Министерство цифрового развития, связи и массовых коммуникаций

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»



Отчет по лабораторной работе №2

по дисциплине «Структура и алгоритмы обработки данных»

по теме «Методы поиска»

Выполнила: студент группы

БВТ1902

Соколова А.Ю.

Москва

2021 г

Оглавление

[Цель работы 3](#_Toc72676475)

[Задание 1 3](#_Toc72676476)

[Код программы 3](#_Toc72676477)

[Задание 2 3](#_Toc72676478)

[Код программы 3](#_Toc72676479)

[Задание 3 3](#_Toc72676480)

[Код программы 3](#_Toc72676481)

[Снимки экрана работы программ 3](#_Toc72676482)

[Вывод 3](#_Toc72676483)

## Цель работы

Реализовать методы поиска, простое рехеширование и метод цепочек в соответствии с заданием. Организовать генерацию начального набора случайных данных и оценить время работы каждого алгоритма поиска.

## Задание 1

|  |  |  |  |
| --- | --- | --- | --- |
| Бинарный поиск | Бинарное дерево | Фибоначчиев | Интерполяционный |

## Код программы

public class Search {

public static void main(String[] args) {

Scanner scan=new Scanner(System.in);

System.out.println("Введите номер поиска(1-4): ");

int k=scan.nextInt();

if (k==1){

System.out.println("Введите количество элементов массива: ");

int n=scan.nextInt();

int[] a=new int[n];

for (int i=0;i<n;i++){

a[i]=(int) (Math.random()\*100);

}

Arrays.sort(a);

for (int i=0;i<n;i++){

System.out.print(a[i]+" ");;

}

int index=binarySearch(a,scan,n);

System.out.println(index);

}

if (k==4){

System.out.println("Введите количество элементов массива: ");

int n=scan.nextInt();

int[] a=new int[n];

for (int i=0;i<n;i++){

a[i]=(int) (Math.random()\*100);

}

Arrays.sort(a);

for (int i=0;i<n;i++){

System.out.print(a[i]+" ");;

}

int index=interpolationSearch(a,scan,n);

System.out.println(index);

}

if (k==3){

System.out.println("Введите количество элементов массива: ");

int n=scan.nextInt();

int[] a=new int[n];

for (int i=0;i<n;i++){

a[i]=(int) (Math.random()\*100);

}

Arrays.sort(a);

for (int i=0;i<n;i++){

System.out.print(a[i]+" ");;

}

int index=FibonacciSearch(a,scan,n);

System.out.println(index);

}

if (k==2){

Tree tree = new Tree();

// вставляем узлы в дерево:

tree.insertNode(6);

tree.insertNode(8);

tree.insertNode(5);

tree.insertNode(8);

tree.insertNode(2);

tree.insertNode(9);

tree.insertNode(7);

tree.insertNode(4);

tree.insertNode(10);

tree.insertNode(3);

tree.insertNode(1);

// отображение дерева:

tree.printTree();

// удаляем один узел и выводим оставшееся дерево в консоли

tree.deleteNode(5);

tree.printTree();

// находим узел по значению и выводим его в консоли

Node foundNode = tree.findNodeByValue(8);

foundNode.printNode();

}

}

public static int binarySearch(int[] a,Scanner scan,int n){

System.out.println("Введите элемент для бинарного поиска: ");

int x=scan.nextInt();

int first=0;

int last=n-1;

while(first<=last){

int middle=Math.round((first+last)/2);

if (a[middle]==x){

return middle;

} else if (a[middle]<x){

first=middle+1;

} else if (a[middle]>x) {

last = middle - 1;

}

}

return -1;

}

public static int FibonacciSearch(int[] a,Scanner scan,int n){

System.out.println("Введите элемент для поиска Фибоначчи: ");

int x=scan.nextInt();

//Инициализировать числа Фибоначчи

int fibM2 = 0;

int fibM1 = 1;

int fibM = fibM2 + fibM1; // fibM собирается хранить самое маленькое Число Фибоначчи, большее или равное n

while (fibM < n)

{

fibM2 = fibM1;

fibM1 = fibM;

fibM = fibM2 + fibM1;

}

// Отмечает удаленный диапазон спереди

int offset = -1;

// пока есть элементы для проверки. Обратите внимание, что мы сравниваем arr [fibM2] с x. Когда fibM становится 1, fibM2 становится 0

while (fibM > 1)

{

// Проверяем, является ли fibM2 действительным местоположением

int i = Math.min(offset+fibM2, n-1);

