Федеральное агентство связи
Ордена трудового красного знамени
Федеральное государственное образовательное
Бюджетное
Учреждение высшего профессионального образования
Московский Технический Университет связи и информатики

Кафедра МКиТ

Лабораторная работа № 2

по дисциплине «Структуры и алгоритмы обработки данных»

На тему: «Методы поиска»

Выполнил: студент
Группы БСТ1902
Птушкин А.Н.
Вариант №16
Проверил: ст. преп.
Мелехин А.

Москва 2021г

Оглавление

1 Цель работы

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.

Методы поиска:

1. Бинарный поиск
2. Бинарное дерево
3. Фибоначчиев
4. Интерполяционный

Методы рехеширования:

1. Простое рехеширование
2. Рехеширование с помощью псевдослучайных чисел
3. Метод цепочек

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 работы

2.1 Методы поиска

В листинге 1 представлен программный код на языке Java 15, отвечающий за методы поиска.

Листинг 1

```
import java.util.ArrayList;
import java.util.List;

/** Отвечает за поиск элементов **/
public class Search {

    /** Методы поиска **/
    public static final int SIMPLE_SEARCH=0;
    public static final int BINARY_SEARCH=1;
    public static final int TREE_SEARCH=2;
    public static final int FIBONACCI_SEARCH=3;
    public static final int INTERPOLATION_SEARCH=4;


    /** Ссылка на объект класса управления **/
    private static Interactions interactions;

    /** Установка ссылка на объект класса управления **/
    public static void setInteraction(Interactions obj){
        interactions=obj;
    }

    /** Главный метод поиска, запускает остальные **/
    public static int find (int value, int method){

        List<Integer> list = interactions.getHashList();

        int low=0;
        int high=list.size()-1;

        int index=0;

        switch (method){
            case SIMPLE_SEARCH:
                index=simpleSearch(value,list);
                break;
            case BINARY_SEARCH:
                index=binarySearch(value,low,high,list);
                break;
            case INTERPOLATION_SEARCH:
                index=interpolationSearch(value,list);
                break;
            case FIBONACCI_SEARCH:
                index=fibonacciSearch(value,list);
                break;
            case TREE_SEARCH:
                index=treeSearch(value,list);
                break;
        }
        return index;
    }
```

```
    /** Метод поиска бинарным деревом, инициализирует его, заполняет и
вызывает метод поиска  **/
    public static int treeSearch(int value,List<Integer> list){
        BST_class tree = new BST_class();
        for(int i=0;i<list.size();i++){
            tree.insert(list.get(i));
        }
        boolean found=tree.search(value);
        if(found)
            return-100;
        else
            return -1;
    }

    /** Простой перебор элементов **/
    private static int simpleSearch(int value, List<Integer> list){
        int index=-1;

        for (int i=0;i<list.size();i++){
            if(list.get(i)==value){
                index=i;
                break;
            }
        }
        return index;
    }

    /** Бинарный поиск **/
    private static int binarySearch(int value, int low, int high,
List<Integer> list){
        int index = -1;

        while (low <= high) {
            int mid = (low + high) / 2;
            if (list.get(mid) < value) {
                low = mid + 1;
            } else if (list.get(mid) > value) {
                high = mid - 1;
            } else if (list.get(mid) == value) {
                index = mid;
                break;
            }
        }
        return index;
    }


    /** Интерполяционный поиск **/
    private static int interpolationSearch(int value, List<Integer> list){
        int startIndex = 0;
        int lastIndex = (list.size() - 1);


        while ((startIndex <= lastIndex) && (value >= list.get(startIndex))
&& (value <= list.get(lastIndex))) {
            // используем формулу интерполяции для поиска возможной лучшей
позиции для существующего элемента
            int pos = startIndex + (((lastIndex-startIndex) /
(list.get(lastIndex)-list.get(startIndex)))* (value -
list.get(startIndex)));

            if (list.get(pos) == value)
```

```
                return pos;

            if (list.get(pos) < value)
                startIndex = pos + 1;

            else
                lastIndex = pos - 1;
        }
        return -1;
    }

    /** Фибоначиев поиск **/
    public static int fibonacciSearch(int value, List<Integer> list)
    {
        int listSize=list.size();
        /* Initialize fibonacci numbers */
        int fibMMm2 = 0; // (m-2)'th Fibonacci No.
        int fibMMm1 = 1; // (m-1)'th Fibonacci No.
        int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

        /* fibM is going to store the smallest
        Fibonacci Number greater than or equal to listSize */
        while (fibM < listSize) {
            fibMMm2 = fibMMm1;
            fibMMm1 = fibM;
            fibM = fibMMm2 + fibMMm1;
        }

        // Marks the eliminated range from front
        int offset = -1;

        /* while there are elements to be inspected.
        Note that we compare list[fibMm2] with value.
        When fibM becomes 1, fibMm2 becomes 0 */
        while (fibM > 1) {
            // Check if fibMm2 is a valid location
            int i = Math.min(offset + fibMMm2, listSize - 1);

            /* If value is greater than the value at
            index fibMm2, cut the subarray array
            from offset to i */
            if (list.get(i) < value) {
                fibM = fibMMm1;
                fibMMm1 = fibMMm2;
                fibMMm2 = fibM - fibMMm1;
                offset = i;
            }

            /* If value is less than the value at index
            fibMm2, cut the subarray after i+1 */
            else if (list.get(i) > value) {
                fibM = fibMMm2;
                fibMMm1 = fibMMm1 - fibMMm2;
                fibMMm2 = fibM - fibMMm1;
            }

            /* element found. return index */
            else
                return i;
        }

        /* comparing the last element with value */
        if (fibMMm1 == 1 && list.get(listSize-1) == value)
            return listSize-1;
```

```
        /*element not found. return -1 */
        return -1;
    }
}
```

