
        for (int j = 0; j < arraySize; j++)          // fill array
            hashArray[j] = new SortedList();     // with lists
    }

    // -------------------------------------------------------------
    public void displayTable() {
        for (int j = 0; j < arraySize; j++) // for each cell,
        {
            System.out.print(j + ". "); // display cell number
            hashArray[j].displayList(); // display list
        }
    }

    // -------------------------------------------------------------
    public int hashFunc(int key)      // hash function
    {
        return key % arraySize;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert a link
    {
        int key = theLink.getKey();
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].insert(theLink); // insert at hashVal
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete a link
    {
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].delete(key); // delete link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)         // find link
    {
```

```
        int hashVal = hashFunc(key);   // hash the key
        Link theLink = hashArray[hashVal].find(key);  // get link
        return theLink;                // return link
    }
// --------------------------------------------------------------
}  // end class HashTable

////////////////////////////////////////////////////////////////
class HashChainApp {
    public static void main(String[] args) throws IOException {
        int aKey;
        Link aDataItem;
        int size, n, keysPerCell = 100;
        // get sizes
        System.out.print("Enter size of hash table: ");
        size = getInt();
        System.out.print("Enter initial number of items: ");
        n = getInt();
        // make table
        HashTable theHashTable = new HashTable(size);

        for (int j = 0; j < n; j++)          // insert data
        {
            aKey = (int) (java.lang.Math.random() *
                    keysPerCell * size);
            aDataItem = new Link(aKey);
            theHashTable.insert(aDataItem);
        }
        while (true)                     // interact with user
        {
            System.out.print("Enter first letter of ");
            System.out.print("show, insert, delete, or find: ");
            char choice = getChar();
            switch (choice) {
                case 's':
                    theHashTable.displayTable();
                    break;
                case 'i':
                    System.out.print("Enter key value to insert: ");
                    aKey = getInt();
                    aDataItem = new Link(aKey);
                    theHashTable.insert(aDataItem);
                    break;
                case 'd':
                    System.out.print("Enter key value to delete: ");
                    aKey = getInt();
                    theHashTable.delete(aKey);
                    break;
                case 'f':
                    System.out.print("Enter key value to find: ");
                    aKey = getInt();
                    aDataItem = theHashTable.find(aKey);
                    if (aDataItem != null)
                        System.out.println("Found " + aKey);
                    else
                        System.out.println("Could not find " + aKey);
                    break;
                default:
```

```
                    System.out.print("Invalid entry\n");
                }  // end switch
            }  // end while
        }  // end main()

        //--------------------------------------------------------------
        public static String getString() throws IOException {
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader br = new BufferedReader(isr);
            String s = br.readLine();
            return s;
        }

        //--------------------------------------------------------------
        public static char getChar() throws IOException {
            String s = getString();
            return s.charAt(0);
        }

        //--------------------------------------------------------------
        public static int getInt() throws IOException {
            String s = getString();
            return Integer.parseInt(s);
        }
    //--------------------------------------------------------------
    }  // end class HashChainApp
    ////////////////////////////////////////////////////////////////
    package com.company;

    import java.io.BufferedReader;
    import java.io.IOException;
    import java.io.InputStreamReader;

    ////////////////////////////////////////////////////////////////
    class Link {                                    // (could be other items)
        private int iData;                  // data item
        public Link next;                   // next link in list

        // -------------------------------------------------------------
        public Link(int it)                 // constructor
        {
            iData = it;
        }

        // -------------------------------------------------------------
        public int getKey() {
            return iData;
        }

        // -------------------------------------------------------------
        public void displayLink()           // display this link
        {
            System.out.print(iData + " ");
        }
    }  // end class Link

    ////////////////////////////////////////////////////////////////
    class SortedList {
```

```
    private Link first;               // ref to first list item

    // -------------------------------------------------------------
    public void SortedList()          // constructor
    {
        first = null;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert link, in order
    {
        int key = theLink.getKey();
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key > current.getKey()) {
// or current > key,
            previous = current;
            current = current.next;     // go to next item
        }
        if (previous == null)             // if beginning of list,
            first = theLink;             //    first --> new link
        else                             // not at beginning,
            previous.next = theLink;     //    prev --> new link
        theLink.next = current;          // new link --> current
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete link
    {                                 // (assumes non-empty list)
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key != current.getKey()) {
// or key == current,
            previous = current;
            current = current.next;     // go to next link
        }
        // disconnect link
        if (previous == null)             //   if beginning of list
            first = first.next;          //      delete first link
        else                             //   not at beginning
            previous.next = current.next; //    delete current link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)         // find link
    {
        Link current = first;          // start at first
        // until end of list,
        while (current != null && current.getKey() <= key) {
// or key too small,
            if (current.getKey() == key)    // is this the link?
                return current;           // found it, return link
            current = current.next;     // go to next item
        }
        return null;                   // didn't find it
    }  // end find()
```

