

МИНОБРАЗОВАНИЯ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего
образования
НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий
Кафедра информатики и систем управления

ОТЧЕТ
по лабораторной работе
по дисциплине:
Алгоритмы и структуры данных

РУКОВОДИТЕЛЬ:
Капранов С.Н.

(подпись)
(фамилия, и.,о.)

СТУДЕНТ:
Еричев Д.А.

(подпись)
(фамилия, и.,о.)

(шифр группы)

Работа защищена «___» _____
С оценкой _____

Реализовать алгоритмы поиска, вставки, удаления элементов таблицы и распечатки таблицы и метод ре-хэширования таблицы при увеличении размера данных.

Вариант №10.

Ключ - Номер телефона

Хэш-функция - функции работы со строковыми ключами

Метод разрешения коллизий - Метод цепочек

1) Программный код.

```
package Laba5;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.List;
import java.util.Scanner;
import java.util.LinkedList;
public class BTree<Key extends Comparable<Key>> {
    // максимальное количество потомков на узел B-дерева = M-1
    // (должно быть четным и больше 2)
    private static final int size = 4;
    int code = 0; //переменная для
    int a = 127; // вспомогательный коэффициент для создания хэша
    private Node root; // корень B-дерева
    private int height; // высота B-дерева
    private int countChildren; // количество пар ключ-
значение в B-дереве
    static LinkedList<String> currHashArr; //вспомогательный
список для создания списка коллизий
    static String[] hashArr; //вспомогательный массив для
заполнения списка коллизий
    static LinkedList<LinkedList> lst; //список для хранения
коллизий
    private static final class Node { //узел со значениями
        private int childrenSize; // количество
детей
        private Entry[] children = new Entry[size]; // массив
детей
        private Node(int k) {
            childrenSize = k;
        }
    }

    // внутренние узлы: используйте только ключ и следующий
    // внешние узлы: используйте только ключ и значение
    private static class Entry { // класс для работы с ключами и
детьми
        private Comparable key; //ключи
        private Node next; // узел для перебора следующих
элементов
        public Entry(Comparable key, Node next) {
            this.key = key;
            this.next = next;
        }
    }
}
```

```

    }
    private String hashString = ""; //строка для хэша
}
//функция работы с коллизиями
static void chainAdd() {
    lst = new LinkedList<>(); //инициализация списка
коллизий
    hashArr = new String[currHashArr.size()]; //всп. массив
    for(int i=0;i<currHashArr.size();i++) {
        hashArr[i] = currHashArr.get(i); //запоминаем хэши
    }
    LinkedList currLst;
    for(int i=0;i<hashArr.length;i++) {
        int counter = 0;
        currLst = new LinkedList<>();
        for(int j=i+1;j<hashArr.length;j++) {
            //если нашли совпадающие хэши
            if(hashArr[i].compareTo(hashArr[j]) == 0) {
                //считаем количество
                ++counter;
                //меняем значение на пустую строку
                hashArr[j] = "";
            }
            //если дошли до последнего элемента и есть
совпадения
            if(j == hashArr.length-1 && counter != 0) {
                for(int k=0;k<=counter;k++) {
                    //добавляем в список совпадающий хэш
                    currLst.add(hashArr[i]);
                }
            }
            //добавляем список в - главный список коллизий
            lst.add(currLst);
        }
        //зачищаем пустые строки полученные в строке 59
        for(int i=0;i<lst.size();i++) {
            //если длина списка > 1 и элемент- пустая строка
            if(lst.get(i).size() > 1 && lst.get(i).getLast() ==
"" ) {
                //очищаем список
                lst.get(i).clear();
            }
        }
        //вывод списка коллизий
        for(LinkedList el: lst) {
            System.out.println(el);
        }
    }

    //Инициализирует пустое B-дерево.
    public BTree() {
        root = new Node(0);
    }
}