2.2 Методы рехеширования

В листинге 2 представлен программный код на языке Java 15, отвечающий за организацию методов рехеширования.

Листинг 2

```
import java.util.*;

/** Отвечает за все методы, связанные с хешом **/
public class Hash {

    public static final int REHASH_SIMPLE=0;
    public static final int REHASH_RANDOM=1;
    public static final int REHASH_CHAINS=2;

    private int rehashMethod;
    private int maxHashSize;

    private Random random= new Random();

    private HashMap<Integer,String> hashMap= new HashMap<>();
    private HashMap<Integer,LinkedList<String>> chainsMap = new HashMap<>();

    public Hash(int method, int size){
        rehashMethod=method;
        maxHashSize=size;
        for(int i=0;i<maxHashSize;i++){
            chainsMap.put(i,new LinkedList<String>());
        }
    }

    public HashMap<Integer,String> getTable(){
        if (rehashMethod!=REHASH_CHAINS) {
            return hashMap;
        }
        else{
            printChainsMap();
            return hashMap;
        }
    }

    public void printChainsMap(){
        int counter=1;

        for(int i: chainsMap.keySet()){
            if(!chainsMap.get(i).isEmpty()) {
                System.out.println(counter + ".\t" + i + "\t" +
chainsMap.get(i));
            }
        }
```

```
    }

    public int getHash(String value){
        int hash=1;
        for(int i=0; i<value.length(); i++){
            hash*=(int)value.charAt(i)*31;
        }
        return Math.abs(hash%maxHashSize);
    }


    public void add(String value){
        int hash=getHash(value);
        if(!hashMap.containsKey(hash)){
            if (rehashMethod==REHASH_CHAINS){
                chainsMap.get(hash).add(value);
            }
            else {
                hashMap.put(hash, value);
            }
        }
        else{
            switch (rehashMethod){
                case REHASH_SIMPLE:
                    rehashSimple(hash,value);
                    break;
                case REHASH_RANDOM:
                    rehashRandom(hash,value);
                    break;
                case REHASH_CHAINS:
                    rehashChains(hash,value);
                    break;
            }

        }
    }


    private void rehashSimple(int hash, String value){
        while(hashMap.containsKey(hash)){
            hash++;
        }
        hash%=maxHashSize;
        hashMap.put(hash,value);
    }
    private void rehashRandom(int hash, String value){
        while(hashMap.containsKey(hash)){
            hash+=random.nextInt();
        }
        hash%=maxHashSize;
        hashMap.put(hash,value);
    }
    private void rehashChains(int hash, String value){
        chainsMap.get(hash).add(value);
    }


    public int find(String value){
        if (rehashMethod!=REHASH_CHAINS) {
            for (int i : hashMap.keySet()) {
                if(hashMap.get(i).equals(value)){
                    return i;
                }
            }
```

```
        }
        else{
            for(int i: chainsMap.keySet()){
                if(chainsMap.get(i).contains(value)){
                    return i;
                }
            }
        }
        return -1;
    }

    public void delete(String value){
        int hash=find(value);
        if(hash>=0){
            if(rehashMethod!=REHASH_CHAINS){
                hashMap.remove(hash);
            }
            else{
                chainsMap.get(hash).remove(value);
            }
        }

    }

}
```

