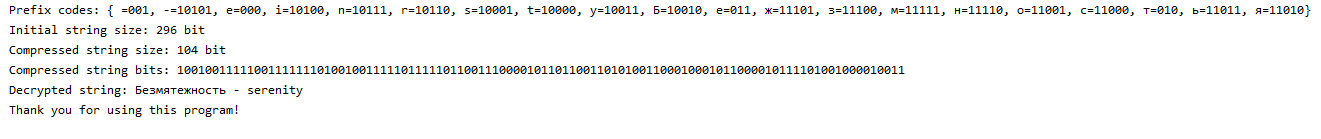
**Листинг практической работы № 15**

package ru.mirea.ikbo20.AOD.pr15;  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Map;  
import java.util.TreeMap;  
  
public class Main {  
 public static void main(String[] args) {  
  
 String text = "Безмятежность - serenity";  
  
 *// Вычисление частоты символов в тексте* TreeMap<Character, Integer> frequencies = *countFrequency*(text);  
  
 *// Генерация списка листов дерева* ArrayList<CodeTreeNode> codeTreeNodes = new ArrayList<>();  
 for(Character c: frequencies.keySet()) {  
 codeTreeNodes.add(new CodeTreeNode(c, frequencies.get(c)));  
 }  
  
 *// Постройка кодового дерева с помощью алгоритма Хаффмана* CodeTreeNode tree = *huffman*(codeTreeNodes);  
  
 *// Генерация таблицы префиксных кодов для кодируемых символов с помощью кодового дерева* TreeMap<Character, String> codes = new TreeMap<>();  
 for(Character c: frequencies.keySet()) {  
 codes.put(c, tree.getCodeForCharacter(c, ""));  
 }  
  
 System.*out*.println("Prefix codes: " + codes.toString());  
  
 *// Кодировка текста, замена сиволов соответствующими кодами* StringBuilder encoded = new StringBuilder();  
 for (int i = 0; i < text.length(); i++) {  
 encoded.append(codes.get(text.charAt(i)));  
 }  
  
 System.*out*.println("Initial string size: " + text.getBytes().length \* 8 + " bit");  
 System.*out*.println("Compressed string size: " + encoded.length() + " bit");  
 System.*out*.println("Compressed string bits: " + encoded);  
  
 *// Декодирование сжатой информации в исходный вид* String decoded = *huffmanDecode*(encoded.toString(), tree);  
  
 System.*out*.println("Decrypted string: " + decoded + "\nThank you for using this program!");  
 }  
  
 private static TreeMap<Character, Integer> countFrequency(String text) {  
 TreeMap<Character, Integer> freqMap = new TreeMap<>();  
 for (int i = 0; i < text.length(); i++) {  
 Character c = text.charAt(i);  
 Integer count = freqMap.get(c);  
 freqMap.put(c, count != null ? count + 1 : 1);  
 }  
 return freqMap;  
 }  
  
 private static CodeTreeNode huffman(ArrayList<CodeTreeNode> codeTreeNodes) {  
 while (codeTreeNodes.size() > 1) {  
 Collections.*sort*(codeTreeNodes);  
 CodeTreeNode left = codeTreeNodes.remove(codeTreeNodes.size() - 1);  
 CodeTreeNode right = codeTreeNodes.remove(codeTreeNodes.size() - 1);  
  
 CodeTreeNode parent = new CodeTreeNode(null, right.weight + left.weight, left, right);  
 codeTreeNodes.add(parent);  
 }  
 return codeTreeNodes.get(0);  
 }  
  
 private static String huffmanDecode(String encoded, CodeTreeNode tree) {  
 StringBuilder decoded = new StringBuilder();  
  
 CodeTreeNode node = tree;  
 for (int i = 0; i < encoded.length(); i++) {  
 node = encoded.charAt(i) == '0' ? node.left : node.right;  
 if (node.content != null) {  
 decoded.append(node.content);  
 node = tree;  
 }  
 }  
 return decoded.toString();  
 }  
  
 *// Класс для представления кодового дерева* private static class CodeTreeNode implements Comparable<CodeTreeNode> {  
  