```
    // -------------------------------------------------------------
    public void displayList() {
        System.out.print("List (first-->last): ");
        Link current = first;       // start at beginning of list
        while (current != null)     // until end of list,
        {
            current.displayLink();  // print data
            current = current.next; // move to next link
        }
        System.out.println("");
    }
}  // end class SortedList

////////////////////////////////////////////////////////////////
class HashTable {
    private SortedList[] hashArray;   // array of lists
    private int arraySize;

    // -------------------------------------------------------------
    public HashTable(int size)        // constructor
    {
        arraySize = size;
        hashArray = new SortedList[arraySize];  // create array
        for (int j = 0; j < arraySize; j++)          // fill array
            hashArray[j] = new SortedList();     // with lists
    }

    // -------------------------------------------------------------
    public void displayTable() {
        for (int j = 0; j < arraySize; j++) // for each cell,
        {
            System.out.print(j + ". "); // display cell number
            hashArray[j].displayList(); // display list
        }
    }

    // -------------------------------------------------------------
    public int hashFunc(int key)      // hash function
    {
        return key % arraySize;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert a link
    {
        int key = theLink.getKey();
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].insert(theLink); // insert at hashVal
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete a link
    {
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].delete(key); // delete link
    }  // end delete()
```

```
    // --------------------------------------------------------------
    public Link find(int key)          // find link
    {
        int hashVal = hashFunc(key);   // hash the key
        Link theLink = hashArray[hashVal].find(key);  // get link
        return theLink;                // return link
    }
// --------------------------------------------------------------
}  // end class HashTable

////////////////////////////////////////////////////////////////
class HashChainApp {
    public static void main(String[] args) throws IOException {
        int aKey;
        Link aDataItem;
        int size, n, keysPerCell = 100;
        // get sizes
        System.out.print("Enter size of hash table: ");
        size = getInt();
        System.out.print("Enter initial number of items: ");
        n = getInt();
        // make table
        HashTable theHashTable = new HashTable(size);

        for (int j = 0; j < n; j++)          // insert data
        {
            aKey = (int) (java.lang.Math.random() *
                    keysPerCell * size);
            aDataItem = new Link(aKey);
            theHashTable.insert(aDataItem);
        }
        while (true)                         // interact with user
        {
            System.out.print("Enter first letter of ");
            System.out.print("show, insert, delete, or find: ");
            char choice = getChar();
            switch (choice) {
                case 's':
                    theHashTable.displayTable();
                    break;
                case 'i':
                    System.out.print("Enter key value to insert: ");
                    aKey = getInt();
                    aDataItem = new Link(aKey);
                    theHashTable.insert(aDataItem);
                    break;
                case 'd':
                    System.out.print("Enter key value to delete: ");
                    aKey = getInt();
                    theHashTable.delete(aKey);
                    break;
                case 'f':
                    System.out.print("Enter key value to find: ");
                    aKey = getInt();
                    aDataItem = theHashTable.find(aKey);
                    if (aDataItem != null)
                        System.out.println("Found " + aKey);
                    else
```

```
                        System.out.println("Could not find " + aKey);
                    break;
                default:
                    System.out.print("Invalid entry\n");
            }  // end switch
        }  // end while
    }  // end main()

    //--------------------------------------------------------------
    public static String getString() throws IOException {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String s = br.readLine();
        return s;
    }

    //--------------------------------------------------------------
    public static char getChar() throws IOException {
        String s = getString();
        return s.charAt(0);
    }

    //--------------------------------------------------------------
    public static int getInt() throws IOException {
        String s = getString();
        return Integer.parseInt(s);
    }
//--------------------------------------------------------------
}  // end class HashChainApp
////////////////////////////////////////////////////////////////
package com.company;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