2.3 Расстановка фигур на шахматной доске

В листинге 3 представлен программный код на языке Java 15, отвечающий за создание шахматной доски и поиск возможной комбинации расстановки n фигур на поле nxn.

Листинг 3

```
import java.util.ArrayList;
import java.util.Arrays;

/** Класс, расставляющий на шахматном поле размером NxN N ферзей так, чтобы
они не находились под боем друг друга **/
public class Chess {

    /** Возможные статусы ячеек **/
    private final int CELL_OPENED=0;
    private final int CELL_CLOSED=1;
    private final int CELL_WITH_FIGURE=2;


    /** Максимальное количество фигур **/
    private final int maxFigureCount;
    /** Текущее количество фиугр **/
    private int figureCount;
    /** Количество ячеек поля **/
    private final int fieldSize;
    /** Длина стороны поля **/
    private final int fieldLength;
```

```
    /** Массив ячеек поля **/
    private final  int[] cells;
    /** Коллекция неподходящих наборов **/
    private final  ArrayList<Integer[]> badCombination=new ArrayList<>();
    /** Набор текущих фиугр на поле **/
    private final ArrayList<Integer> figuresOnBoard=new ArrayList<>();


    /** Конструктор класса **/
    public Chess(int fieldLength, int maxFigureCount){
        this.fieldLength=fieldLength;
        fieldSize=fieldLength*fieldLength;
        this.maxFigureCount=maxFigureCount;

        figureCount=0;
        cells=new int[fieldSize];
        for(int i=0;i<fieldSize;i++){
            cells[i]=CELL_OPENED;
        }
    }

    /** Метод, закрывающий ячейки по диагонали слева направо **/
    private void closeLeftDiagonal(int startCell, int status){

        int currentCell=startCell;
        int nextCell = currentCell-fieldLength+1;

        int currentCol = currentCell%fieldLength;
        int nextCol = nextCell%fieldLength;

        //Going up
        while (currentCell>0 && nextCol==currentCol+1){
            currentCell=nextCell;
            nextCell=currentCell-fieldLength+1;

            currentCol=nextCol;
            nextCol=nextCell%fieldLength;
        }

        nextCell=currentCell+fieldLength-1;

        currentCol=currentCell%fieldLength;
        nextCol= nextCell%fieldLength;

        //Going down
        while(currentCell<fieldSize && nextCol==currentCol-1){
            if(cells[currentCell]!=CELL_WITH_FIGURE){
                cells[currentCell]=status;
            }
            currentCell=nextCell;
            nextCell=currentCell+fieldLength-1;

            currentCol=nextCol;
            nextCol=nextCell%fieldLength;
        }

        //last cell check
        if (currentCell<fieldSize && currentCell%fieldLength==0 &&
cells[currentCell]!=CELL_WITH_FIGURE){
            cells[currentCell]=status;
        }
    }

    /** Метод, закрывающий ячейки по диагонали справо налево **/
```

```
    private void closeRightDiagonal(int startCell, int status){
        int currentCell=startCell;
        int nextCell = currentCell-fieldLength-1;

        int currentCol = currentCell%fieldLength;
        int nextCol = currentCol-1;

        //Going up
        while (currentCell>0 && nextCol==currentCol-1){
            currentCell=nextCell;
            nextCell=currentCell-fieldLength-1;

            currentCol=nextCol;
            nextCol=nextCell%fieldLength;
        }
        if(currentCell<0){
            currentCell+=fieldLength+1;
        }

        nextCell=currentCell+fieldLength+1;

        currentCol=currentCell%fieldLength;
        nextCol= nextCell%fieldLength;

        //Going down
        while(currentCell<fieldSize && nextCol==currentCol+1){
            if(cells[currentCell]!=CELL_WITH_FIGURE){
                cells[currentCell]=status;
            }
            currentCell=nextCell;
            nextCell=currentCell+fieldLength+1;

            currentCol=nextCol;
            nextCol=nextCell%fieldLength;
        }

        //Last cell check
        if (currentCell<fieldSize && currentCell%fieldLength==7 &&
cells[currentCell]!=CELL_WITH_FIGURE){
            cells[currentCell]=status;
        }
    }

    /** Метод, закрывающий ячейки по вертикали **/
    private void closeVertical(int startCell, int status){

        int currentCell=startCell;

        //Поднимаемся выше по вертикали
        while(currentCell>0){
            currentCell-=fieldLength;
        }

        if(currentCell!=0) {
            currentCell += fieldLength;
        }

        //Спускаемся и закрываем все открытые ячейки
        while(currentCell<fieldSize){
            if(cells[currentCell]!=CELL_WITH_FIGURE){
                cells[currentCell]=status;

            }
            currentCell+=fieldLength;
```

```
        }
    }

    /** Метод, закрывающий ячейки по горизонатли **/
