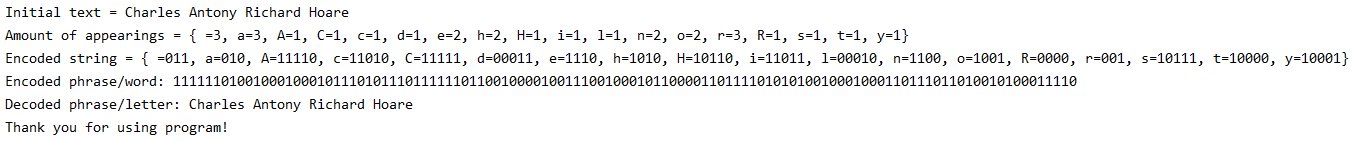
**Листинг практической работы № 14**

package ru.mirea.ikbo20.AOD.pr14;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.PriorityQueue;  
import java.util.Set;  
  
public class HuffmanCodeSolution {  
  
 private static Map<Character, String> *charPrefixHashMap* = new HashMap<>();  
 static HuffmanNode *root*;  
  
 public static void main(String[] args) {  
  
 String test = "Charles Antony Richard Hoare";  
 System.*out*.println("Initial text = " + test);  
 String s = *encode*(test);  
 *decode*(s);  
  
 }  
  
 private static HuffmanNode buildTree(Map<Character, Integer> freq) {  
  
 PriorityQueue<HuffmanNode> priorityQueue = new PriorityQueue<>();  
 Set<Character> keySet = freq.keySet();  
 for (Character c : keySet) {  
  
 HuffmanNode huffmanNode = new HuffmanNode();  
 huffmanNode.data = c;  
 huffmanNode.frequency = freq.get(c);  
 huffmanNode.left = null;  
 huffmanNode.right = null;  
 priorityQueue.offer(huffmanNode);  
 }  
 assert priorityQueue.size() > 0;  
  
 while (priorityQueue.size() > 1) {  
  
 HuffmanNode x = priorityQueue.peek();  
 priorityQueue.poll();  
  
 HuffmanNode y = priorityQueue.peek();  
 priorityQueue.poll();  
  
 HuffmanNode sum = new HuffmanNode();  
  
 assert y != null;  
 sum.frequency = x.frequency + y.frequency;  
 sum.data = '-';  
  
 sum.left = x;  
  
 sum.right = y;  
 *root* = sum;  
  
 priorityQueue.offer(sum);  
 }  
  
 return priorityQueue.poll();  
 }  
  
  
 private static void setPrefixCodes(HuffmanNode node, StringBuilder prefix) {  
  
 if (node != null) {  
 if (node.left == null && node.right == null) {  
 *charPrefixHashMap*.put(node.data, prefix.toString());  
  
 } else {  
 prefix.append('0');  
 *setPrefixCodes*(node.left, prefix);  
 prefix.deleteCharAt(prefix.length() - 1);  
  
 prefix.append('1');  
 *setPrefixCodes*(node.right, prefix);  
 prefix.deleteCharAt(prefix.length() - 1);  
 }  
 }  
  
 }  
  
 private static String encode(String test) {  
 Map<Character, Integer> freq = new HashMap<>();  
 for (int i = 0; i < test.length(); i++) {  
 if (!freq.containsKey(test.charAt(i))) {  
 freq.put(test.charAt(i), 0);  
 }  
 freq.put(test.charAt(i), freq.get(test.charAt(i)) + 1);  
 }  
  
 System.*out*.println("Amount of appearings = " + freq);  
 *root* = *buildTree*(freq);  
  
 *setPrefixCodes*(*root*, new StringBuilder());  
 System.*out*.println("Encoded string = " + *charPrefixHashMap*);  
 StringBuilder s = new StringBuilder();  
  
 for (int i = 0; i < test.length(); i++) {  
 char c = test.charAt(i);  
 s.append(*charPrefixHashMap*.get(c));  
 }  
  
 return s.toString();  
 }  
  
 private static void decode(String s) {  
  
 StringBuilder stringBuilder = new StringBuilder();  
  
 HuffmanNode temp = *root*;  
  
 System.*out*.println("Encoded phrase/word: " + s);  
  
 for (int i = 0; i < s.length(); i++) {  
 int j = Integer.*parseInt*(String.*valueOf*(s.charAt(i)));  
  
 if (j == 0) {  
 temp = temp.left;  
 if (temp.left == null && temp.right == null) {  
 stringBuilder.append(temp.data);  
 temp = *root*;  
 }  
 }  
 if (j == 1) {  
 temp = temp.right;  
 if (temp.left == null && temp.right == null) {  
 stringBuilder.append(temp.data);  
 temp = *root*;  
 }  
 }  
 }  
  
 System.*out*.println("Decoded phrase/letter: "  
 + stringBuilder.toString() + "\nThank you for using program!");  
  
 }  
}  
  
class HuffmanNode implements Comparable<HuffmanNode> {  
 int frequency;  
 char data;  
 HuffmanNode left, right;  
  
 public int compareTo(HuffmanNode node) {  
 return frequency - node.frequency;  
 }  
}

**Демонстрация работы программы**

****