////////////////////////////////////////////////////////////////
class Link {                                    // (could be other items)
    private int iData;                  // data item
    public Link next;                   // next link in list

    // -------------------------------------------------------------
    public Link(int it)                 // constructor
    {
        iData = it;
    }

    // -------------------------------------------------------------
    public int getKey() {
        return iData;
    }

    // -------------------------------------------------------------
    public void displayLink()           // display this link
    {
        System.out.print(iData + " ");
    }
}  // end class Link
```

```
////////////////////////////////////////////////////////////////
class SortedList {
    private Link first;               // ref to first list item

    // -------------------------------------------------------------
    public void SortedList()          // constructor
    {
        first = null;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert link, in order
    {
        int key = theLink.getKey();
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key > current.getKey()) {
// or current > key,
            previous = current;
            current = current.next;     // go to next item
        }
        if (previous == null)             // if beginning of list,
            first = theLink;             //    first --> new link
        else                             // not at beginning,
            previous.next = theLink;     //    prev --> new link
        theLink.next = current;          // new link --> current
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete link
    {                                 // (assumes non-empty list)
        Link previous = null;          // start at first
        Link current = first;
        // until end of list,
        while (current != null && key != current.getKey()) {
// or key == current,
            previous = current;
            current = current.next;     // go to next link
        }
        // disconnect link
        if (previous == null)             //   if beginning of list
            first = first.next;          //      delete first link
        else                             //   not at beginning
            previous.next = current.next; //    delete current link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)         // find link
    {
        Link current = first;          // start at first
        // until end of list,
        while (current != null && current.getKey() <= key) {
// or key too small,
            if (current.getKey() == key)    // is this the link?
                return current;          // found it, return link
            current = current.next;     // go to next item
```

```
        }
        return null;                    // didn't find it
    }  // end find()

    // -------------------------------------------------------------
    public void displayList() {
        System.out.print("List (first-->last): ");
        Link current = first;       // start at beginning of list
        while (current != null)      // until end of list,
        {
            current.displayLink();   // print data
            current = current.next;  // move to next link
        }
        System.out.println("");
    }
}  // end class SortedList

////////////////////////////////////////////////////////////////
class HashTable {
    private SortedList[] hashArray;   // array of lists
    private int arraySize;

    // -------------------------------------------------------------
    public HashTable(int size)        // constructor
    {
        arraySize = size;
        hashArray = new SortedList[arraySize];  // create array
        for (int j = 0; j < arraySize; j++)          // fill array
            hashArray[j] = new SortedList();     // with lists
    }

    // -------------------------------------------------------------
    public void displayTable() {
        for (int j = 0; j < arraySize; j++) // for each cell,
        {
            System.out.print(j + ". "); // display cell number
            hashArray[j].displayList(); // display list
        }
    }

    // -------------------------------------------------------------
    public int hashFunc(int key)      // hash function
    {
        return key % arraySize;
    }

    // -------------------------------------------------------------
    public void insert(Link theLink)  // insert a link
    {
        int key = theLink.getKey();
        int hashVal = hashFunc(key);   // hash the key
        hashArray[hashVal].insert(theLink); // insert at hashVal
    }  // end insert()

    // -------------------------------------------------------------
    public void delete(int key)       // delete a link
    {
        int hashVal = hashFunc(key);   // hash the key
```

```
        hashArray[hashVal].delete(key); // delete link
    }  // end delete()

    // -------------------------------------------------------------
    public Link find(int key)         // find link
    {
        int hashVal = hashFunc(key);   // hash the key
        Link theLink = hashArray[hashVal].find(key);  // get link
        return theLink;                // return link
    }
// -------------------------------------------------------------
}  // end class HashTable

////////////////////////////////////////////////////////////////
class HashChainApp {
    public static void main(String[] args) throws IOException {
        int aKey;
        Link aDataItem;
        int size, n, keysPerCell = 100;
        // get sizes
        System.out.print("Enter size of hash table: ");
        size = getInt();
        System.out.print("Enter initial number of items: ");
        n = getInt();
        // make table
        HashTable theHashTable = new HashTable(size);

        for (int j = 0; j < n; j++)          // insert data
        {
            aKey = (int) (java.lang.Math.random() *
                    keysPerCell * size);
            aDataItem = new Link(aKey);
            theHashTable.insert(aDataItem);
        }
        while (true)                    // interact with user
        {
            System.out.print("Enter first letter of ");
            System.out.print("show, insert, delete, or find: ");
            char choice = getChar();
            switch (choice) {
                case 's':
                    theHashTable.displayTable();
                    break;
                case 'i':
                    System.out.print("Enter key value to insert: ");
                    aKey = getInt();
                    aDataItem = new Link(aKey);
                    theHashTable.insert(aDataItem);
                    break;
                case 'd':
                    System.out.print("Enter key value to delete: ");
                    aKey = getInt();
                    theHashTable.delete(aKey);
                    break;
                case 'f':
                    System.out.print("Enter key value to find: ");
                    aKey = getInt();
                    aDataItem = theHashTable.find(aKey);
```

