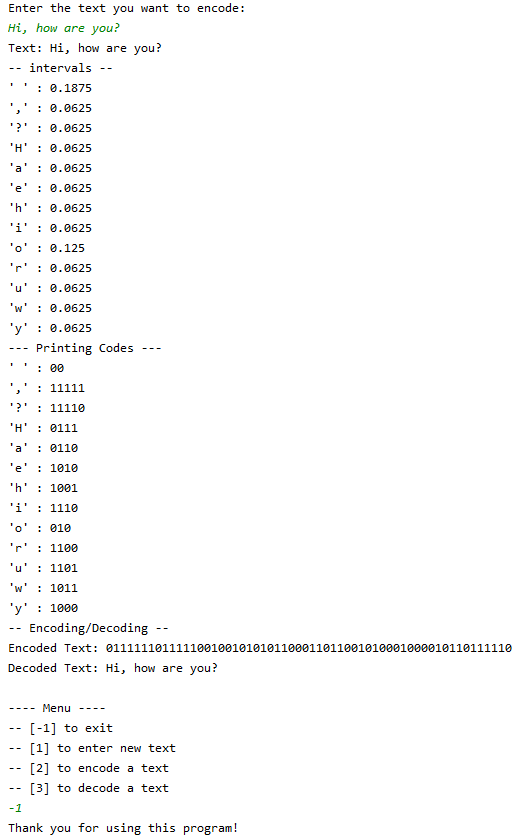
**Листинг практической работы № 12**

package ru.mirea.ikbo20.AOD;  
import java.util.Objects;  
import java.util.PriorityQueue;  
import java.util.Scanner;  
import java.util.TreeMap;  
  
public class Node {  
 Node left, right;  
 double value;  
 String character;  
  
 public Node(double value, String character) {  
 this.value = value;  
 this.character = character;  
 left = null;  
 right = null;  
 }  
  
 public Node(Node left, Node right) {  
 this.value = left.value + right.value;  
 character = left.character + right.character;  
 if (left.value < right.value) {  
 this.right = right;  
 this.left = left;  
 } else {  
 this.right = left;  
 this.left = right;  
 }  
 }  
}  
  
class Huffman {  
 static final boolean *readFromFile* = false;  
 static final boolean *newTextBasedOnOldOne* = false;  
  
 static PriorityQueue<Node> *nodes* =  
 new PriorityQueue<>((o1, o2) -> (o1.value < o2.value) ? -1 : 1);  
 static TreeMap<Character, String> *codes* = new TreeMap<>();  
 static String *text* = "";  
 static String *encoded* = "";  
 static String *decoded* = "";  
 static int[] *ASCII* = new int[128];  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 int decision = 1;  
 while (decision != -1) {  
 if (*handlingDecision*(scanner, decision)) continue;  
 decision = *consoleMenu*(scanner);  
 }  
 }  
  
 private static int consoleMenu(Scanner scanner) {  
 int decision;  
 System.*out*.println("\n---- Menu ----\n" +  
 "-- [-1] to exit \n" +  
 "-- [1] to enter new text\n" +  
 "-- [2] to encode a text\n" +  
 "-- [3] to decode a text");  
 decision = Integer.*parseInt*(scanner.nextLine());  
 if (*readFromFile*)  
 System.*out*.println("Decision: " + decision +  
 "\n---- End of Menu ----\n");  
 return decision;  
 }  
  
 private static boolean handlingDecision(Scanner scanner, int decision) {  
 if (decision == 1) {  
 if (*handleNewText*(scanner)) return true;  
 } else if (decision == 2) {  
 if (*handleEncodingNewText*(scanner)) return true;  
 } else if (decision == 3) {  
 *handleDecodingNewText*(scanner);  
 }  
 return false;  
 }  
  
 private static void handleDecodingNewText(Scanner scanner) {  
 System.*out*.println("Enter the text to decode:");  
 *encoded* = scanner.nextLine();  
 System.*out*.println("Text to Decode: " + *encoded*);  
 *decodeText*();  
 }  
  