// Если х больше значения в индекс fibM2, вырезать массив подмассива от смещения до i

if (a[i] < x)

{

fibM = fibM1;

fibM1 = fibM2;

fibM2 = fibM - fibM1;

offset = i;

}

// Если х больше, чем значение в индексе fibM2, вырезать подрешетку после i + 1

else if (a[i] > x)

{

fibM = fibM2;

fibM1 = fibM1 - fibM2;

fibM2 = fibM - fibM1;

}

else { return i; }

}

// сравнение последнего элемента с x

if(fibM1 == 1 && a[offset+1] == x) {

return offset + 1;

}

return -1;

}

public static int interpolationSearch(int[] a,Scanner scan,int n){

System.out.println("Введите элемент для Интерполяционного поиска: ");

int x=scan.nextInt();

int first=0;

int last=n-1;

while(first<=last&&x>a[first]&&x<a[last]){

// используем формулу интерполяции для поиска возможной лучшей позиции для существующего элемента

int pos= first+(((last-first)/(a[last]-a[first]))\*(x-a[first]));

if (a[pos]==x){

return pos;

} else if (a[pos]<x){

first=pos+1;

} else if (a[pos]>x) {

last = pos - 1;

}

}

if (a[first] == x)

return first;

if (a[last] == x)

return last;

return -1;

}

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 +

'}';

}

}

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 то вставить справа

parentNode.setRightChild(newNode);

return;

}

}

}

}

}

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; // yзел 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()); // соохраняем 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

}

}

## Задание 2

|  |  |
| --- | --- |
| Простое рехеширование | Метод цепочек |

## Код программы

class HashNode<K, V> {

K key;

V value;

HashNode<K, V> next;

public HashNode(K key, V value)

{

this.key = key;

this.value = value;

}

}

class Map<K, V>

{

private ArrayList<HashNode<K, V> > bucketArray;

private int numBuckets;

private int size;

public Map()

{

bucketArray = new ArrayList<>();

numBuckets = 10;

size = 0;

for (int i = 0; i < numBuckets; i++)

bucketArray.add(null);

}

public int size() { return size; }

private int getBucketIndex(K key)

{

int hashCode = key.hashCode();

int index = hashCode % numBuckets;

index = index < 0 ? index \* -1 : index;

return index;

}

public V remove(K key)

{

int bucketIndex = getBucketIndex(key);

HashNode<K, V> head = bucketArray.get(bucketIndex);

HashNode<K, V> prev = null;

while (head != null)

{

if (head.key.equals(key))

break;

prev = head;

head = head.next;

}

if (head == null)

return null;

size--;

if (prev != null)

prev.next = head.next;

else

bucketArray.set(bucketIndex, head.next);

return head.value;

}

// Adds a key value pair to hash

public void add(K key, V value)

{

// Find head of chain for given key

int bucketIndex = getBucketIndex(key);

HashNode<K, V> head = bucketArray.get(bucketIndex);

// Check if key is already present

while (head != null) {

if (head.key.equals(key)) {

head.value = value;

return;

}

head = head.next;

}

// Insert key in chain

size++;

head = bucketArray.get(bucketIndex);

HashNode<K, V> newNode = new HashNode<K, V>(key, value);

newNode.next = head;

bucketArray.set(bucketIndex, newNode);

// If load factor goes beyond threshold, then

// double hash table size

if ((1.0 \* size) / numBuckets >= 0.7)

{

ArrayList<HashNode<K, V> > temp = bucketArray;

bucketArray = new ArrayList<>();

numBuckets = 2 \* numBuckets;

size = 0;

for (int i = 0; i < numBuckets; i++)

bucketArray.add(null);

for (HashNode<K, V> headNode : temp) {

while (headNode != null) {

add(headNode.key, headNode.value);

headNode = headNode.next;

}

}

}

}

// Driver method to test Map class

public static void main(String[] args)

{

Map<String, Integer> map = new Map<>();

map.add("this", 1);

map.add("coder", 2);

map.add("this", 4);

map.add("hi", 5);

System.out.println(map.size());

System.out.println(map.remove("this"));

System.out.println(map.remove("this"));

System.out.println(map.size());

}

}

class ReHashMap<K, V>

{

class MapNode<K, V>

{

K key;

V value;

MapNode<K, V> next;

public MapNode(K key, V value)

{

this.key = key;

this.value = value;

next = null;

}

}

ArrayList<MapNode<K, V>> buckets;