```
                    if (aDataItem != null)
                        System.out.println("Found " + aKey);
                    else
                        System.out.println("Could not find " + aKey);
                    break;
                default:
                    System.out.print("Invalid entry\n");
            }  // end switch
        }  // end while
    }  // end main()

    //-------------------------------------------------------------
    public static String getString() throws IOException {
        InputStreamReader isr = new InputStreamReader(System.in);
        BufferedReader br = new BufferedReader(isr);
        String s = br.readLine();
        return s;
    }

    //-------------------------------------------------------------
    public static char getChar() throws IOException {
        String s = getString();
        return s.charAt(0);
    }

    //-------------------------------------------------------------
    public static int getInt() throws IOException {
        String s = getString();
        return Integer.parseInt(s);
    }
//-------------------------------------------------------------
}  // end class HashChainApp
////////////////////////////////////////////////////////////////
package com.company;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {
   int [] array = StartGenerate();
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i]+" ");
        }
        System.out.println();
        System.out.println("/////");
        StartTreeFind(array);
        System.out.println("/////");
        binarySearch(array,0,array.length,  array[3]);
        System.out.println("/////");
        System.out.println("На позиции = " +fibFind(array,array[3]));
        System.out.println("/////");
        System.out.println("На позиции = " +(InterpolSearch(array,array[3])+1));

       //ReHash.StartReHashProst();
        //ReHash.StartReHashPsevd();

        //Ferzi.startFerzi();
```

```
    }
    public static int rnd(int min, int max){
        max -= min;
        return (int) (Math.random() * ++max) + min;
    }
    public static int [] StartGenerate(){
        int [] mas = new int[14];
        for (int i = 0; i <mas.length; i++) {
            mas[i]= rnd(0,130);
        }
        Arrays.sort(mas);
        return mas;
    }
    public static int [] insertElement(int[] array, int key, int posytion){
        int [] arrayNew = new int[array.length+1];
        int i=0;
        posytion--;
        while(i!=posytion){
            arrayNew[i]=array[i];
            i++;
        }
        arrayNew[posytion]=key;
        i++;
        while(i!=arrayNew.length-1){
            arrayNew[i+1]=array[i];
            i++;
        }
        return arrayNew;
    }
    public static void deleteElement(int[] array, int posytion ){
        for (int i = posytion-1; i <array.length-1; i++) {
            array[i]=array[i+1];
        }
    }
    public static void StartTreeFind(int [] array){
        Tree tree = new Tree();
        // вставляем узлы в дерево:
        for (int i = 0; i < array.length; i++) {
            tree.insertNode(array[i]);
        }


        // находим узел по значению и выводим его в консоли
        Node foundNode = tree.findNodeByValue(array[3]);
        foundNode.printNode();
    }
    public static void binarySearch(int[] array, int first, int last, int item)
{
        int position;
        int comparisonCount = 1;     // для подсчета количества сравнений

        // для начала найдем индекс среднего элемента массива
        position = (first + last) / 2;

        while ((array[position] != item) && (first <= last)) {
            comparisonCount++;
            if (array[position] > item) {  // если число заданного для поиска
```

```
                last = position - 1; // уменьшаем позицию на 1.
            } else {
                first = position + 1;    // иначе увеличиваем на 1
            }
            position = (first + last) / 2;
        }
        if (first <= last) {
            System.out.println(item + " является " + ++position + " элементом в
массиве");
            System.out.println("Метод бинарного поиска нашел число после " +
comparisonCount +
                    " сравнений");
        } else {
            System.out.println("Элемент не найден в массиве. Метод бинарного
поиска закончил работу после "
                    + comparisonCount + " сравнений");
        }
    }
    public static int fibFind(int [] array,int key){
        int f=1,s=1,i=f+s;
        if(array[0]==key)
            return 0;
        if(array[1]==key)
            return 1;
        while (key>array[i]){
            f=s;
            s=i;
            i=s+f;
            if(i>=array.length){
                break;
            }
        }
        if(array[i]==key)
            return i+1;
        else {
            int f1=0,s1=1,i1=f1+s1;
            while (array[s+i1]<key){
                f1=s;
                s1=i;
                i1=s+f;
                if(s+i1> array.length){
                    i=-1;
                    break;
                }
            }

