

Федеральное агентство связи

**Государственное бюджетное образовательное учреждение
высшего образования
ордена Трудового Красного Знамени
«Московский технический университет связи и информатики»**

**Кафедра
«Математическая кибернетика и информационные технологии»**

**Лабораторная работа №2
по дисциплине «Структура и алгоритмы обработки данных»
По теме «Методы поиска»**

Выполнил студент группы БФИ 1902

Леонов Н.Н.

Научный руководитель:

Мкртчян Г.М.

Москва 2021

Задание на лабораторную работу: 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.

Ход выполнения лабораторной работы

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

Листинг:

```
package com.company;
import java.util.Stack;
import java.util.Scanner;

// Методы поиска
//Лабораторная работа №2
public class Main {

    public static void main(String[] args) {
        System.out.println("Задание №1");
        System.out.println("*БИНАРНЫЙ ПОИСК* ");
        Scanner sc = new Scanner(System.in);
        int n;
        System.out.println("Введите длину массива: ");
        n = sc.nextInt();

        int arr[] = new int[n];
        inArr(arr,n);
        selectionSort(arr); //сортируем

        System.out.println("Введите искомый элемент: ");
        int el = sc.nextInt();

        System.out.print("Индекс искомого элемента равен: ");
        System.out.println(binaryPoisk(arr,el));

        System.out.println(" ----- ");

        System.out.println("*ПОИСК БИНАРНЫМ ДЕРЕВОМ* ");
        Tree tree = new Tree();
        // вставляем узлы в дерево:
        for (int i = 0;i<arr.length;i++)
        {
            tree.insertNode(arr[i]);
        }
        System.out.println(tree.findNodeByValue(el));
    }
}
```

```

        System.out.println(" ----- ");
        System.out.println("*ФИБОНАЧИЕВ ПОИСК* ");
        FibonacciSearch F = new FibonacciSearch();
        int index = F.search(arr,el);
        System.out.println(index);

        System.out.println(" ----- ");
        System.out.println("*ИНТЕРПОЛЯЦИОННЫЙ ПОИСК* ");
        System.out.println(interpolationSearch(arr,el));
    }

```

```

    public static void inArr(int[] arr,int n){
        for (int i = 0; i < n; i++){
            Scanner sc = new Scanner(System.in);
            arr[i] = sc.nextInt();
        }
    }

```

```

    public static void selectionSort(int[] arr){
        for (int i = 0; i < arr.length; i++) {

            int min = arr[i];
            int min_i = i;
            for (int j = i+1; j < arr.length; j++) {
                //Если находим, запоминаем его индекс
                if (arr[j] < min) {
                    min = arr[j];
                    min_i = j;
                }
            }
            if (i != min_i) {
                int tmp = arr[i];
                arr[i] = arr[min_i];
                arr[min_i] = tmp;
            }
        }
    }

```

```

    // Binary поиск
    public static int binaryPoisk(int[] arr,int element){
        int startIn = 0;
        int endIn = arr.length - 1;
        while (startIn<=endIn){
            int middleIn = startIn + (endIn - startIn) / 2;
            if (arr[middleIn]==element){
                return middleIn;
            }
            if (arr[middleIn]>element){
                endIn = middleIn - 1;
            }
            else {
                startIn = middleIn + 1;
            }
        }
        return -1;
    }

```

```

    // Binary Tree поиск
    static 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 +
            '}';
    }
}

static 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 пока 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,
                System.out.print("Элемент не найден: ");
                return null; // возвращаем null
            }
        }
        System.out.print("Элемент найден: ");
        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
            }
        }
    }
}

// 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) { // пока 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
    }
}

//Фибоначиев поиск
public static class FibonachySearch{
    private int i;
    private int p;
    private int q;
    private boolean stop = false;

    private void init(int[] arr){
        stop = false;
        int k = 0;
        int n = arr.length;
        for(; getFibonachyNumber(k+1) < n+1;){
            k +=1;
        }
        int m = getFibonachyNumber(k+1) - (n+1);
        i = getFibonachyNumber(k) - m;
    }
}