// No. of pairs stored - n

int size;

// Size of the bucketArray - b

int numBuckets;

// Default loadFactor

final double DEFAULT\_LOAD\_FACTOR = 0.75;

public ReHashMap()

{

numBuckets = 5;

buckets = new ArrayList<>(numBuckets);

for (int i = 0; i < numBuckets; i++)

{

// Initialising to null

buckets.add(null);

}

System.out.println("Size of Map: " + numBuckets);

System.out.println("Default Load Factor : " + DEFAULT\_LOAD\_FACTOR + "\n");

}

private int getBucketId(K key)

{

int hashCode = key.hashCode();

return (hashCode % numBuckets);

}

public void insert(K key, V value)

{

int bucketId = getBucketId(key);

MapNode<K, V> head;

MapNode<K, V> newElementNode = new MapNode<K, V>(key, value);

head = buckets.get(bucketId);

newElementNode.next = head;

buckets.set(bucketId, newElementNode);

System.out.println("Pair(" + key + ", " + value + ") inserted");

size++;

double loadFactor = size\*1.0 / numBuckets;

System.out.println("Load factor = " + loadFactor);

//check for rehash

if (loadFactor > DEFAULT\_LOAD\_FACTOR)

{

System.out.println("Load factor rather then 0.75, rehashing");

rehash();

System.out.println("New Size of Map: " + numBuckets + "\n");

}

else

System.out.println("Size of Map: " + numBuckets + "\n");

}

private void rehash()

{

ArrayList<MapNode<K, V>> temp = buckets;

buckets = new ArrayList<MapNode<K, V>>(2 \* numBuckets);

for (int i = 0; i < 2 \* numBuckets; i++)

{

// Initialised to null

buckets.add(null);

}

size = 0;

numBuckets \*= 2;

for (int i = 0; i < temp.size(); i++)

{

MapNode<K, V> head = temp.get(i);

while (head != null)

{

K key = head.key;

V val = head.value;

insert(key, val);

head = head.next;

}

}

System.out.println("Rehashing Done \n");

}

public void printMap()

{

// The present bucket list is made temp

ArrayList<MapNode<K, V>> temp = buckets;

System.out.println("Current HashMap:");

// loop through all the nodes and print them

for (int i = 0; i < temp.size(); i++)

{

MapNode<K, V> head = temp.get(i);

while (head != null)

{

System.out.println("key = " + head.key + ", val = " + head.value);

head = head.next;

}

}

System.out.println();

}

public static void main(String[] args)

{

Scanner scanner= new Scanner(System.in);

int count=1;

ReHashMap<Integer, String> reHashMap = new ReHashMap<Integer, String>();

String word;

do

{

System.out.println("Insert object for HashMap, '0' for exit");

word=scanner.nextLine();

if (!word.equals("0"))

{

reHashMap.insert(count, word);

reHashMap.printMap();

count++;

}

} while (!word.equals("0"));

}

}

## 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

## Код программы

public class chess {

static int S = 0;

public static void main(String[] args) {

int n=8;

queen(n);

System.out.println("\nВсего решений: " + S);

}

static int[] recQueen(int[] p, int k) {

int n = p.length;

if (k == n) return p;

for (int j = 1; j <= n; j++) {

boolean correct = true;

for (int i = 0; i < k; i++) {

if (p[i] == j || k - i == Math.abs(j - p[i])) {

correct = false;

break;

}

}

if (correct) {

p[k] = j ;

int[] pos = recQueen(p, k+1);

if (pos != null) {

S++;

printBoard(pos);

}

}

}

return null;

}

static void queen(int n) {

recQueen(new int[n], 0);

}

static void printBoard(int[] pos) {

System.out.println("\nРешение №" + S );

for (int i = 0; i < pos.length; i++) {

int queenPos = pos[i];

for (int k = 1; k < queenPos; k++) {

System.out.print("\_ ");

}

System.out.print("Q ");

for (int k = queenPos + 1; k <= pos.length; k++) {

System.out.print("\_ ");

}

System.out.print("\n");

