Федеральное агентство связи

Ордена трудового красного знамени

Федеральное государственное образовательное

Бюджетное

Учреждение высшего профессионального образования

Московский Технический Университет связи и информатики

Кафедра информатики

Лабораторная работа №2

по дисциплине «СиАОД»

«Методы поиска»

Выполнил: студ. гр. БСТ1902

Потрываев А.Г

Вариант №15

Проверил: Кутейников И.А.

Москва 2021

Содержание

1 Задание

Реализовать методы поиска в соответствии с заданием. Организовать генерацию начального набора случайных данных. Для всех вариантов добавить реализацию добавления, поиска и удаления элементов. Оценить время работы каждого алгоритма поиска и сравнить его со временем работы стандартной функции поиска, используемой в выбранном языке программирования

Задание 1 – Бинарный поиск, Бинарное дерево, Фиббоначиев, Интерполяционный.

Задание 2 – Простое рехеширование, Рехеширование 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

2 Ход выполнения работы

Задание 1

Листинг бинарного поиска

```
public class BinarySearch{


   public static void search(int[] array, int first, int last, int item) {
      int comparisonCount = 1;

      for(int position = (first + last) / 2 ; array[position]!=item && first <= last; position = (first + last) / 2) {
         ++comparisonCount;
         if (array[position] > item) {
            last = position - 1;
         } else {
            first = position + 1;
         }
      }

      if (first <= last) {
         System.out.println("Бинарный поиск. Индификатор найден. Количество проверок: " + comparisonCount);
      } else {
         System.out.println("Бинарный поиск. Индификатор не найден. Количество проверок: " + comparisonCount);
      }

   }

}
```

Листинг бинарного дерева

```
public class BinaryTree {

   public static class Node {
```

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


    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;
    }
}

private Node rootNode; // корневой узел

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

public BinaryTree(int[] arr){
    for (int i : arr)
        insert(i);
}
```

```
    public void find(int value) {
        Node currentNode = rootNode;
        int comparisonCount = 1;

        while (currentNode != null) {
            if(currentNode.getValue() == value)
                System.out.println("Бинарное дерево. Индификатор найден.
Количество проверок: " + comparisonCount);
            if (value < currentNode.getValue()) {
                comparisonCount++;
                currentNode = currentNode.getLeftChild();
            } else {
                comparisonCount++;
                currentNode = currentNode.getRightChild();
            }
            if (currentNode == null) {
                System.out.println("Бинарное дерево. Индификатор не найден.
Количество проверок: " + comparisonCount);
            }
        }
    }

    public void insert(int value) {
        Node newNode = new Node();
        newNode.setValue(value);
        if (rootNode == null) {
            rootNode = newNode;
        } else {
            Node currentNode = rootNode;
            Node parentNode;
            while (true)
                {
                    parentNode = currentNode;
                    if (value == currentNode.getValue()) {
                        return;
                    } else if (value < currentNode.getValue()) {
                        currentNode = currentNode.getLeftChild();
                        if (currentNode == null) {
                            parentNode.setLeftChild(newNode);
                            return;
                        }
```

```
                } else {
                    currentNode = currentNode.getRightChild();
                    if (currentNode == null) {
                        parentNode.setRightChild(newNode);
                        return;
                    }
                }
            }
        }
    }

    public boolean delete(int value)
    {
        Node currentNode = rootNode;
        Node parentNode = rootNode;
        boolean isLeftChild = true;
        while (currentNode.getValue() != value) {
            parentNode = currentNode;
            if (value < currentNode.getValue()) {
                isLeftChild = true;
                currentNode = currentNode.getLeftChild();
            } else {
                isLeftChild = false;
                currentNode = currentNode.getRightChild();
            }
            if (currentNode == null)
                return false;
        }

        if (currentNode.getLeftChild() == null && currentNode.getRightChild()
== null) {
            if (currentNode == rootNode)
                rootNode = null;
            else if (isLeftChild)
                parentNode.setLeftChild(null);
            else
                parentNode.setRightChild(null);
        } else if (currentNode.getRightChild() == null) {
            if (currentNode == rootNode)
                rootNode = currentNode.getLeftChild();
            else if (isLeftChild)
```