    private void closeHorizontal(int startCell, int status){
        int currentCell=startCell;

        //Определить строку
        int row=startCell/fieldLength;

        //Сдвигаемся влево внутри строки
        while(currentCell/fieldLength==row && currentCell>=0){
            currentCell--;
        }
        currentCell++;

        while(currentCell/fieldLength==row && currentCell<fieldSize){
            if(cells[currentCell]!=CELL_WITH_FIGURE){
                cells[currentCell]=status;
            }
            currentCell++;
        }
    }


    /** Открывает ячейки во все стороны от текущей **/
    public void openCells(int i){
        closeHorizontal(i,CELL_OPENED);
        closeVertical(i,CELL_OPENED);
        closeLeftDiagonal(i,CELL_OPENED);
        closeRightDiagonal(i,CELL_OPENED);
    }

    /** Закрывает ячейки во все стороны от текущей **/
    public void closeCells(int i){
        closeHorizontal(i,CELL_CLOSED);
        closeVertical(i,CELL_CLOSED);
        closeLeftDiagonal(i,CELL_CLOSED);
        closeRightDiagonal(i,CELL_CLOSED);
    }

    /** Проверяет на наличие комбинации в коллекции неподходящих комбинаций
**/
    private boolean canTry(int index){

        int size=figuresOnBoard.size()+1;
        Integer[] combination = new Integer[size];

        if(figuresOnBoard.size()>0) {
            for (int i = 0; i < size - 1; i++) {
                combination[i] = figuresOnBoard.get(i);
            }
        }
        combination[size-1]=index;

        for (Integer[] integers : badCombination) {
            if (Arrays.equals(integers, combination)) {
                return false;
            }
        }
        return true;
    }

    /** Возвращается на один шаг назад и запоминает текущую комбинацию **/
```

```
    private boolean getBack(){
        //Добавление плохой комбинации
        int size=figuresOnBoard.size();
        Integer[] combination = new Integer[size];

        for(int i=0;i<size;i++){
            combination[i]=figuresOnBoard.get(i);
        }

        badCombination.add(combination);
        if (size==1 && combination[0]==maxFigureCount-1) {
            return true;
        }
        else{
            for (int i: figuresOnBoard){
                openCells(i);
            }

            cells[figuresOnBoard.get(figuresOnBoard.size() - 1)] =
CELL_OPENED;
            figuresOnBoard.remove(figuresOnBoard.get(figuresOnBoard.size() -
1));
            figureCount--;
            for (int i: figuresOnBoard){
                closeCells(i);
            }
            return false;
        }
    }

    /** Основной метод, расставляющий фигуры **/
    private boolean putFigures(){
        int i;
        boolean noCellsLeft=false;

        while(figureCount<maxFigureCount && !noCellsLeft){

            i = getAvailableCell();
            System.out.println(figuresOnBoard);


            if (i>=0) {
                putNextFigure(i);
            } else{
                noCellsLeft=getBack();
            }
        }
        return noCellsLeft;
    }

    /** Ставит одну фигуру на поле **/
    private void putNextFigure(int index){
        cells[index]=CELL_WITH_FIGURE;
        figureCount++;
        figuresOnBoard.add(index);
        closeCells(index);
    }

    /** Получить первую доступную ячейку **/
    private int getAvailableCell(){
        int size =figuresOnBoard.size()-1;
        int index;
        if (size<0){
            index=0;
```

```
        }
        else{
            index=figuresOnBoard.get(size);
        }


        while (index<fieldSize){
            if(cells[index]==CELL_OPENED && canTry(index)){
                return index;
            }
        index++;
        }
        return -1;
    }

    /** Выводит на экран шахматное поле **/
    private void printChessTable(){
        for(int i=0;i<fieldSize;i++){
            if(i%fieldLength==0){
                System.out.print("\n");
            }

            switch (cells[i]) {
                case CELL_OPENED -> System.out.print("- ");
                case CELL_CLOSED -> System.out.print("x ");
                case CELL_WITH_FIGURE -> System.out.print("O ");
            }
        }
        System.out.print("\n");
    }

    /** Запускает перебор **/
    public void startApp(){
        boolean notDone=putFigures();
        if (!notDone) {
            System.out.println("\nШахматное поле (0-Фигура, x-ячейка под
боем):");
            printChessTable();
            System.out.println("\nКоординаты фигур:");
            System.out.println(figuresOnBoard + "\n");
        }
        else{
            System.out.println("\nНевозможно поставить "+maxFigureCount+"
фигур на поле "+fieldLength+"x"+fieldLength);
        }
    }
}
```