        }
        return i;
    }
    public static int InterpolSearch(int A[], int key){
        int mid, left=0, right=A.length-1;
        while (A[left]<=key && A[right]>=key)
        {
            mid=left+((key-A[left])*(right-left))/(A[right]-A[left]);
            if (A[mid]<key) left=mid+1;
            else if (A[mid]>key) right=mid-1;
            else return mid;
        }
```

```
        if (A[left]==key) return left;
        else return -1;
    }
 }
package com.company;

class Node {
    private int value; // ключ узла
    private Node leftChild; // Левый узел потомок
    private Node rightChild; // Правый узел потомок

    public void printNode() { // Вывод значения узла в консоль
        System.out.println(" Выбранный узел имеет значение :" + value);
    }

    public int getValue() {
        return this.value;
    }

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

    public Node getLeftChild() {
        return this.leftChild;
    }

    public void setLeftChild(final Node leftChild) {
        this.leftChild = leftChild;
    }

    public Node getRightChild() {
        return this.rightChild;
    }

    public void setRightChild(final Node rightChild) {
        this.rightChild = rightChild;
    }

    @Override
    public String toString() {
        return "Node{" +
                "value=" + value +
                ", leftChild=" + leftChild +
                ", rightChild=" + rightChild +
                '}';
    }
}
package com.company;

import java.util.Arrays;
import java.util.Scanner;

public class ReHash {
    static int MaxN = 10; //Размерность таблицы
    static String[] mas = new String[MaxN];
    static int [] bufMas = new int[MaxN];
    static boolean[] masFlag = new boolean[MaxN];
```

```
    static String sstr;
    static int j,c,n;

    public static void initArray(){

        //Процедура инициализации массива (хеш-таблицы).
        //Массив типа string, '' - пустая ячейка.

    int j;

    for ( j = 0; j <MaxN ; j++)
    {
    mas[j]= "";
    masFlag[j]= true;
    } }
    public static void printMas(){
        for (int i = 0; i <MaxN; i++) {
            System.out.println("i = "+i+" mas = "+mas[i] +" bufMas ="+bufMas[i]
);
        }
    }
    public static int hash(String str) {
        int hash;
        hash = Integer.parseInt(str)%MaxN;
        return hash;
    }
    public static int rhash(int ii, int c, String str) {
        //Разрешение коллизий. Подбирает новый адрес для элемента,
        //если место, которое ему определила хеш-функция, занято.
        //ii - ii-й элемент последовательности проб;
        //c - фиксированный шаг;
        //str - текущий элемент.
        //Прим. Для того, чтобы все ячейки оказались просмотренными по одному
разу,
        //необходимо, чтобы "c" было взаимно-простым с размером хеш-таблицы
(maxn).
        //например, такой метод - линейное пробирование
        int srhash = (hash(str) + c * ii) % MaxN;
        return srhash;
    }

    public static int rhashPsevd(int ii, String str) {

        int srhash = (hash(str) + Main.rnd(1,9) * ii) % MaxN;
        return srhash;
    }
    public static void AddHash(int j, int c, int n) {
        String str;
        //Процедура добавления элемента в таблицу. Здесь c - шаг,
        // j - счетчик элементов, str - вводимая строка n - сколько свобод ячеек

        int f=0;
        int i, ii;
        //В случае, если потребуется подбирать свободное
        //        место для элемента в таблице, начинать всегда
        //        нужно с начала. Поэтому i=0
```

```
        Scanner scanner = new Scanner(System.in);

        i = 0;  //переменная для rhash

        if (n == 0)
            System.out.println("Таблица заполнена");
        else {

           // j++; // счетчик для элементов
            System.out.println("Элемент " + j);
            str = scanner.nextLine();
            bufMas[j]=Integer.parseInt(str);
            ii = hash(str);  //получения адреса для хранения элемента в хеш-
таблице
            while (true) {

