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: Tokhtakhunov Il'murat  Group: CSSE - 1607  Accepted : senior-lecturer L.A. Kozina |  |

Almaty 2019

**Content**

**INTRODUCTION3**

**MAIN PART4**

Part 14

Part 25

Part 37

Part 47

Part 58

Part 69

**CONCLUSION10**

**REFERENCES11**

INTRODUCTION

Our project is Communication system. In this project we had to illustrate the process which starts from sending source of information and ends with received information to the user. The main goal of the work that we done is understand and get basic knowledge about how it works, what kind of problem there can be appeared and etc. During the project doing process we got profitable and usable in real life experience about information theory, loss of information, recovering and so on. Although, whole project was divided into several parts, it did not loose its main purpose it even made project more understandable and specific. Also, each part had its own goal and we surely have view that we got all that points.

In addition, we will show how we solved every problem by one by in the main part of project, what kind of method did we use, inputs, outputs, probabilities, encoding, decoding, errors and results all of them are included.

## **MAIN PART** Part 1

Goal: C++ (or another) program finds all symbols (not only letters) in this text and computes their probabilities.  
 Initially we have a file “Text.txt”, and we must find out the probability of every symbol of this whole Text.   
 The idea of our code is take the first symbol from the Text and count it(how many times the symbol repeats). After counting, remove the chosen symbol from Text.txt. Then, code chooses next one and begins the cycle that we wrote before. Also, cycle will work till the text ends. Following this, it divides quantity of each symbol to amount of symbol of whole text. Finally, we print the output and can be seen that probability calculation of every symbol, for instance:   
*{'char': 'I', 'abs\_prob': 0.05953991880920163, 'rel\_prob': '44/739'}*

Output: Figure - 1, Figure - 2.

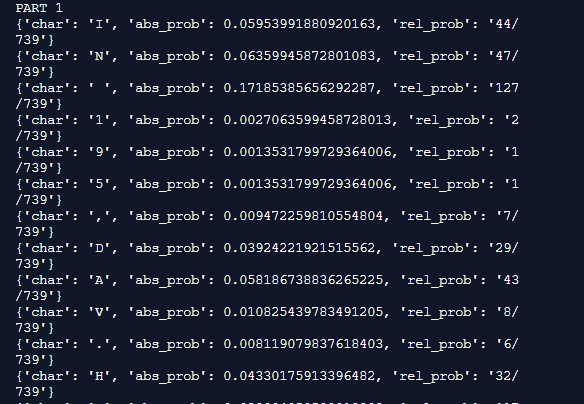


Figure - 1. List of symbols probability

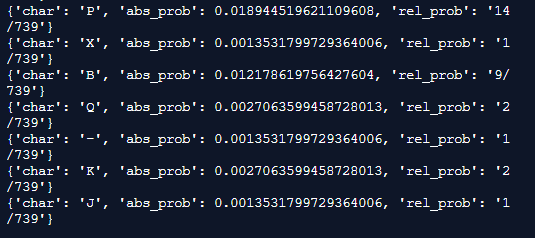


Figure - 2. List of symbols probability

## Part 2

Goal: Encode the text from the “Text.txt” file using Shannon-Fano or Huffman algorithm with probabilities from Part 1.

Take the result of previous part. Sorted probability by descending order. Following this, joined the lowest 2 values(probability of symbols), and repeat it again and again till there would not left only one value. Result of cycle: *Node ((((H|((W|V)|U))|E)|((R|((Y|B)|(G|(((9|5)|1)|.))))|(A|I)))|((((M|(L|C))|N)|(T|(((((J|(X|-))|(K|Q))|,)|P)|S)))|((O|(D|F))| ))), probability: 0.9932341001353181*

So, we created binary tree of Huffman. After that, we can get values of every letter or symbol. Take Dictionary *{'char': 'A', 'code': '1001'} as an example. “*Char” is the symbol, which is “A” in this case. Also, “Code” is bit code of given symbol, bit code of “A” is 1001.