3 Результат работы

3.1 Результаты поиска

На рисунках 1-5 представлены скриншоты работы реализованных методов поиска.

```
Введите комнаду:
Print
1.  256 xtHMaIm
2.  752 luWfwfO
3.  352 HIl
4.  336 HZ
5.  192 lQrBTC
6.  448 cahBD
7.  2   GSjKEO
8.  146 fS
9.  437 gC
10. 367 YQw
Команда выполнена

Введите комнаду:
Find
Выберите способ поиска:
1.Простой
2.Бинарный
3.Интерполяционный
4.Бинарное дерево
5.Фибоначиев
1
Введите значение:
fS
Значение найдено в наборе с под номером 8
Команда выполнена
```

Рисунок 1 – Простой поиск

```
Введите комнаду:
Print
1.  256 xtHMaIm
2.  752 luWfwfO
3.  352 HIl
4.  336 HZ
5.  192 lQrBTC
6.  448 cahBD
7.  2   GSjKEO
8.  146 fS
9.  437 gC
10. 367 YQw
Команда выполнена

Введите комнаду:
Find
Выберите способ поиска:
1.Простой
2.Бинарный
3.Интерполяционный
4.Бинарное дерево
5.Фибоначиев
2
Введите значение:
GSjKEO
Значение найдено в наборе с под номером 7
Команда выполнена
```