}

}

}

## Снимки экрана работы программ

Задача 1

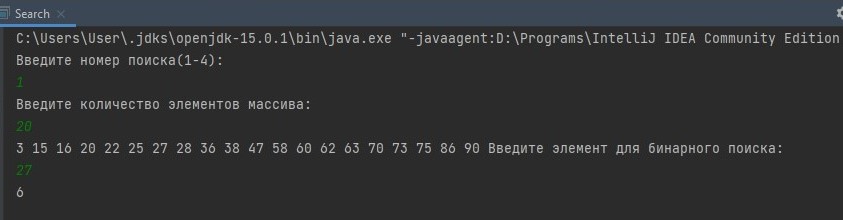


Рисунок 1 – Результат выполнения бинарного поиска

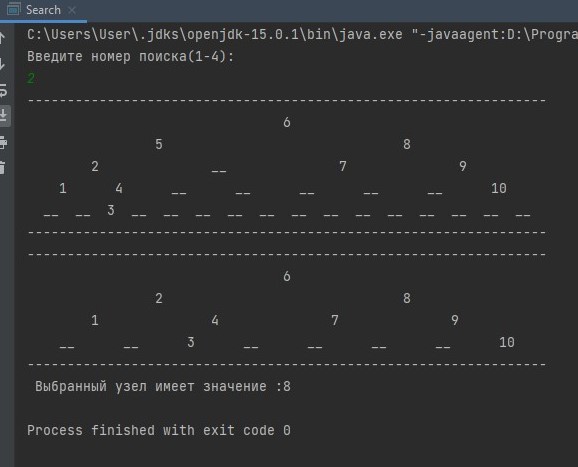


Рисунок 2 – Результат выполнения поиска бинарным деревом

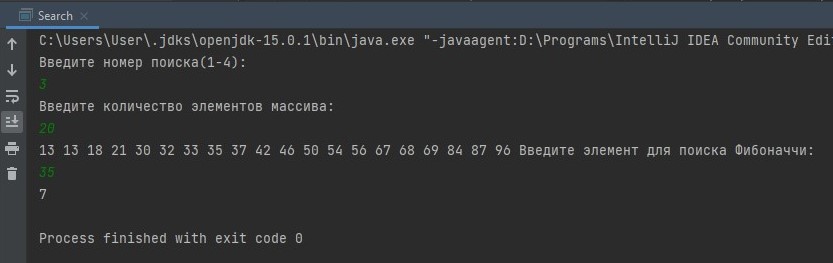


Рисунок 3 – Результат выполнения поиска фибоначчи

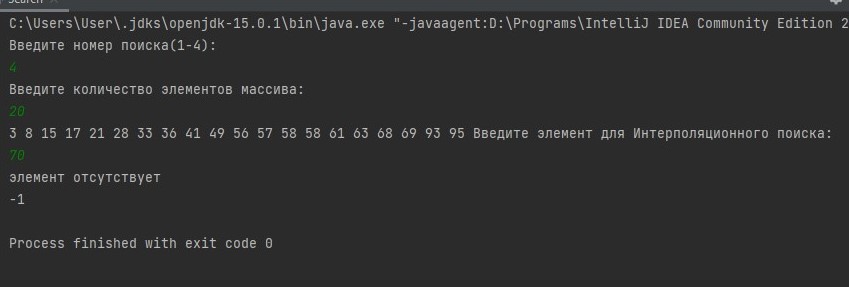


Рисунок 4 – Результат выполнения интерполяционного поиска

Задача2

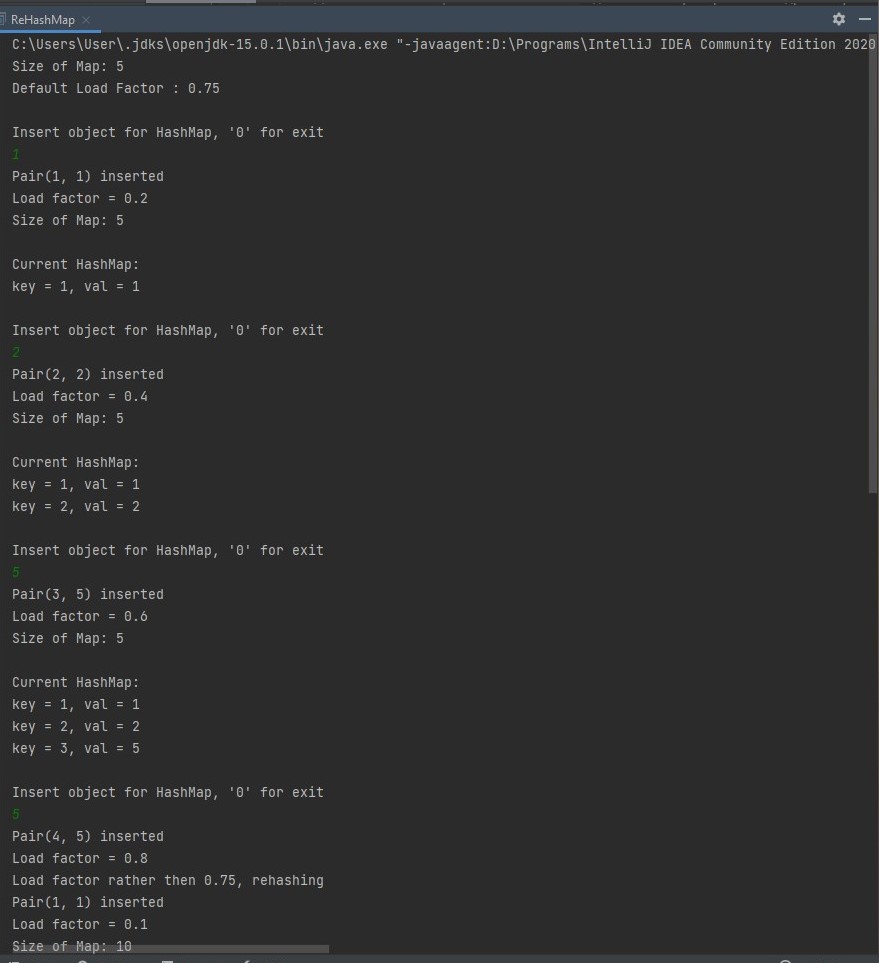


Рисунок 5