```

```

    }
    public int height() {
        return height;
    }
    //получение элемента
    public String get(String key) {
        if (key == null) throw new
IllegalArgumentException("argument to get() is null");
        System.out.println();
        return search(root, key, height);
    }
    //получение hash-a
    private String search(Node x, String key, int ht) {
        Entry[] children = x.children;

        // если лист
        if (ht == 0) {
            for (int j = 0; j < x.childrenSize; j++) {
                String s = (String) children[j].key;
                if (key.compareTo(s) == 0) {
                    return children[j].hashString;
                }
            }
        }

        // если не лист
        else {
            for (int j = 0; j < x.childrenSize; j++) {
                String s = (String) children[j].key;
                if (j+1 == x.childrenSize || less(key,
children[j+1].key))
                    return search(children[j].next, key, ht-1);
            }
        }
        return null;
    }

    public void delete(String hash, BTree tree) {
        if(hash!=null) {
            deleteElement(root, hash, height,tree);
        }
    }
    private void deleteElement(Node x, String hash, int ht,BTree
tree) {
        Entry[] children = x.children;

        // external node
        if (ht == 0) {
            for (int j = 0; j < x.childrenSize; j++) {
                if (eq(hash, children[j].hashString)) {
                    children[j] = null;
                    return;
                }
            }
        }
    }

```

```

        }
    }
    // internal node
    else {
        for (int j = 0; j < x.childrenSize; j++) {
            if(children[j] != null) {
                deleteElement(children[j].next, hash, ht-
1,tree);
            }
        }
    }
}

public void put(Key key) {
    if (key == null) throw new
IllegalArgumentException("argument key to put() is null");
    Node u = insert(root, key, height);
    countChildren++;
    if (u == null) return;

    // need to split root
    Node t = new Node(2);
    t.children[0] = new Entry(root.children[0].key, root);
    t.children[1] = new Entry(u.children[0].key, u);
    root = t;
    height++;
}
private Node insert(Node node, Key key, int ht) {
    int j;
    Entry currentNode = new Entry(key,null);

    // если лист
    if (ht == 0) {
        for (j = 0; j < node.childrenSize; j++) {
            if (eq(key, node.children[j].key)) break;
        }
    }

    // если не лист
    else {
        for (j = 0; j < node.childrenSize; j++) {
            if ((j+1 == node.childrenSize) || less(key,
node.children[j+1].key)) {
                Node newNode = insert(node.children[j+1].next,
key, ht-1);

                if (newNode == null) return null;
                currentNode.key = newNode.children[0].key;
                currentNode.next = newNode;
                break;
            }
        }
    }
}

```

```

        for (int i = node.childrenSize; i > j; i--){
            node.children[i] = node.children[i-1];
        }
        node.children[j] = currentNode;
        node.childrenSize++;
        if (node.childrenSize < size) {
            return null;
        }
        else return split(node);

    }

    public void onHash() {
        currHashArr = new LinkedList<>();
        Hash(root, height);
//        chainAdd();
    }

    public void Hash(Node h, int ht) {
        if(h!=null) {
            Entry[] children = h.children;
            if (ht == 0) {
                String key="";
                for (int j = 0; j < h.childrenSize; j++) {
                    String s = (String) children[j].key;
                    String[] arrStr = s.split("■");
//                    int code = 0;
//                    int a = 127;
                    for(int i=0;i<arrStr.length;i++) {
                        String[] arrKey = arrStr[i].split("■");
                        for(int k=0;k<arrKey.length;k++) {
                            code += (char) a*code +
arrKey[k].charAt(i);
//                            a++;
                        }
                    }
                    key+=String.format("%04X", code);
                    children[j].hashString = key;
                    currHashArr.add(children[j].hashString);
                }
            }

            else{
                for (int j = 0; j < h.childrenSize; j++) {
                    Hash(children[j].next, ht-1);
                }
            }
        }
    }

    // split node in half
    private Node split(Node h) {
        Node t = new Node(size/2);
        h.childrenSize = size/2;
        for (int j = 0; j < size/2; j++)