                //ячейка по определенному хеш-функцией адресу пуста
                if (ii <= MaxN) {

                    if (mas[ii].equals("") ) {

                        mas[ii] = str;
                        //добавляем элемент в ячейку
                        masFlag[ii] = false;
                        //помечаем как занятую
                        n--;
                        break;
                        //выходим для дальшейших операций
                    }
                    //по указанному адресу лежит элемент-клон
                    if ((((!mas[ii].equals("")) && (mas[ii].equals(str))) ) {

                        mas[ii] = str;  //заменяем его (они ведь одинаковы)
                        masFlag[ii] = false; // помечаем ячейку как занятую
                        j--;  //элемент заменился, значит счетчик не увеличиваем
                        break;  //выходим для дальшейших операций
                    }
                    //по указанному адресу уже есть элемент, отдичный от
текущего
                    if ((!mas[ii].equals("") && (!mas[ii].equals(str))) )

                        ii = rhash(i, c, str);  //меняем адрес элементу во
избежание коллизии

                    //проверяем таблицу на наличие свободного места
                } else {
                    ii = rhash(i, c, str);

                     // в случае, если адрес, выданный rhash, тоже занят,
продолжаем поиск

                }
                i++;
            }


        }
```

```
    printMas();
    }
    public static void DeleteHash(int j, int n,int c){
        //Процедура удаления элемента по адресу в таблице.
        //Здесь j - номер вводимого элемента, с -шаг

    String str;
    int i,ii ;
    i =1;
    //переменная для rhash
        System.out.println("Введите элемент для удаления:");
        Scanner scanner = new Scanner(System.in);
    str=scanner.nextLine();
    //вводим элемент
    ii = hash(str);
    //вычисляем его адрес



  while (true) {

    if (ii <=MaxN) {
        //ищем в таблице
        if (mas[ii].equals("") || ( mas[ii].equals(str) && masFlag[ii]))
        {

            System.out.println("Элемент не найден");
            break;
        }
         //находим

        if (mas[ii].equals( str)  && !masFlag[ii]) {
            //Помечаем ячейку как свободную.Элемент при этом не удаляем.
            //Добавляемый в нее элемент просто перезапишет пред.значение


            masFlag[ii]=true;

bufMas[Arrays.binarySearch(bufMas,Integer.parseInt(str))]=Integer.MIN_VALUE;//де
лаю метку, какой удален
            System.out.println("Элемент удален");
            n++;
            j--;
            //элемент удален, уменьшаем счетчик
            break;
        }
    }
    //Поиск другого адреса.Это значит, что элемент был
      //        добавлен в таблицу с применением функции rhash, т.е.выдачей
      //        ему нового адреса из -за конфликта с другим элементом.

    ii = rhash(i, c, str);

    i++;
    //переменная для rhash
    }
  printMas();
}
```

```
  public static void SearchHash(int c)    {
      //Процедура для поиска элемента в таблице по хеш-адресу.
      //Здесь c - шаг
      String  str;
  int i, ii;

  i = 1;  //переменная для rhash
      System.out.println("Введите элемент для поиска: ");
      Scanner scanner = new Scanner(System.in);
  str=scanner.nextLine();//вводим элемент для поиска
  ii = hash(str);  //вычисляем его адрес в таблице
while (true) {

  if (ii <= MaxN) {

   //ищем
    if (mas[ii].equals("") ) {
        System.out.println("Элемент не найден");
      break;
    }

  //нашли
    if (mas[ii].equals( str)){
        System.out.println("Адрес: "+ ii+" сравнений: "+ i);
      break;  //прекращаем поиск
    }
  }
      //Поиск другого адреса. Это значит, что элемент был
    //        добавлен в таблицу с применением функции rhash, т.е. выдачей
    //        ему нового адреса из-за конфликта с другим элементом.

  ii = rhash(i, c, str);
      //rhash присвоил недопустимый адрес
    //        для ячейки или ячейка со стандартным адресом пуста
  i++;
  //переменная для rhash

}
printMas();
  }
  public static void SearchHashPsevd(int c)    {
      //Процедура для поиска элемента в таблице по хеш-адресу.
      //Здесь c - шаг
      String  str;
      int i, ii;

      i = 1;  //переменная для rhash
      System.out.println("Введите элемент для поиска: ");
      Scanner scanner = new Scanner(System.in);
      str=scanner.nextLine();//вводим элемент для поиска
      ii = hash(str);  //вычисляем его адрес в таблице
      while (true) {

          if (ii <= MaxN) {

              //ищем
              if (mas[ii].equals("") ) {
                  System.out.println("Элемент не найден");
```