 Character content;  
 int weight;  
 CodeTreeNode left;  
 CodeTreeNode right;  
  
 public CodeTreeNode(Character content, int weight) {  
 this.content = content;  
 this.weight = weight;  
 }  
  
 public CodeTreeNode(Character content, int weight, CodeTreeNode left, CodeTreeNode right) {  
 this.content = content;  
 this.weight = weight;  
 this.left = left;  
 this.right = right;  
 }  
  
 public int compareTo(CodeTreeNode o) {  
 return o.weight - weight;  
 }  
  
 *// Извлечение кода символа* public String getCodeForCharacter(Character ch, String parentPath) {  
 if (content == ch) {  
 return parentPath;  
 } else {  
 if (left != null) {  
 String path = left.getCodeForCharacter(ch, parentPath + 0);  
 if (path != null) {  
 return path;  
 }  
 }  
 if (right != null) {  
 return right.getCodeForCharacter(ch, parentPath + 1);  
 }  
 }  
 return null;  
 }  
 }  
  
 *// Класс, реализующий битовый массив* public static class BitArray {  
 int size;  
 byte[] bytes;  
  
 private byte[] masks = new byte[] {0b00000001, 0b00000010, 0b00000100, 0b00001000,  
 0b00010000, 0b00100000, 0b01000000, (byte) 0b10000000};  
  
 public BitArray(int size) {  
 this.size = size;  
 int sizeInBytes = size / 8;  
 if (size % 8 > 0) {  
 sizeInBytes = sizeInBytes + 1;  
 }  
 bytes = new byte[sizeInBytes];  
 }  
  
 public BitArray(int size, byte[] bytes) {  
 this.size = size;  
 this.bytes = bytes;  
 }  
  
 public int get(int index) {  
 int byteIndex = index / 8;  
 int bitIndex = index % 8;  
 return (bytes[byteIndex] & masks[bitIndex]) != 0 ? 1 : 0;  
 }  
  
 public void set(int index, int value) {  
 int byteIndex = index / 8;  
 int bitIndex = index % 8;  
 if (value != 0) {  
 bytes[byteIndex] = (byte) (bytes[byteIndex] | masks[bitIndex]);  
 } else {  
 bytes[byteIndex] = (byte) (bytes[byteIndex] & ~masks[bitIndex]);  
 }  
 }  
  
 public String toString() {  
 StringBuilder sb = new StringBuilder();  
 for (int i = 0; i < size; i++) {  
 sb.append(get(i) > 0 ? '1' : '0');  
 }  
 return sb.toString();  
 }  
  
 public int getSize() {  
 return size;  
 }  
  
 public int getSizeInBytes() {  
 return bytes.length;  
 }  
  
 public byte[] getBytes() {  
 return bytes;  
 }  
 }  
  
 *// Сохранение таблицы частот и сжатой информации в файл* private static void saveToFile(File output, Map<Character, Integer> frequencies, String bits) {  
 try {  
 DataOutputStream os = new DataOutputStream(new FileOutputStream(output));  
 os.writeInt(frequencies.size());  
 for (Character character: frequencies.keySet()) {  
 os.writeChar(character);  
 os.writeInt(frequencies.get(character));  
 }  
 int compressedSizeBits = bits.length();  
 BitArray bitArray = new BitArray(compressedSizeBits);  
 for (int i = 0; i < bits.length(); i++) {  
 bitArray.set(i, bits.charAt(i) != '0' ? 1 : 0);  
 }  
  
 os.writeInt(compressedSizeBits);  
 os.write(bitArray.bytes, 0, bitArray.getSizeInBytes());  
 os.flush();  
 os.close();  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *// Загрузка сжатой информации и таблицы частот из файла* private static void loadFromFile(File input, Map<Character, Integer> frequencies, StringBuilder bits) {  
 try {  
 DataInputStream os = new DataInputStream(new FileInputStream(input));  
 int frequencyTableSize = os.readInt();  
 for (int i = 0; i < frequencyTableSize; i++) {  
 frequencies.put(os.readChar(), os.readInt());  
 }  
 int dataSizeBits = os.readInt();  
 BitArray bitArray = new BitArray(dataSizeBits);  
 os.read(bitArray.bytes, 0, bitArray.getSizeInBytes());  
 os.close();  
  
 for (int i = 0; i < bitArray.size; i++) {  
 bits.append(bitArray.get(i) != 0 ? "1" : 0);  
 }  
  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

**Демонстрация работы программы**

****