```
            parentNode.setLeftChild(currentNode.getLeftChild());
         else
            parentNode.setRightChild(currentNode.getLeftChild());
      } else if (currentNode.getLeftChild() == null) {
         if (currentNode == rootNode)
            rootNode = currentNode.getRightChild();
         else if (isLeftChild)
            parentNode.setLeftChild(currentNode.getRightChild());
         else
            parentNode.setRightChild(currentNode.getRightChild());
      } else {
         Node heir = receiveHeir(currentNode);
         if (currentNode == rootNode)
            rootNode = heir;
         else if (isLeftChild)
            parentNode.setLeftChild(heir);
         else
            parentNode.setRightChild(heir);
      }
      return true;
   }


   private Node receiveHeir(Node node) {
      Node parentNode = node;
      Node heirNode = node;
      Node currentNode = node.getRightChild();
      while (currentNode != null)
      {
         parentNode = heirNode;
         heirNode = currentNode;
         currentNode = currentNode.getLeftChild();
      }
      if (heirNode != node.getRightChild())
      {
         parentNode.setLeftChild(heirNode.getRightChild());
         heirNode.setRightChild(node.getRightChild());
      }
      return heirNode;
   }
}
```

Листинг интерполяционного поиска

```
public class Interpolation {
   public Interpolation() {
   }

   public static void interpolationSearch(int[] sortedArray, int toFind) {
      int comparisonCount = 0;
      int low = 0;
      int high = sortedArray.length - 1;

      while(sortedArray[low] < toFind && sortedArray[high] > toFind) {
         comparisonCount++;
         if (sortedArray[high] == sortedArray[low]) {
            break;
         }

         int mid = low + (toFind - sortedArray[low]) * (high - low) /
(sortedArray[high] - sortedArray[low]);
         if (sortedArray[mid] < toFind) {
            comparisonCount++;
            low = mid + 1;
         } else if (sortedArray[mid] > toFind) {
            comparisonCount++;
            high = mid - 1;
         } else {
            System.out.println("Интерполяционный поиск - индификатор найден.
Количество проверок: " + comparisonCount);
            return;
         }
      }

      if (sortedArray[low] == toFind) {
         System.out.println("Интерполяционный поиск - индификатор найден.
Количество проверок: " + comparisonCount);
      }else

      if (sortedArray[high] == toFind) {
```

```
        System.out.println("Интерполяционный поиск - индификатор найден. Количество проверок: " + comparisonCount);
      }else

        System.out.println("Интерполяционный поиск - индификатор не найден. Количество проверок: " + comparisonCount);

   }

}
```

Листинг поиска Фиббоначи

```
class Fibonacci

{

   public static int min(int x, int y)

   { return (x <= y)? x : y; }



   public static void Search(int arr[], int toFind, int ArrayLength) {
      int fibMMm2 = 0;
      int fibMMm1 = 1;
      int comparisonCount =0;
      int fibM = fibMMm2 + fibMMm1;

      while (fibM < ArrayLength) {
         fibMMm2 = fibMMm1;
         fibMMm1 = fibM;
         fibM = fibMMm2 + fibMMm1;
         comparisonCount++;
      }

      int offset = -1;

      while (fibM > 1) {
```

```
            int i = min(offset+fibMMm2, ArrayLength-1);
            if (arr[i] < toFind)
            {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
                comparisonCount++;
            }
            else if (arr[i] > toFind)
        {
            comparisonCount++;
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        } else {
                System.out.println("Фиббоначев поиск - индификатор найден.
Количество проверок: " + comparisonCount);
                return;
            }
        }
        if(fibMMm1 == 1 && arr[offset+1] == toFind)
            System.out.println("Фиббоначев поиск - индификатор найден.
Количество проверок: " + comparisonCount);
        else
            System.out.println("Фиббоначев поиск - индификатор не найден.
Количество проверок: " + comparisonCount);