```
                    break;
                }

                //нашли
                if (mas[ii].equals( str)){
                    System.out.println("Адрес: "+ ii+" сравнений: "+ i);
                    break;  //прекращаем поиск
                }
            }
            //Поиск другого адреса. Это значит, что элемент был
            //        добавлен в таблицу с применением функции rhash, т.е.
выдачей
            //        ему нового адреса из-за конфликта с другим элементом.

            ii = rhashPsevd(i, str);
            //rhash присвоил недопустимый адрес
            //        для ячейки или ячейка со стандартным адресом пуста
            i++;
            //переменная для rhash

        }
        printMas();
    }
    public static void StartReHashProst(){
    initArray();
        System.out.println("init");
        for (int i = 0; i <10; i++) {
            AddHash(i,1,10);
        }
    SearchHash(1);

    }
    public static void StartReHashPsevd(){
        initArray();
        System.out.println("init");
        for (int i = 0; i <10; i++) {
            AddHash(i,1,10);
        }
        SearchHashPsevd(1);

    }

}


package com.company;

import java.util.Stack;

public class Tree {
    private Node rootNode; // корневой узел

    public Tree() { // Пустое дерево
        rootNode = null;
    }

    public Node findNodeByValue(int value) { // поиск узла по значению
        Node currentNode = rootNode; // начинаем поиск с корневого узла
```

```
        while (currentNode.getValue() != value) { // поиск покуда не будет
найден элемент или не будут перебраны все
            if (value < currentNode.getValue()) { // движение влево?
                currentNode = currentNode.getLeftChild();
            } else { //движение вправо
                currentNode = currentNode.getRightChild();
            }
            if (currentNode == null) { // если потомка нет,
                return null; // возвращаем null
            }
        }
        return currentNode; // возвращаем найденный элемент
    }

    public void insertNode(int value) { // метод вставки нового элемента
        Node newNode = new Node(); // создание нового узла
        newNode.setValue(value); // вставка данных
        if (rootNode == null) { // если корневой узел не существует
            rootNode = newNode;// то новый элемент и есть корневой узел
        }
        else { // корневой узел занят
            Node currentNode = rootNode; // начинаем с корневого узла
            Node parentNode;
            while (true) // мы имеем внутренний выход из цикла
            {
                parentNode = currentNode;
                if(value == currentNode.getValue()) {   // если такой элемент в
дереве уже есть, не сохраняем его
                    return;    // просто выходим из метода
                }
                else  if (value < currentNode.getValue()) {   // движение влево?
                    currentNode = currentNode.getLeftChild();
                    if (currentNode == null){ // если был достигнут конец
цепочки,
                        parentNode.setLeftChild(newNode); //  то вставить слева
и выйти из методы
                        return;
                    }
                }
                else { // Или направо?
                    currentNode = currentNode.getRightChild();
                    if (currentNode == null) { // если был достигнут конец
цепочки,
                        parentNode.setRightChild(newNode);  //то вставить справа
                        return; // и выйти
                    }
                }
            }
        }
    }

    public boolean deleteNode(int value) // Удаление узла с заданным ключом
    {
        Node currentNode = rootNode;
        Node parentNode = rootNode;
        boolean isLeftChild = true;
        while (currentNode.getValue() != value) { // начинаем поиск узла
            parentNode = currentNode;
```

```
            if (value < currentNode.getValue()) { // Определяем, нужно ли
движение влево?
                isLeftChild = true;
                currentNode = currentNode.getLeftChild();
            }
            else { // или движение вправо?
                isLeftChild = false;
                currentNode = currentNode.getRightChild();
            }
            if (currentNode == null)
                return false; // узел не найден
        }