Code value depends on symbol probability, if probability is low, the length of bit code is long as well. Or if probability is high, the length of bit code is short. After getting dictionary we encrypted Text.

Output: Figure - 3, Figure - 4, Figure - 5

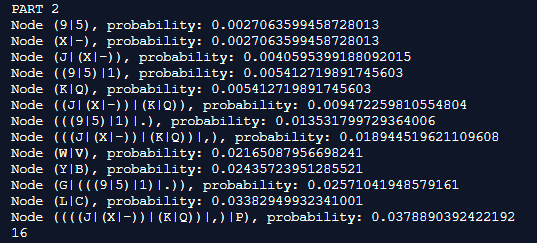


Figure - 3. Binary tree of Huffman

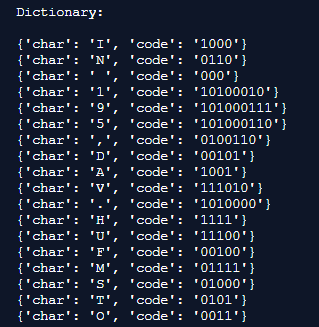


Figure - 4. Dictionary

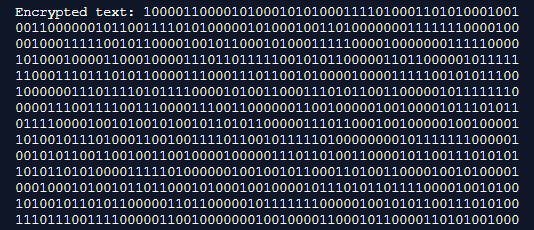


Figure - 5. Encrypted text

## Part 3

Goal: Decode back into text the result binary sequence after Part2 (Shannon-Fano or Huffman code).

In this part we decrypted encrypted Text(from previous one) with the support of dictionary list and decrypt function. The result is original text. We did it to check out did it somehow changed or not.

Output: Figure - 6

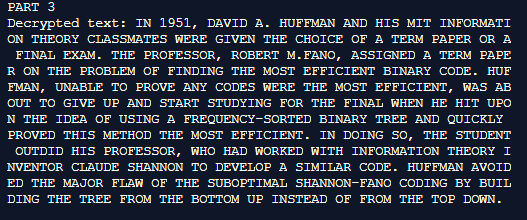


Figure - 6. Decrypted text

## Part 4

Goal: Encode the result binary sequence after Part2 (Shannon-Fano or Huffman code) with Hamming code.

We took the sequence after Part 2. And, check(with “%”) can we divide it into groups which consist of 4 bits to avoid the error. If it divides, there is no any problem. Yet if does not, we adding extra one or several “0” till whole sting will be dividable into 4. The next step, we used Hamming code (7, 4) (first res4 value equals to XOR of first 3 values of group, the second value equals to XOR of last 3 values of group and the third value equals to XOR of first 2 and last value of group)

*for i in range(len(res)):*

*res4 = res4 + res[i]  
if (i+1) % 4 == 0:*

*res4 = res4 + str((int(res[i-3]) ^ int(res[i-2]) ^ int(res[i-1])))  
 res4 = res4 + str((int(res[i-2]) ^ int(res[i-1]) ^ int(res[i])))  
 res4 = res4 + str((int(res[i-3]) ^ int(res[i-2]) ^ int(res[i])))*

and added 3 bits one by one. Then, printed the result.

Output: Figure - 7, Figure - 8

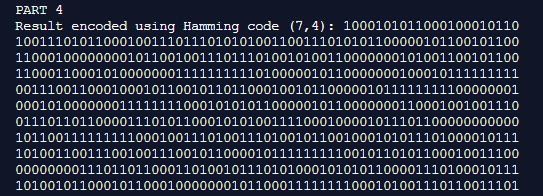


Figure - 7. Result of encoded using Hamming code

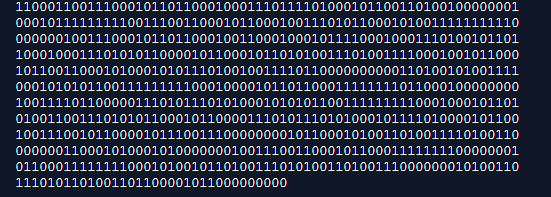


Figure - 8. Other part of Figure - 7

## Part 5

Goal: Add errors to the result binary sequence after Part4 (Hamming code) with a certain interval.