```

```

        p = getFibonachyNumber(k-1);
        q = getFibonachyNumber(k-2);
    }

    public int getFibonachyNumber(int k){
        int firstNumber = 0;
        int secondNumber = 1;
        for (int i = 0; i < k; i++){
            int temp = secondNumber;
            secondNumber += firstNumber;
            firstNumber = temp;
        }
        return firstNumber;
    }

    private void upIndex(){
        if (p==1)
            stop = true;
        i = i + q;
        p = p - q;
        q = q - p;
    }

    private void downIndex(){
        if (q==0)
            stop = true;
        i = i - q;
        int temp = q;
        q = p - q;
        p = temp;
    }

    public int search(int[] arr, int element){
        init(arr);
        int n = arr.length;
        int resIn = -1;
        for (; !stop;){
            if (i < 0){
                upIndex();
            }
            else if (i >= n){
                downIndex();
            }
            else if (arr[i] == element){
                resIn = i;
                break;
            }
            else if (element < arr[i]){
                downIndex();
            }
            else if (element > arr[i]){
                upIndex();
            }
        }
        return resIn;
    }
}

//Интерполяционный поиск
public static int interpolationSearch(int[] sortedArray, int toFind) {
    // Возвращает индекс элемента со значением toFind или -1, если такого
    // элемента не существует
    int mid;

```

```
int low = 0;
int high = sortedArray.length - 1;

while (sortedArray[low] < toFind && sortedArray[high] > toFind) {
    if (sortedArray[high] == sortedArray[low]) // Защита от деления
на 0
        break;
    mid = low + ((toFind - sortedArray[low]) * (high - low)) /
(sortedArray[high] - sortedArray[low]);

    if (sortedArray[mid] < toFind)
        low = mid + 1;
    else if (sortedArray[mid] > toFind)
        high = mid - 1;
    else
        return mid;
}

if (sortedArray[low] == toFind)
    return low;
if (sortedArray[high] == toFind)
    return high;

return -1;
}
```

Результат выполнения кода представлен на рисунке 1.

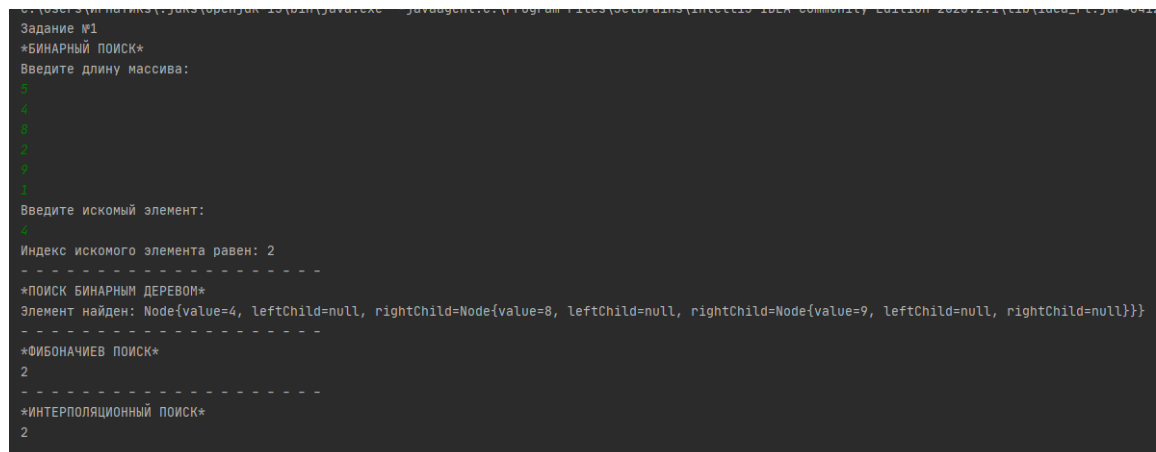


Рисунок 1 – Первое задание

Задание №2:

Простое рехэширование	Рехэширование помощью псевдослучайных чисел	с Метод цепочек

Код для задания:

```
package com.company;

public class HashTable {

    //массив для хранения элементов
    private Item[] table;
    //количество элементов в таблице
    private int count;
    //размер таблицы
    private int size;

    public HashTable(int size) {
        this.size = size;
        table = new Item[size];
    }

    public int hash(String key)
    {
        int hash = 0;

        for(int i = 0; i < key.length(); i++)
            hash = (31 * hash + key.charAt(i)) % size;

        return hash;
    }

    public void insert(String key) {
        Item item = new Item(key);
        int hash = hash(key);
        while (table[hash] != null) {
            hash++;
            hash %= size;
        }
        table[hash] = item;
    }

    public void print()
    {
        for(int i = 0; i < size; i++)
            if(table[i] != null)
                System.out.println(i + " " + table[i].getKey());
    }

    public Item find(String key)
    {
        int hash = hash(key);
        while(table[hash] != null)
        {
            if(table[hash].getKey().equals(key))
                return table[hash];
            hash++;
            hash = hash % size;
        }

        return null;
    }
}
```

```
package com.company;

import java.util.Scanner;
```

```

public class Rehashing {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        HashTable hashTable = new HashTable(128);
        hashTable.insert("rhino");
        hashTable.insert("man");
        hashTable.insert("computer");
        hashTable.insert("home");
        hashTable.insert("basket");
        hashTable.print();
        String word = scanner.nextLine();
        Item item = hashTable.find(word);
        if (item != null)
            System.out.println("Элемент найден, его хэш: " +
hashTable.hash(word));
        else
            System.out.println("Элемент не найден!");
    }
}
class Item{

    private String key;

    public Item(String key)
    {
        this.key = key;
    }

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

}

```

```

package com.company;

public class HashTable_random {

    //массив для хранения элементов
    private Item[] table;
    //количество элементов в таблице
    private int count;
    //размер таблицы
    private int size;

    public HashTable_random(int size) {
        this.size = size;
        table = new Item[size];
    }

    public int hash_random(String key)
    {
        double hash=0;
        double R = 1;

        for(int i = 0; i < key.length(); i++)
            R=5*R;
            R=R%(4*size);
            hash=Math.floor(R/4);
    }
}

```

```

        return (int)hash;
    }

    public void insert(String key) {
        Item item = new Item(key);
        int hash = hash_random(key);
        while (table[hash] != null) {
            hash++;
            hash %= (4*size);
        }
        table[hash] = item;
    }

    public void print()
    {
        for(int i = 0; i < size; i++)
            if(table[i] != null)
                System.out.println(i + " " + table[i].getKey());
    }

    public Item find(String key)
    {
        int hash = hash_random(key);
        while(table[hash] != null)
        {
            if(table[hash].getKey().equals(key))
                return table[hash];
            hash++;
            hash = hash % (4*size);
        }

        return null;
    }
}

```

Результат выполнения кода представлен на рисунках 2,3 и 4.

```

2 van
15 computer
43 home
60 rhina
61 rhino
64 basket
rhina
Элемент найден, его хэш: 60

```

Рисунок 2 – Простое рехэширование

```

13 rhino
28 home
31 man
66 basket
120 computer
400000
Элемент найден, его хэш: 66

```

Рисунок 3 – Рэхеширование псевдослучайными числами

```

Ключ: значение
0: absolutely deal emphasize violate obtain range
1: briefcase assign belong satisfy inverse once
2: require law failure domain
3: set tes
4:
5: unique shorthand
6: identity confuse
Введите слово для поиска: set
Такое слово есть.

```

Рисунок 4 – 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.