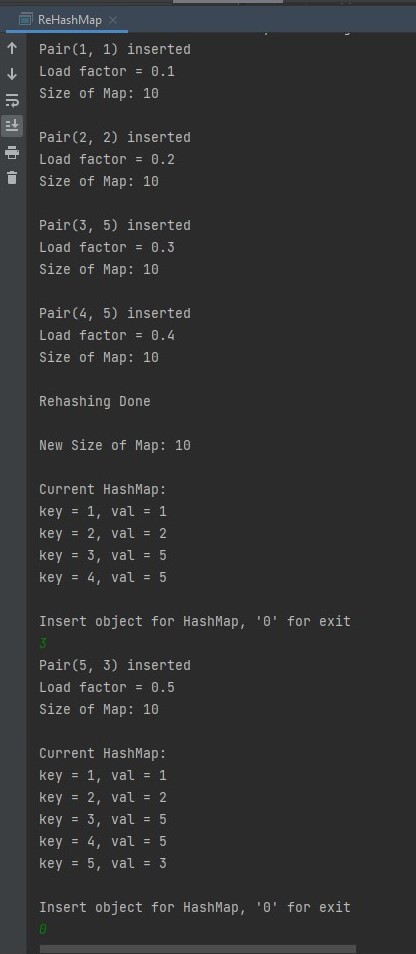


Рисунок 6 – Результат выполнения рехэширования

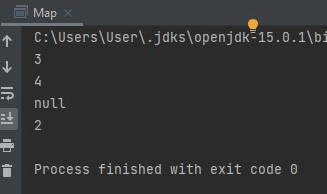


Рисунок 7 – Результат выполнения метода цепоцек

Задача 3

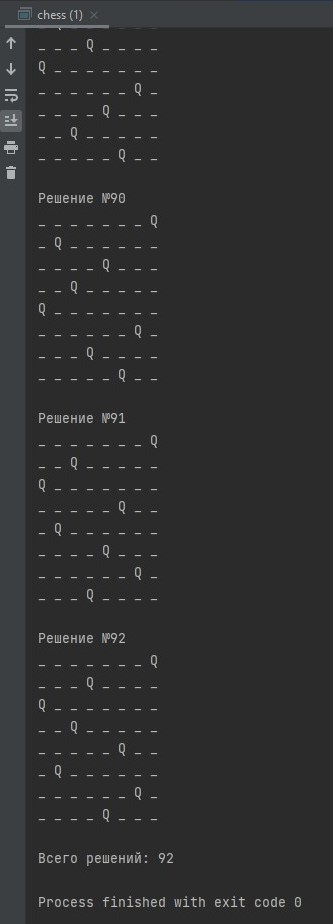


Рисунок 8

## Вывод

Я реализовала методы поиска, простое рехеширование и метод цепочек в соответствии с заданием, организовала генерацию начального набора случайных данных и оценила время работы каждого алгоритма поиска. А так же написала алгоритм решения задачи про 8 ферзей.