    }
```

Листинг вызывающей функции

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

public class SearchMain {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Введите количество элементов для поиска: ");
        int n = in.nextInt();
```

```
        int[] array = new int [n];
        for(int i=0; i<n; i++)
            array[i]= (int)(Math.random()*1000)+1;
        Arrays.sort(array);
        for(int i=0; i<n; i++)
            System.out.print(array[i]+" ");
        int num = 1;
        while(num > 0) {
            System.out.print("Напишите число для поиска: ");
            num = in.nextInt();
            BinaryTree bt = new BinaryTree(array);
            bt.find(num);
            BinarySearch.search(array, 0, array.length, num);
            Interpolation.interpolationSearch(array, num);
            Fibonacci.Search(array, num, array.length);
        }
        in.close();
    }
}
```

Задание 2

Листинг простого рехеширования

```
import java.util.Objects;

class Hash {

    private final int MAXarray_size = 32;
    private final int[][] HashTable;
    private int array_size;

    Hash() {
        HashTable = new int[MAXarray_size][2];
        array_size = 0;
    }


    private int indexOfHashTable(int key, int comparisonsCount) {
        return (key + comparisonsCount) % this.MAXarray_size;
```

```
    }

    public void add(int key, int number){
        int comparisonsCount = 0;
        if(HashTable[indexOfHashTable(key,comparisonsCount)][0]==0) {
            HashTable[indexOfHashTable(key, comparisonsCount)][1] = number;
            HashTable[indexOfHashTable(key, comparisonsCount)][0] = key;
        }
        else {
            while (HashTable[indexOfHashTable(key, comparisonsCount)][0]!=0)
                ++comparisonsCount;
            HashTable[indexOfHashTable(key, comparisonsCount)][1] = number;
            HashTable[indexOfHashTable(key, comparisonsCount)][0] = key;
        }
        array_size++;
    }

    public void delete(int key){
        int comparisonCount;
        boolean flag = false;
        for(comparisonCount = 0; !flag && comparisonCount < this.array_size
                && HashTable[indexOfHashTable(key, comparisonCount)][0]!=0;
++comparisonCount) {
            if (this.HashTable[this.indexOfHashTable(key,
comparisonCount)][0]==key) {
                flag = true;
                HashTable[indexOfHashTable(key, comparisonCount)][0] = 0;
                HashTable[indexOfHashTable(key, comparisonCount)][1] = 0;
                array_size--;
            }
        }
        if (!flag)
            System.out.println("Элемент не найден");

    }




    public void search(int key) {
        boolean flag = false;
```

```
        int comparisonCount;
        for(comparisonCount = 0; !flag && comparisonCount < array_size
                && HashTable[indexOfHashTable(key, comparisonCount)][0]!=0;
++comparisonCount) {
            if (HashTable[indexOfHashTable(key, comparisonCount)][0]==key) {
                flag = true;
                System.out.println("Поиск по хеш-таблице, с простым
рехешированием. Индификатор равен: "+HashTable[indexOfHashTable(key,
comparisonCount)][1]+". Количество проверок: " + comparisonCount+1);
            }
        }

        if (!flag) {
            System.out.println("Поиск по хеш-таблице, с простым
рехешированием. Индификатор не найден. Количество проверок: " +
comparisonCount);
        }

    }


}
```