        if (currentNode.getLeftChild() == null && currentNode.getRightChild() ==
null) { // узел просто удаляется, если не имеет потомков
            if (currentNode == rootNode) // если узел - корень, то дерево
очищается
                rootNode = null;
            else if (isLeftChild)
                parentNode.setLeftChild(null); // если нет - узел отсоединяется,
от родителя
            else
                parentNode.setRightChild(null);
        }
        else if (currentNode.getRightChild() == null) { // узел заменяется левым
поддеревом, если правого потомка нет
            if (currentNode == rootNode)
                rootNode = currentNode.getLeftChild();
            else if (isLeftChild)
                parentNode.setLeftChild(currentNode.getLeftChild());
            else
                parentNode.setRightChild(currentNode.getLeftChild());
        }
        else if (currentNode.getLeftChild() == null) { // узел заменяется правым
поддеревом, если левого потомка нет
            if (currentNode == rootNode)
                rootNode = currentNode.getRightChild();
            else if (isLeftChild)
                parentNode.setLeftChild(currentNode.getRightChild());
            else
                parentNode.setRightChild(currentNode.getRightChild());
        }
        else { // если есть два потомка, узел заменяется преемником
            Node heir = receiveHeir(currentNode);// поиск преемника для
удаляемого узла
            if (currentNode == rootNode)
                rootNode = heir;
            else if (isLeftChild)
                parentNode.setLeftChild(heir);
            else
                parentNode.setRightChild(heir);
        }
        return true; // элемент успешно удалён
    }

    // метод возвращает узел со следующим значением после передаваемого
аргументом.
    // для этого он сначала переходим к правому потомку, а затем
```

```
    // отслеживаем цепочку левых потомков этого узла.
    private Node receiveHeir(Node node) {
        Node parentNode = node;
        Node heirNode = node;
        Node currentNode = node.getRightChild(); // Переход к правому потомку
        while (currentNode != null) // Пока остаются левые потомки
        {
            parentNode = heirNode;// потомка задаём как текущий узел
            heirNode = currentNode;
            currentNode = currentNode.getLeftChild(); // переход к левому
потомку
        }
        // Если преемник не является
        if (heirNode != node.getRightChild()) // правым потомком,
        { // создать связи между узлами
            parentNode.setLeftChild(heirNode.getRightChild());
            heirNode.setRightChild(node.getRightChild());
        }
        return heirNode;// возвращаем приемника
    }

   /* public void printTree() { // метод для вывода дерева в консоль
        Stack globalStack = new Stack(); // общий стек для значений дерева
        globalStack.push(rootNode);
        int gaps = 32; // начальное значение расстояния между элементами
        boolean isRowEmpty = false;
        String separator = "-----------------------------------------------------
-------------";
        System.out.println(separator);// черта для указания начала нового дерева
        while (isRowEmpty == false) {
            Stack localStack = new Stack(); // локальный стек для задания
потомков элемента
            isRowEmpty = true;

            for (int j = 0; j < gaps; j++)
                System.out.print(' ');
            while (globalStack.isEmpty() == false) { // покуда в общем стеке
есть элементы
                Node temp = (Node) globalStack.pop(); // берем следующий, при
этом удаляя его из стека
                if (temp != null) {
                    System.out.print(temp.getValue()); // выводим его значение в
консоли
                    localStack.push(temp.getLeftChild()); // сохраняем в
локальный стек, наследники текущего элемента
                    localStack.push(temp.getRightChild());
                    if (temp.getLeftChild() != null ||
                            temp.getRightChild() != null)
                        isRowEmpty = false;
                }
                else {
                    System.out.print("__");// - если элемент пустой
                    localStack.push(null);
                    localStack.push(null);
                }
                for (int j = 0; j < gaps * 2 - 2; j++)
                    System.out.print(' ');
            }
```

```
            System.out.println();
            gaps /= 2;// при переходе на следующий уровень расстояние между
элементами каждый раз уменьшается
            while (localStack.isEmpty() == false)
                globalStack.push(localStack.pop()); // перемещаем все элементы
из локального стека в глобальный
        }
        System.out.println(separator);// подводим черту
    }

    */}
```

3.2 Результат выполнения программы

```
7 14 18 21 33 54 56 59 64 74 87 116 119 130
/////
 Выбранный узел имеет значение :21
/////
21 является 4 элементом в массиве
Метод бинарного поиска нашел число после 2 сравнений
/////
На позиции = 4
/////
На позиции = 4
init
Элемент 0
```

Рисунок 1 – результат выполнения

Список использованных источников

1) ГОСТ 7.32-2017 Система стандартов по информации, библиотечному и издательскому делу. Отчёт о научно-исследовательской работе. Структура и правила оформления

2) ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание. Общие требования и правила составления