```

```

        t.children[j] = h.children[size/2+j];
    return t;
}

/**
 * @return строковое представление этого B-дерева.
 */
public String toString() {
    System.out.println("Таблица:");
    System.out.println("Ключ:          |   Хэш:
|");
    System.out.println("_____|_____
_____");

    return toString(root, height, "") + "\n";
}

private String toString(Node h, int ht, String indent) {
    StringBuilder s = new StringBuilder();
    Entry[] children = h.children;
    //если дошли до листа
    if (ht == 0) {
        //проверяем все элементы полученные ранее
        for (int j = 0; j < h.childrenSize; j++) {
            if(children[j] != null) {
                System.out.println(children[j].key + "
+" " +
                children[j].hashString + "
");
            }
        }
    }
    //если не лист
    else {
        for (int j = 0; j < h.childrenSize; j++) {
            toString(children[j].next, ht-1, indent);
        }
    }
    return s.toString();
}

private boolean less(Comparable k1, Comparable k2) {
    return k1.compareTo(k2) < 0;
}

private boolean eq(Comparable k1, Comparable k2) {
    return k1.compareTo(k2) == 0;
}

private boolean more(Comparable k1, Comparable k2) {
    return k1.compareTo(k2) > 0;
}

public static void main(String[] args) throws

```

```

FileNotFoundException {
    BTree<String> tree = new BTree<String>();

    readInfoTree("/home/denis/JavaPrograms/src/Laba5/file.txt",
tree);//чтение элементов из файла
        tree.onHash();
        tree.toString();
        System.out.println("\nТаблица коллизий:");
        chainAdd();

    delElement("/home/denis/JavaPrograms/src/Laba5/deleteElement.txt",
tree);//удаление элемента
        getElem("/home/denis/JavaPrograms/src/Laba5/search.txt",
tree);
    }

    static void readInfoTree(String path, BTree tree) throws
FileNotFoundException {
        Scanner sc = new Scanner(new File(path));
        int size = sc.nextInt();
        while (size != 0) {
            size--;
            tree.put(sc.next());//вставка элементов
        }
    }

    //функция для работы с файлом
    static String readElement(String path) throws
FileNotFoundException {
        Scanner sc = new Scanner(new File(path));
        return sc.next();
    }

    static void getElem(String path, BTree tree) throws
FileNotFoundException{
        String deleteElement = readElement(path);//удаляемый
элемент
        String getHash = tree.get(deleteElement);//получаем хэш
        System.out.println("Элемент: "+deleteElement);
        System.out.println("Хэш получаемого элемента: " +
getHash);
    }

    static void delElement(String path, BTree tree) throws
FileNotFoundException {
        String deleteElement = readElement(path);//удаляемый
элемент
        String getHash = tree.get(deleteElement);//получаем хэш
        System.out.println("Удаляемый элемент: "+deleteElement);
        System.out.println("Хэш удаляемого элемента: " + getHash);
        tree.delete(getHash,tree);//удаление элемента
        tree.toString();
    }
}

```


2)Описание функций:

static void chainAdd() - функция работы с коллизиями

public String get(String key) - получение элемента

private String search(Node x, String key, int ht) - получение hash-a

public void delete(String hash, BTree tree) - удаление элемента из таблицы

private void deleteElement(Node x, String hash, int ht, BTree tree) - удаление

public void put(Key key) - добавление элемента

private Node insert(Node node, Key key, int ht) - добавление

public void Hash(Node h, int ht) - создание хэша

private Node split(Node h) - разделение узла дерева

private String toString(Node h, int ht, String indent) - вывод дерева

static void readInfoTree(String path, BTree tree) - считывание из файла

static void getElem(String path, BTree tree) - функция вызова получения элемента

static void delElement(String path, BTree tree) - функция удаления элемента

3) Результаты работы программы.