Код для задания:

```

public class Queen {
    /**
     * размерность доски
     */
    /**
     * хранит расстановку ферзей. каждый ферзь находится на отдельной линии,
на
     * одной линии находится не могут так как бьют друг друга.
     */
    private int[] state;
    /**
     * Порядковый номер комбинации
     */
    private int index = 1;

    /**

```

```

    * n - размерность доски и количество ферзей
    */
    public Queen(int n) {

        state = new int[n];

        for (int i = 0; i < state.length; i++) {
            state[i] = 0;
        }
    }

    /*
    * генерирует следующую комбинацию (расстановку фигур)
    */
    public boolean next() {
        index++;
        return move(8 - 1);
    }

    /*
    * Двигает фигуру в указанной линии на одну клетку вправо и возвращает
    true.
    * Если фигура находится в крайнем положении, то фигура устанавливается в
    * первое положение и двигается фигура находящаяся на линии выше и так
    далее.
    * Если линий выше не осталось возвращает false.
    */
    private boolean move(int index) {
        if (state[index] < 8 - 1) {
            state[index]++;
            return true;
        }

        state[index] = 0;
        if (index == 0) {
            return false;
        } else {
            return move(index - 1);
        }
    }

    /*
    * Возвращает порядковый номер комбинации, которая в данный момент
    * установлена.
    */
    public int getIndex() {
        return index;
    }
}

//проверяем бьет ли наша королева другую фигуру если да то возвращаем фолс
public boolean isPeace() {
    for (int i = 0; i < state.length; i++) {
        for (int j = i + 1; j < state.length; j++) {
            // бьет ли по вертикали
            if (state[i] == state[j]) {
                return false;
            }
            // бьет ли по диагонали
            if (Math.abs(i - j) == Math.abs(state[i] - state[j])) {
                return false;
            }
        }
    }

    return true;
}

```

```

    }

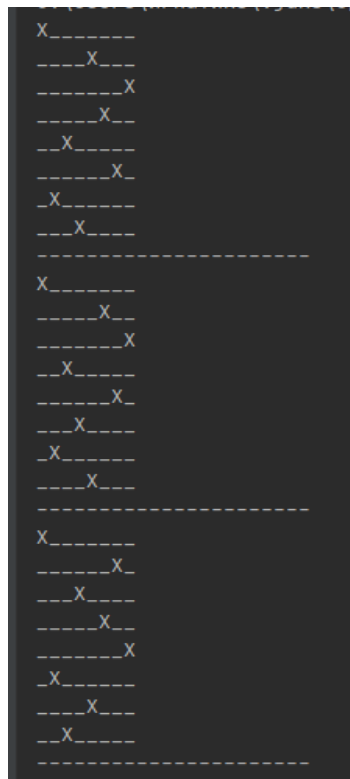
    /*
     * Выводит доску с фигурами.
     */
    public void printState() {
        for (int i = 0; i < state.length; i++) {
            int position = state[i];
            for (int j = 0; j < 8 ; j++) {
                System.out.print(j == position ? 'X' : '_');
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {
        Queen c = new Queen(8);
        int counter = 0;
        do {
            if (c.isPeace()) {
                counter++;
                c.printState();
                System.out.println(" -----");
            }
        } while (c.next());

        System.out.println("Итого: " + counter);
    }
}

```

Результат выполнения кода представлен на рисунке 5.



```

X_ _ _ _ _
_ _ _ X _ _
_ _ _ _ X
_ _ _ X _
_ X _ _ _
_ _ _ _ X
_ X _ _ _
_ _ X _ _
-----
X_ _ _ _ _
_ _ _ X _
_ _ _ _ X
_ X _ _ _
_ _ _ _ X
_ X _ _ _
_ X _ _ _
_ _ X _ _
-----
X_ _ _ _ _
_ _ _ X _
_ X _ _ _
_ _ _ X _
_ _ _ _ X
_ X _ _ _
_ _ X _ _
_ X _ _ _
-----

```

Рисунок 5 – Способы решения