Рисунок 2 – Бинарный поиск

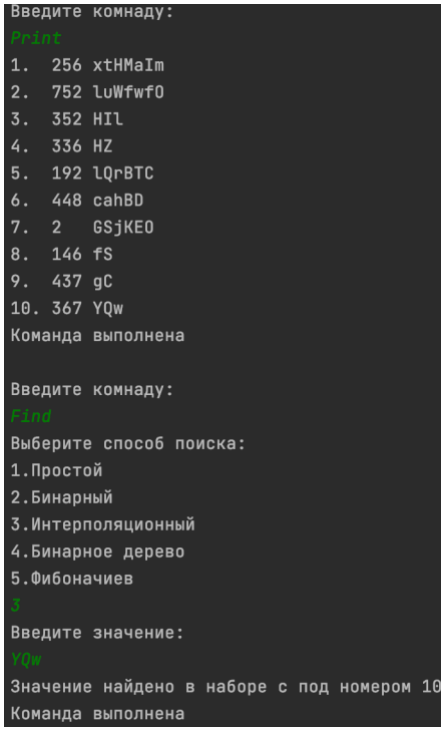

Рисунок 3 – Интерполяционный поиск

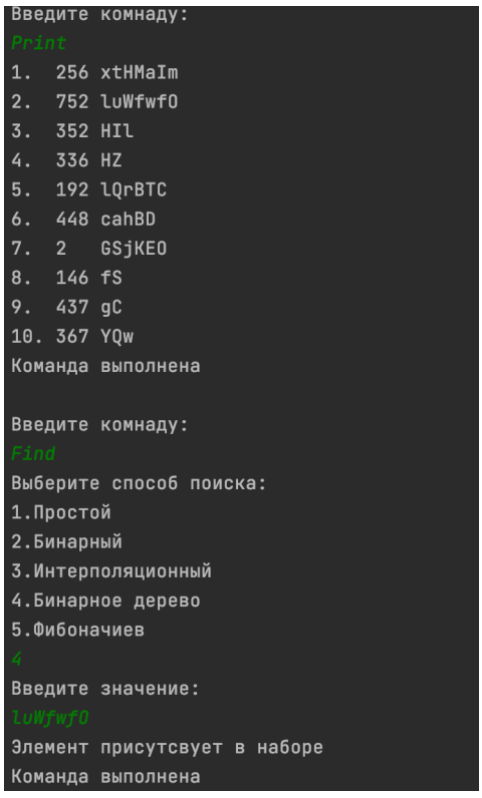

Рисунок 4 – Поиск в бинарном дереве

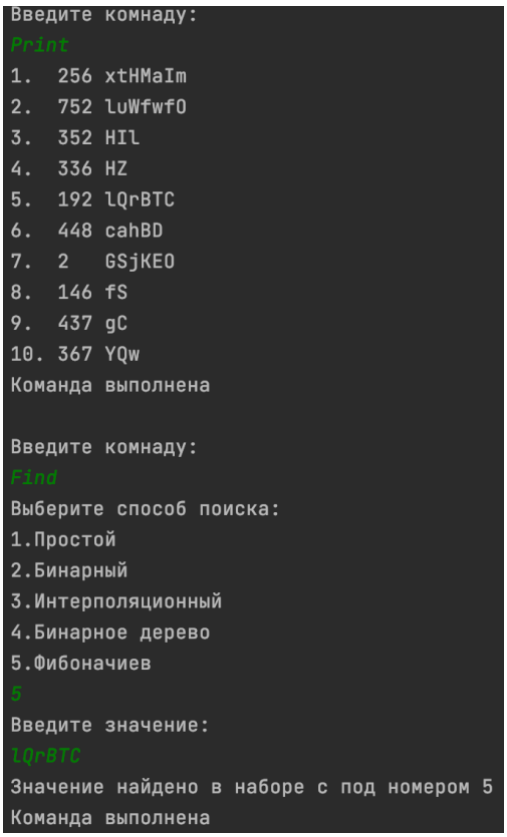

Рисунок 5 – Поиск с помощью метода Фибоначчи

3.2 Результаты рехеширования

На скриншотах 6-8 представлены скриншоты работы реализованных методов рехеширования. (таблица на скриншотах имеет следующую структуру: номер по порядку – хеш – значение)