Вывод в консоль:

Таблица:

Ключ:	Хэш:
8-900-55-34-44	872D59B4
8-920-30-85-32	872D59B4872CD0C33
8-903-21-59-92	872C9AB9
8-903-21-59-91	872C9AB9872C9AB9
8-903-21-59-99	872C9AB9
8-903-21-59-99	872C9AB9872C9AB9
8-903-21-59-99	872C9AB9872C9AB9872C9AB9
8-920-30-85-32	872CD0C33
8-904-83-97-64	872CD0C33872E1CB6
8-923-50-60-43	872D5B34
8-785-12-49-54	872D5B3486EC5A35
8-932-85-69-54	872D5B3486EC5A35872E1B35
8-951-60-33-88	872D99B8
8-943-12-34-89	872D99B88872C59B8

Таблица коллизий:

```
[ ]  
[ ]  
[872C9AB9, 872C9AB9]  
[872C9AB9872C9AB9, 872C9AB9872C9AB9]  
[ ]  
[ ]  
[ ]  
[ ]  
[ ]  
[ ]  
[ ]  
[ ]  
[ ]  
[ ]
```

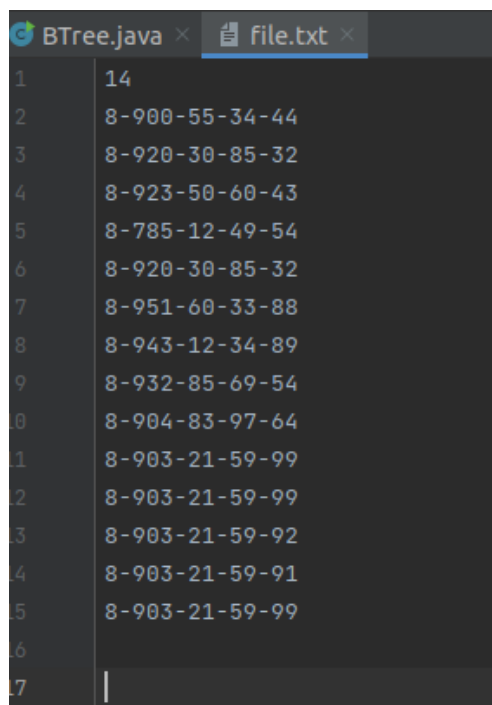
Таблица:

```
Удаляемый элемент: 8-900-55-34-44
Хэш удаляемого элемента: 872D59B4
Таблица:
Ключ:      | Хэш:      |
-----|-----|
8-920-30-85-32      872D59B4872CDC33
8-903-21-59-92      872C9AB9
8-903-21-59-91      872C9AB9872C9AB9
8-903-21-59-99      872C9AB9
8-903-21-59-99      872C9AB9872C9AB9
8-903-21-59-99      872C9AB9872C9AB9872C9AB9
8-920-30-85-32      872CDC33
8-904-83-97-64      872CDC33872E1CB6
8-923-50-60-43      872D5B34
8-785-12-49-54      872D5B3486EC5A35
8-932-85-69-54      872D5B3486EC5A35872E1B35
8-951-60-33-88      872D99B8
8-943-12-34-89      872D99B8872C59B8

Элемент: 8-920-30-85-32
Хэш получаемого элемента: 872CDC33

Process finished with exit code 0
```

Считывание из файла:



```
BTtree.java x file.txt x
1 14
2 8-900-55-34-44
3 8-920-30-85-32
4 8-923-50-60-43
5 8-785-12-49-54
6 8-920-30-85-32
7 8-951-60-33-88
8 8-943-12-34-89
9 8-932-85-69-54
10 8-904-83-97-64
11 8-903-21-59-99
12 8-903-21-59-99
13 8-903-21-59-92
14 8-903-21-59-91
15 8-903-21-59-99
16
17
```