Листинг рехеширования случайными числами

```
public class RandomHash {

    private final int MAXarray_size = 32;
    private int[] additionalArray;
    private final int[][] HashTable;
    private int array_size;

    RandomHash() {
        HashTable = new int[MAXarray_size][2];
        array_size = 0;
        setAdditionalArray();
    }

    private void setAdditionalArray(){
```

```
        additionalArray = new int[MAXarray_size];
        for(int i = 0; i < 31; ++i) {
            this.additionalArray[i] = (int)(Math.random() * 100.0D);
        }
    }

    private int indexOfHashTable(int key, int comparisonsCount) {
        return (key + additionalArray[comparisonsCount]) % this.MAXarray_size;
    }

    public void add(int key, int number){
        int comparisonsCount = 0;
        if(HashTable[indexOfHashTable(key,comparisonsCount)][0]==0) {
            HashTable[indexOfHashTable(key,comparisonsCount)][1]= number;
            HashTable[indexOfHashTable(key,comparisonsCount)][0]= key;
        }
        else {
            for(; HashTable[indexOfHashTable(key, comparisonsCount)][0]!=0;
++comparisonsCount) {
            }
            HashTable[indexOfHashTable(key,comparisonsCount)][1]= number;
            HashTable[indexOfHashTable(key,comparisonsCount)][0]= key;
        }
        array_size++;
    }
    public void delete(int key){
        int comparisonCount;
        boolean flag = false;
        for(comparisonCount = 0; !flag && comparisonCount < this.array_size
                && HashTable[indexOfHashTable(key, comparisonCount)][0]!=0;
++comparisonCount) {
            if (this.HashTable[this.indexOfHashTable(key,
comparisonCount)][0]==key) {
                flag = true;
                HashTable[indexOfHashTable(key, comparisonCount)][0] = 0;
                HashTable[indexOfHashTable(key,comparisonCount)][1]= 0;
                array_size--;
            }
        }
        if (!flag)
            System.out.println("Элемент не найден");
```

```
    }


    public void search(int key) {
        boolean flag = false;

        int comparisonCount;
        for(comparisonCount = 0; !flag && comparisonCount < this.array_size
                && HashTable[indexOfHashTable(key, comparisonCount)][0]!=0;
++comparisonCount) {
            if (HashTable[this.indexOfHashTable(key, comparisonCount)][0]==key) {
                flag = true;
                System.out.println("Поиск по хеш-таблице, с рехешированием
псевдослучайными числами. Индификатор равен:
"+HashTable[indexOfHashTable(key, comparisonCount)][1]+". Количество
проверок: " + comparisonCount+1);
            }
        }

        if (!flag) {
            System.out.println("Поиск по хеш-таблице, с рехешированием
псевдослучайными числами. Индификатор не найден. Количество проверок:
" + comparisonCount);
        }

    }


}
```

Листинг рехеширования методом цепочек

```
import java.util.*;
public class Chains {

    private final int MAXarray_size = 32;
    private final LinkedList[] list = new LinkedList[MAXarray_size];
    private  final int[] HashTable;
```

```
private  int array_size;

Chains(int[] array) {
   HashTable = new int[MAXarray_size];
   array_size = array.length;
   for(int i=0 ; i< MAXarray_size; i++){
      list[i] = new LinkedList<Integer>();
   }
   ResetToZero();

   for(int i = 0; i < array_size; ++i) {
      if(HashTable[indexOfHashTable(array[i])]!=0)
         list[indexOfHashTable(array[i])].add(array[i]);
      else
         HashTable[indexOfHashTable(array[i])] = array[i];
   }

}

public void add(int number){
   if(HashTable[indexOfHashTable(number)]!=0)
      list[indexOfHashTable(number)].add(number);
   else
      HashTable[indexOfHashTable(number)] = number;
   array_size++;
}
public void delete(int number){
   if(HashTable[indexOfHashTable(number)]!=0)
      list[indexOfHashTable(number)].remove((Integer) 12);
   else
      HashTable[indexOfHashTable(number)] = 0;
   array_size--;
}

private int indexOfHashTable(int num) {
   return (num) % this.MAXarray_size;
}

private void ResetToZero() {
   for(int i = 0; i < this.MAXarray_size; ++i) {
      this.HashTable[i] = 0;
```

```
            list[i].clear();
        }

    }



    public boolean listSearch(int num){
        return list[indexOfHashTable(num)].contains(num);
    }
    public int listNumberLocationIndex(int num){
        return list[indexOfHashTable(num)].indexOf(num)+1;
    }

    public void search(int num) {
        boolean flag = false;
        int comparisonCount=1;
        if (HashTable[indexOfHashTable(num)]==0)
            flag = false;
        if (HashTable[indexOfHashTable(num)]!=num) {
            flag = listSearch(num);
            comparisonCount+= listNumberLocationIndex(num);
            }
        if(HashTable[indexOfHashTable(num)]==num)
            flag = true;


        if (flag) {
            System.out.println("Поиск по хеш-таблице. Индификатор найден.
Количество проверок: " + comparisonCount);
        } else {
            System.out.println("Поиск по хеш-таблице. Индификатор не найден.
Количество проверок: " + comparisonCount);
        }
    }

    public  void see(){
        for(int i=0; i<32; i++){
            System.out.println(HashTable[i]);
            for(int j=0 ; j<list[i].size(); j++)
```

```
            System.out.print("->"+list[i].get(j)+" ");
        }
    }
```

