THE MINISTRY OF EDUCATION AND SCIENCE OF THE REPUBLIC OF KAZAKHSTAN

INTERNATIONAL INFORMATION TECHNOLOGIES UNIVERSITY

Faculty of Information technologies

Department of Computer Systems, Software Engineering and Telecommunications

**Project report**

**Course: Information theory**

|  |  |
| --- | --- |
| Student: Nurtai Maksat  Group: CSSE-1605  Accepted : senior-lecturer L.A. Kozina |  |

Almaty 2015

**CONTENTS**

[**INTRODUCTION** 3](#_Toc7204441)

[**PROJECT IMPLEMENTATION** 4](#_Toc7204442)

# **INTRODUCTION**

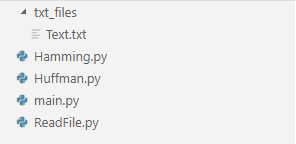
This project was written to learn how to use algorithms to transmit information through a noisy channel. In this project, the Huffman and Hamming algorithms were used.

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding and/or using such a code proceeds by means of Huffman coding, an algorithm developed by David A. Huffman while he was a Sc.D. student at MIT, and published in the 1952 paper "A Method for the Construction of Minimum-Redundancy Codes".

In telecommunication, Hamming codes are a family of linear error-correcting codes. Hamming codes can detect up to two-bit errors or correct one-bit errors without detection of uncorrected errors. By contrast, the simple parity code cannot correct errors, and can detect only an odd number of bits in error. Hamming codes are perfect codes, that is, they achieve the highest possible rate for codes with their block length and minimum distance of three. Richard W. Hamming invented Hamming codes in 1950 as a way of automatically correcting errors introduced by punched card readers. In his original paper, Hamming elaborated his general idea, but specifically focused on the Hamming(7,4) code which adds three parity bits to four bits of data.

# **PROJECT IMPLEMENTATION**

First, you need the data, that is, the text to work. We will retrieve this data from the txt file. Let me show you the structure of the project. It looks like this:



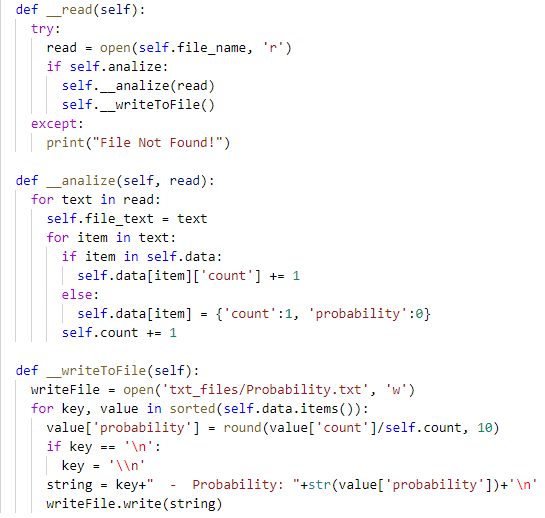
*Figure 1. Project structure*

The project was written in Python. The main file of my project is main.py. The very first step is to read the Text.txt file. To do this, the main.py file calls a method from the ReadFile.py file that reads the data from the file.



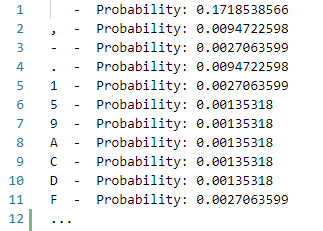
*Figure 2. From main.py 22 line*

As you can see, we pass the “analize” parameter, so that after reading the file, it will automatically read the probabilities of the characters and write to the file.



*Figure 3. ReadFile.py*

After launch, we will see it as output written in the file:

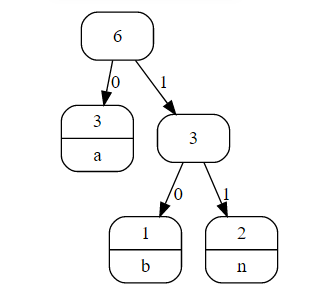


*Figure 4. Probability.txt*

The next thing we need to do is implement the Huffman code algorithm to take the prefix code at the heart of our probabilities. The idea behind the Huffman coding is based on the frequency of occurrence of a character in a sequence. The symbol that occurs in the sequence most often receives a new very small code, and the symbol that occurs least often receives, on the contrary, a very long code. This is necessary, since we want, when we have processed the entire input, the most frequency symbols occupy the least space, and the rarest - more. For this algorithm, you will need a minimal understanding of the structure of the binary tree and the priority queue. To build a tree, I used an array. The array was used in this way, with each iteration of the loop I added the first two elements to the array, sorted in ascending characters. After each iteration (adding to the array), I again sorted the list in ascending order. The array looked like this when the words "banana" were used:

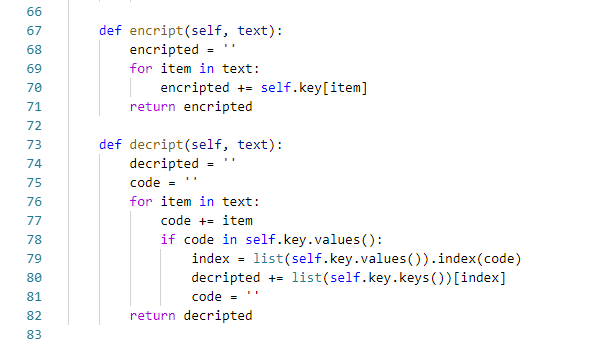
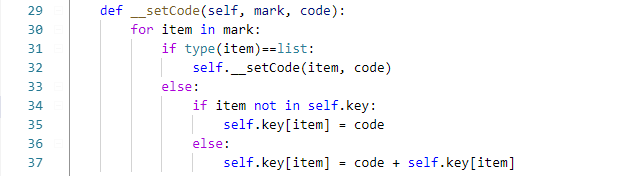
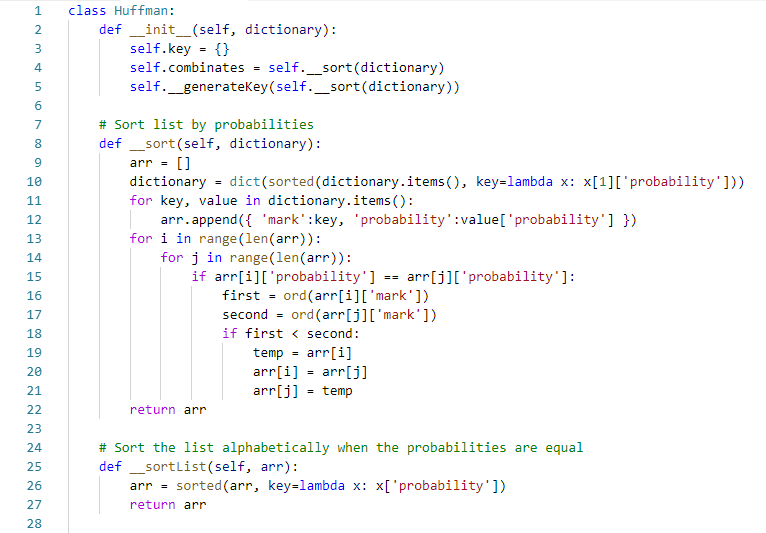


The tree would look like this:



*Figure 5. Example of Huffman tree*

The following is the screenshots of the code:



The result of the Huffman algorithm is recorded in the Keys.txt file.