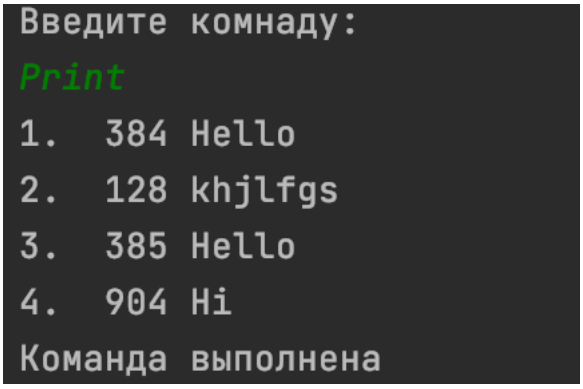

Рисунок 6 – Таблица, построенная с использованием простого рехеширования

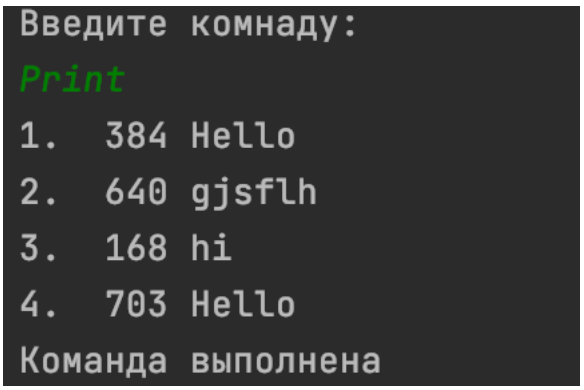

Рисунок 7 – Таблица, построенная с использованием случайного рехеширования

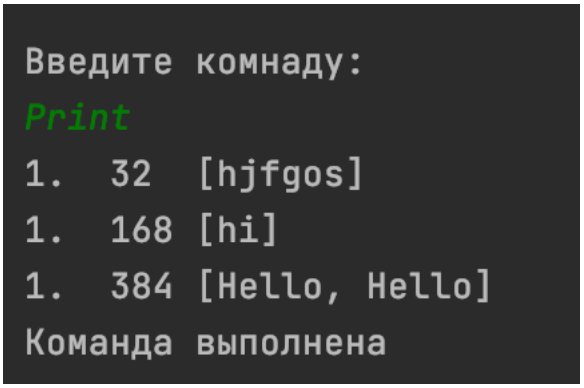

Рисунок 8 – Таблица, построенная с использованием метода цепочек

3.3 Результаты работы расстановки фигур

На рисунке 9 представлен скриншот работы алгоритма, который расставляет n фигур на доске nxn (Для примера было выбрано n=8).

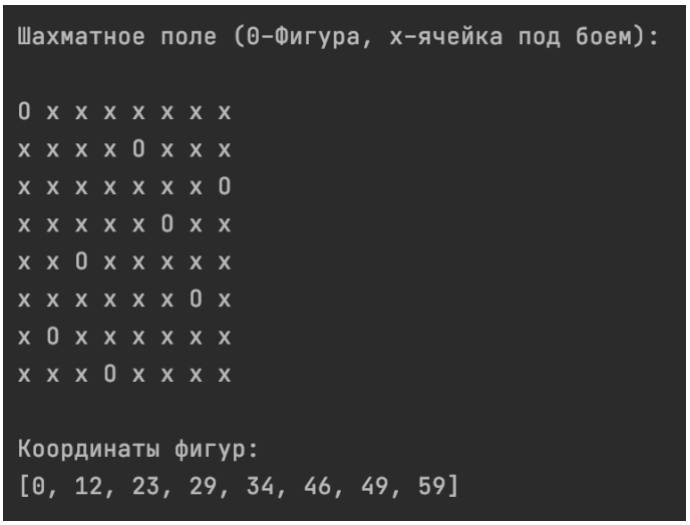

Рисунок 9 – Результат работы алгоритма

3.4 Вывод

В ходе работы реализовали методы поиска в соответствии с заданием. Организовали генерацию начального набора случайных данных. Для всех вариантов добавили реализацию добавления, поиска и удаления элементов.

Написали программу, которая находит хотя бы один способ расстановки 8 ферзей на стандартной 64-клеточной шахматной доске, так чтобы ни один из них не находился под боем другого.