Листинг вызывающей функции

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

public class MainHash {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Введите количество элементов для поиска: ");
        int n = in.nextInt();
        int[] array = new int [n];
        for(int i=0; i<n; i++)
            array[i]= (int)(Math.random()*1000)+1;
        for(int i=0; i<n; i++)
            System.out.print(array[i]+" ");
        RandomHash rHash = new RandomHash(array);
        Hash hash = new Hash(array);
        Chains chains = new Chains(array);
        int num = 1;
        while(num > 0) {
            System.out.print("Напишите число для поиска: ");
            num = in.nextInt();
            rHash.search(num);
            hash.search(num);
            chains.search(num);
        }
    }
}
```

Задание 3

```
public class Chess {
    int[][] borad = {
            {0,0,0,0,0,0,0,0},
```

```
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},};

int[][] openBorad = {
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},
        {0,0,0,0,0,0,0,0},};

public void place(int x, int y){
    borad[y][x] = 1;
    close(x,y,1);
    placeNext(y+1);
}

public void replace(int x, int y){
    borad[y][x] = 0;
    close(x,y,-1);
}

public void close(int x, int y, int p){
    int i=0;
    while (x+i<8 && y+i<8) {
        openBorad[y + i][x + i] += p;
        i++;
    }
    i=1;
    while (x-i>=0 && y+i<8) {
        openBorad[y + i][x - i] += p;
        i++;
    }
    i=1;
```

```
        while (y+i<8){
            openBorad[y+i][x] += p;
            i++;
        }
    }

    public void placeNext(int y){
        if(y!=8) {
            for (int i = 0; i < 8; i++) {
                if (openBorad[y][i] <= 0) {
                    place(i, y);
                    replace(i, y);
                }
            }
        }
        else {
            showBoard();
            System.out.println("\n\n");
        }
    }

    public void showBoard(){
        for(int i=0; i<8; i++){
            for(int j=0; j<8; j++)
                System.out.print(borad[i][j]+"\t");
            System.out.println("\n");
        }
    }