 private static boolean handleEncodingNewText(Scanner scanner) {  
 System.*out*.println("Enter the text to encode:");  
 *text* = scanner.nextLine();  
 System.*out*.println("Text to Encode: " + *text*);  
  
 if (*IsSameCharacterSet*()) {  
 System.*out*.println("Invalid input");  
 *text* = "";  
 return true;  
 }  
 *encodeText*();  
 return false;  
 }  
  
 private static boolean handleNewText(Scanner scanner) {  
 int oldTextLength = *text*.length();  
 System.*out*.println("Enter the text you want to encode:");  
 *text* = scanner.nextLine();  
 if (*newTextBasedOnOldOne* && (oldTextLength != 0 &&  
 *IsSameCharacterSet*())) {  
 System.*out*.println("Invalid input, please try again");  
 *text* = "";  
 return true;  
 }  
 *ASCII* = new int[128];  
 *nodes*.clear();  
 *codes*.clear();  
 *encoded* = "";  
 *decoded* = "";  
 System.*out*.println("Text: " + *text*);  
 *calculateCharIntervals*(*nodes*);  
 *buildTree*(*nodes*);  
 *generateCodes*(*nodes*.peek(), "");  
  
 *printCodes*();  
 System.*out*.println("-- Encoding/Decoding --");  
 *encodeText*();  
 *decodeText*();  
 return false;  
 }  
  
 private static boolean IsSameCharacterSet() {  
 boolean flag = true;  
 for (int i = 0; i < *text*.length(); i++)  
 if (*ASCII*[*text*.charAt(i)] == 0) {  
 flag = false;  
 break;  
 }  
 return !flag;  
 }  
  
 private static void decodeText() {  
 *decoded* = "";  
 Node node = *nodes*.peek();  
 for (int i = 0; i < *encoded*.length(); ) {  
 Node tmpNode = node;  
 while (tmpNode.left != null && tmpNode.right != null &&  
 i < *encoded*.length()) {  
 if (*encoded*.charAt(i) == '1')  
 tmpNode = tmpNode.right;  
 else tmpNode = tmpNode.left;  
 i++;  
 }  
 if (tmpNode.character.length() == 1)  
 *decoded* += tmpNode.character;  
 else  
 System.*out*.println("Input not Valid");  
  
 }  
 System.*out*.println("Decoded Text: " + *decoded*);  
 }  
  
 private static void encodeText() {  
 *encoded* = "";  
 for (int i = 0; i < *text*.length(); i++)  
 *encoded* += *codes*.get(*text*.charAt(i));  
 System.*out*.println("Encoded Text: " + *encoded*);  
 }  
  
 private static void buildTree(PriorityQueue<Node> vector) {  
 while (vector.size() > 1)  
 vector.add(new Node(vector.poll(), Objects.*requireNonNull*(vector.poll())));  
 }  
  
 private static void printCodes() {  
 System.*out*.println("--- Printing Codes ---");  
 *codes*.forEach((k, v) -> System.*out*.println("'" + k + "' : " + v));  
 }  
  
 private static void calculateCharIntervals(PriorityQueue<Node> vector) {  
 System.*out*.println("-- intervals --");  
  
 for (int i = 0; i < *text*.length(); i++)  
 *ASCII*[*text*.charAt(i)]++;  
  
 for (int i = 0; i < *ASCII*.length; i++)  
 if (*ASCII*[i] > 0) {  
 vector.add(new Node(*ASCII*[i] / (*text*.length() \* 1.0),  
 ((char) i) + ""));  
 System.*out*.println("'" + ((char) i) + "' : " +  
 *ASCII*[i] / (*text*.length() \* 1.0));  
 }  
 }  
  
 private static void generateCodes(Node node, String s) {  
 if (node != null) {  
 if (node.right != null)  
 *generateCodes*(node.right, s + "1");  
  
 if (node.left != null)  
 *generateCodes*(node.left, s + "0");  
  
 if (node.left == null && node.right == null)  
 *codes*.put(node.character.charAt(0), s);  
 }  
 }  
}

**Демонстрация работы программы**

****