**Федеральное агентство связи**

**Государственное бюджетное образовательное учреждение высшего Образование**

**Ордена Трудового Красного Знамени**

**«Московский технический университет связи и информатики»**

**Кафедра «МКиИТ»**

**дисциплина «СиАОД»**

Отчет по Лабораторной работе №2

Подготовил студент

группы БВТ1902: Капленко Е. М.

Руководитель: Мкртчян Г. М.

Москва 2020

## Лабораторная работа 2. Методы поиска.

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** | **Фибоначчиев** | **Интерполяционный** |

### Задание №2:

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

В первом задании были реализованы различные алгоритмы поиска элементов:

```
public static int binarySearch(int arr[], int elementToSearch) {

    int firstIndex = 0;
    int lastIndex = arr.length - 1;

    while(firstIndex <= lastIndex) {
        int middleIndex = (firstIndex + lastIndex) / 2;
        // если средний элемент - целевой элемент, вернуть его индекс
        if (arr[middleIndex] == elementToSearch) {
            return middleIndex;
        }

        // если средний элемент меньше
        // направляем наш индекс в middle+1, убирая первую часть из
рассмотрения
        else if (arr[middleIndex] < elementToSearch)
            firstIndex = middleIndex + 1;

            // если средний элемент больше
            // направляем наш индекс в middle-1, убирая вторую часть из
рассмотрения
        else if (arr[middleIndex] > elementToSearch)
            lastIndex = middleIndex - 1;

    }
    return -1;
```

```
}

public static int interpolationSearch(int integers[], int elementToSearch) {

    int startIndex = 0;
    int lastIndex = (integers.length - 1);

    while ((startIndex <= lastIndex) && (elementToSearch >=
integers[startIndex]) &&
            (elementToSearch <= integers[lastIndex])) {
        // используем формулу интерполяции для поиска возможной лучшей
позиции для существующего элемента
        int pos = startIndex + (((lastIndex-startIndex) /
                (integers[lastIndex]-integers[startIndex]))*
                (elementToSearch - integers[startIndex]));

        if (integers[pos] == elementToSearch)
            return pos;

        if (integers[pos] < elementToSearch)
            startIndex = pos + 1;

        else
            lastIndex = pos - 1;
    }
    return -1;
}
static int min(int x, int y) { return (x <= y) ? x : y; }
// искомый элемент x , n - размер массива
static int fibMonaccianSearch(int arr[], int x, int n)
{
    int fibMMm2 = 0; // (m-2)'th Fibonacci No.
    int fibMMm1 = 1; // (m-1)'th Fibonacci No.
    int fibM = fibMMm2 + fibMMm1; // m'th Fibonacci

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

    int offset = -1;

    while (fibM > 1) {
        int i = min(offset + fibMMm2, n - 1);

        if (arr[i] < x) {
            fibM = fibMMm1;
            fibMMm1 = fibMMm2;
            fibMMm2 = fibM - fibMMm1;
            offset = i;
        }

        else if (arr[i] > x) {
            fibM = fibMMm2;
            fibMMm1 = fibMMm1 - fibMMm2;
            fibMMm2 = fibM - fibMMm1;
        }

        else
            return i;
    }

    if (fibMMm1 ==x && arr[offset + 1] == x)
```

```
        return offset + 1;

    return -1;
}
public class BinaryTree {
private Node rootNode; // корневой узел

public BinaryTree() { // 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()); // 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
```

```
        }
}

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 +
                '}';
    }
}
```

Во втором задании реализованы 3 способа рехеширования:

```
ublic static class Map<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;
    int size;
    int numBuckets;
    final double DEFAULT_LOAD_FACTOR = 0.75;
    public Map()
    {
        numBuckets = 5;
        buckets = new ArrayList<>(numBuckets);
        for (int i = 0; i < numBuckets; i++) {
            buckets.add(null);
        }
        System.out.println("HashMap созданный");
        System.out.println("\n" + "Количество пар на Map: " + size);
        System.out.println("Размер Map: " + numBuckets);
        System.out.println("Коэффициент нагрузки по умолчанию : " +
DEFAULT_LOAD_FACTOR + "\n");
    }

    private int getBucketInd(K key)
    {
        int hashCode = key.hashCode();
        return (hashCode % numBuckets);
    }

    public void insert(K key, V value)
    {
        int bucketInd = getBucketInd(key);
        MapNode<K, V> head = buckets.get(bucketInd);
          /*  while (head != null) {

                // Если уже присутствует, значение обновляется
                if (head.key.equals(key)) {
                    head.value = value;
                    return;
                }
                head = head.next;
            }*/

        MapNode<K, V> newElementNode = new MapNode<K, V>(key, value);
        head = buckets.get(bucketInd);
        newElementNode.next = head;
        buckets.set(bucketInd, newElementNode);
        System.out.println("Пара (" + key + ", " + value + ") вставлено
успешно.\n");
        size++;

        double loadFactor = (1.0 * size) / numBuckets;

        System.out.println("Текущий коэффициент нагрузки = " + loadFactor);

        if (loadFactor > DEFAULT_LOAD_FACTOR) {
            System.out.println(loadFactor + " больше, чем " +
DEFAULT_LOAD_FACTOR);
            System.out.println("Поэтому повторное хеширование будет
выполнено.\n");
            // Rehash
            rehash();
            System.out.println("Новый размер для Map: " + numBuckets + "\n");
        }
        System.out.println("Количество пар в Map: " + size);
        System.out.println("Размер Map: " + numBuckets + "\n");
    }
    private void rehash() {
        System.out.println("\nНачало рехеширования\n");
```

```
        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("\nРехеширование закончилось\n");
    }

    public void printMap() {
        ArrayList<MapNode<K, V> > temp = buckets;
        System.out.println("Построенный HashMap:");
        for (int i = 0; i < temp.size(); i++) {
            MapNode<K, V> head = temp.get(i);
            while (head != null) {
                System.out.println("ключ = " + head.key + ", значение = " +
head.value);
                head = head.next;
            }
        }
        System.out.println();
    }
}

public static class HashNode<K, V> {
    K key;
    V value;

    HashNode<K, V> next;

    public HashNode(K key, V value)
    {
        this.key = key;
        this.value = value;
    }
}

public static class Map2<K, V> {
    private ArrayList<HashNode<K, V>> bucketArray;
    private int numBuckets;
    private int size;

    public Map2() {
        bucketArray = new ArrayList<>();
        numBuckets = 10;
        size = 0;
        for (int i = 0; i < numBuckets; i++)
            bucketArray.add(null);
    }

    public int size() {
```

```
        return size;
    }

    public boolean isEmpty() {
        return size() == 0;
    }

    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;
    }
    public V get(K key) {
        int bucketIndex = getBucketIndex(key);
        HashNode<K, V> head = bucketArray.get(bucketIndex);
        while (head != null) {
            if (head.key.equals(key))
                return head.value;
            head = head.next;
        }
        return null;
    }

    public void add(K key, V value) {
        int bucketIndex = getBucketIndex(key);
        HashNode<K, V> head = bucketArray.get(bucketIndex);
        while (head != null) {
            if (head.key.equals(key)) {
                head.value = value;
                return;
            }
            head = head.next;
        }
        size++;
        head = bucketArray.get(bucketIndex);
        HashNode<K, V> newNode
                = new HashNode<K, V>(key, value);
        newNode.next = head;
        bucketArray.set(bucketIndex, newNode);
        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;
                }
            }
        }
    }
}
```

Вывод: Реализованы бинарный, фибоначиев, интерполяционный поиски и поиск в бинарном дереве. Так же реализовано рехеширование.