    public static void main(String[] args) {
        Chess chess = new Chess();
        chess.placeNext(0);
    }
}
```

3 Результат работы программы

Результат работы функции поиска элементов в массиве изображен на рисунке 1.

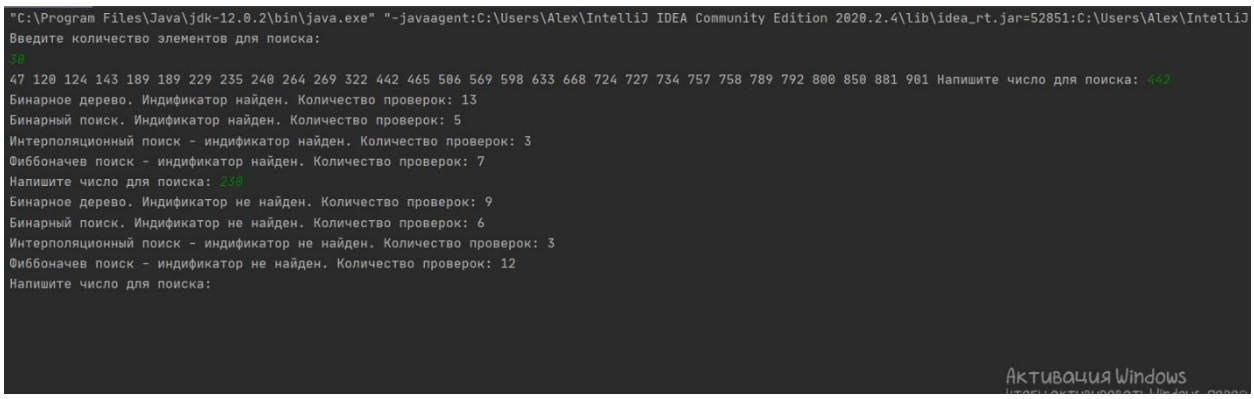

Рисунок 1 – результат работы функции поиска

Результат работы функции рехеширования изображен на рисунке 2.

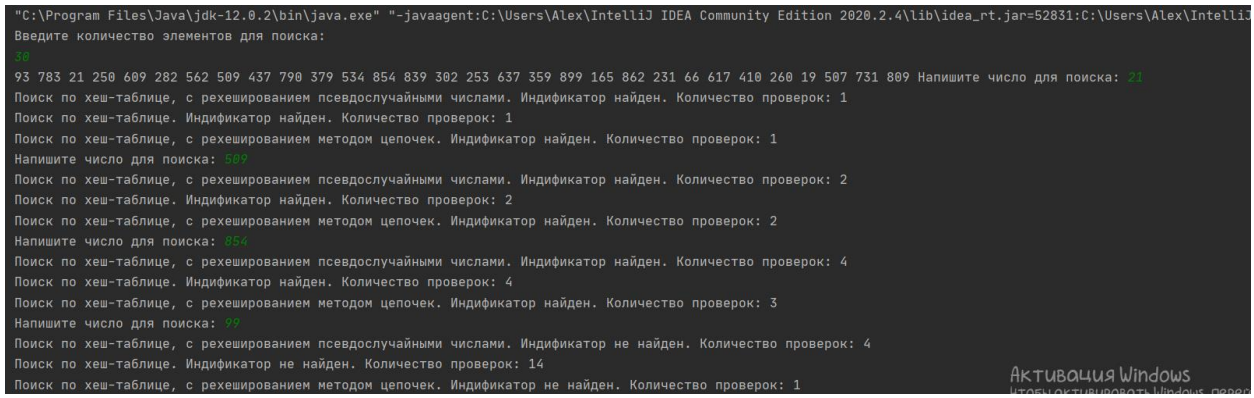

Рисунок 2 – результат работы функции рехеширования

Результат работы функции решения 3-го задания изображен на рисунке 3.

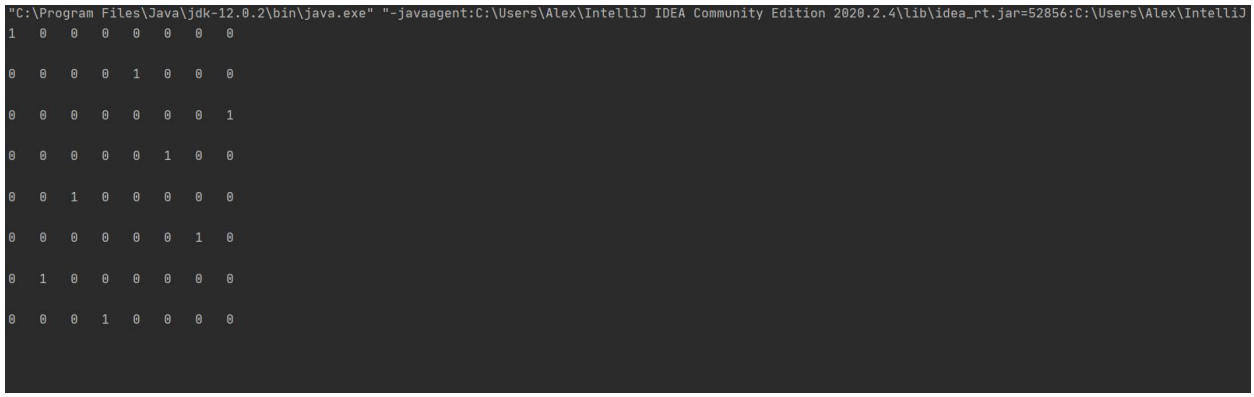

Рисунок 3 – результат работы функции поиска решения 3-го задания

Вывод: В результате лабораторной работы, были сделаны разные функции поиска и рехеширования.