**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине: **«Визуальные средства разработки программных**

**приложений»**

на тему: **Коллекции**

Выполнил: студент гр. ИТП-31

Дашкевич Д.А.

Принял: ассистент

Белявский Е. В.

Гомель 2019

**Цель**: изучить работу с коллекциями в языке программирования *Java*.

**Ход работы**

**Вариант 2**

# **Задание:** написать программу, осуществляющую сжатие английского текста. Построить для каждого слова в тексте оптимальный префиксный код по алгоритму Хаффмана. Использовать класс PriorityQueue.

**Результат работы программы:**

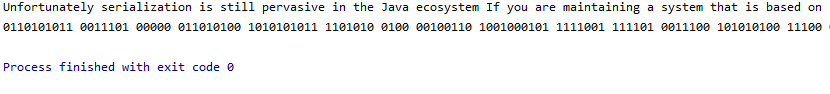
****

Рисунок 1 – Вывод в консоль работы программы, верхняя строка исходный текст, нижняя строка текст после сжатия алгоритмом Хаффмана

**Вывод:** в результате выполнения лабораторной работы была изучена работа с коллекциями в языке программирования *Java,* написана программа, использующая коллекции для сжатия английского текста.

**Приложение А**

**Исходный код программы**

**Main.java:**

**public class** Main {  
 **public static void** main(String[] args) {  
 String text = *enterTextFromFile*();  
 String convertedText = Huffman.*convert*(text);  
 System.***out***.println(text);  
 System.***out***.println(convertedText);  
 }  
  
 **private static** String enterTextFromFile() {  
 String text = **""**;  
 **try** (Scanner file = **new** Scanner(**new** FileReader(**"text.txt"**))) {  
 text = file.useDelimiter(**"\\A"**).next();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** text;  
 }  
}

**Huffman.java:**

**import** java.util.\*;  
  
**class** Huffman {  
 **static** String convert(String text) {  
 String[] words = text.split(**" "**);  
  
 List<Node> priorityWords = **new** ArrayList<>();  
 *filtering*(priorityWords, words);  
  
 Queue<Node> queue = **new** PriorityQueue<>(Comparator.*comparingInt*(Node::getCount));  
 queue.addAll(priorityWords);  
 *createTree*(queue);  
  
 Map<String, String> codeTable = **new** HashMap<>();  
 **if** (queue.size() == 1) {  
 *createCodeTable*(queue.poll(), **new** StringBuilder(), codeTable);  
 }  
  
 StringBuilder builder = **new** StringBuilder();  
 **for** (String word : words) {  
 builder.append(codeTable.get(word)).append(**' '**);  
 }  
 **return** builder.toString();  
 }  
  
 **private static void** createCodeTable(Node node, StringBuilder builder, Map<String, String> codeTable) {  
 **if** (node.**word** != **null**) {  
 codeTable.put(node.**word**, builder.toString());  
 **return**;  
 }  
 *createCodeTable*(node.**leftChild**, builder.append(**'0'**), codeTable);  
 builder.deleteCharAt(builder.length() - 1);  
 *createCodeTable*(node.**rightChild**, builder.append(**'1'**), codeTable);  
 builder.deleteCharAt(builder.length() - 1);  
 }  
  
 **private static void** createTree(Queue<Node> queue) {  
 **while** (queue.size() > 1) {  
 Node node1 = queue.poll();  
 Node node2 = queue.poll();  
 **if** (node1 != **null** && node2 != **null**) {  
 Node node = **new** Node(node1, node2);  
 queue.add(node);  
 } **else** {  
 **throw new** NullPointerException(**"Node1 or Node2 are null!"**);  
 }  
 }  
 }  
  
 **private static void** filtering(List<Node> priorityWords, String[] words) {  
 **for** (String s : **new** TreeSet<>(Arrays.*asList*(words))) {  
 priorityWords.add(**new** Node(s));  
 }  
  
 **for** (Node priorityWord : priorityWords) {  
 **for** (String word : words) {  
 **if** (priorityWord.getWord().equals(word)) {  
 priorityWord.addToCount();  
 }  
 }  
 }  
 }  
  
  
 **private static class** Node {  
 **private** String **word**;  
 **private int count**;  
 **private** Node **leftChild**;  
 **private** Node **rightChild**;  
  
 **private** Node(String word, **int** count) {  
 **this**.**word** = word;  
 **this**.**count** = count;  
 **this**.**leftChild** = **null**;  
 **this**.**rightChild** = **null**;  
 }  
  
 **private** Node(String word) {  
 **this**(word, 0);  
 }  
  
 **private** Node(Node o1, Node o2) {  
 **this**(**null**, o1.**count** + o2.**count**);  
 **leftChild** = o1;  
 **rightChild** = o2;  
 }  
  
 **private void** addToCount() {  
 **count**++;  
 }  
  
 **private** String getWord() {  
 **return word**;  
 }  
  
 **private int** getCount() {  
 **return count**;  
 }  
 }  
}