Part 5 takes values after Part 4. Then, with support of Split\_text function we are splitting string into 7 each group. After dividing, we made program that chooses one random number from that 7 and changes to another value(we created function of replacing by our own and named it “inversebit”). Then, we got “splitted” result with errors in every splitted part.

Output:

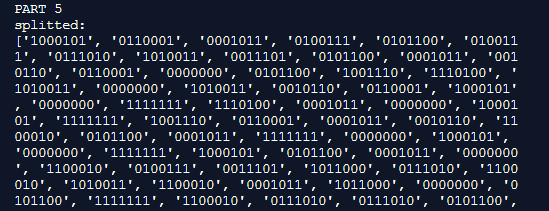


Figure - 9. Splitted binary sequence

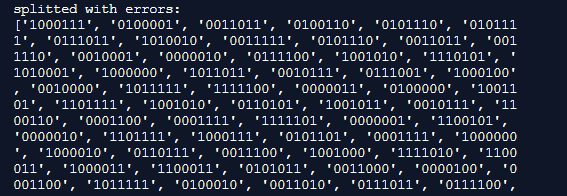


Figure - 10. Splitted binary sequence with errors

## Part 6

Goal: Decode Hamming code and fix errors.   
 We take results of part 5 then split it with split\_bit function. And, realized Hamming(7,4) Encoding. Then, we found out error position and replaced(fixed) it to another value(0 to 1 or 1 to zero). After correction, we chose the first 4 elements from 7 to leave 3 extra bit which we added in part 4. Next, we converted massive into string with function join(joins every element of massive one by one and that creates string). Last step is decryption. Finally, we see that the result did not change from the initial one.

Output:

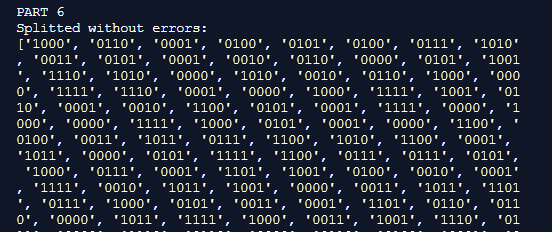


Figure - 11. Splitted binary sequence without errors and 3 extra binary values

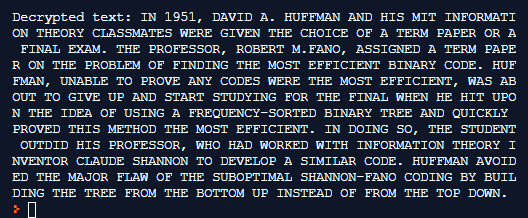


Figure - 12. Decrypted text, which we splitted before

**CONCLUSION**

The essence of this project was to ensure optimal encoding of information in a bit format with the ability to decode and transfer information with minimal changes, errors. Taking everything into account, our team has done a great job completing each part of the project step by step. Doing each part of the project, we learned something new and put it into practice. We also saw how the algorithms of Shannon Fano and Huffman work. It is also worth mentioning that we encoded the resulting binary sequence using Hamming code, thus understanding how it works. After that, we specifically made mistakes in the coding of the Hamming code in order to search for these errors and correct them for decoding. Also this project teaches you to work in a team that we definitely need in the future.

**REFERENCES**

1. Lectures from dl.iitu.kz
2. [http://www.sns.ias.edu/pitp2/2012files/Hamming\_CHs1-3.pdf]
3. [https://en.wikipedia.org/wiki/Hamming\_code]
4. [https://www.sciencedirect.com/topics/engineering/huffman-code]
5. [http://ericmoorhouse.org/handouts/encoding.pdf]
6. [https://www.slideshare.net/HetangChauhan/hamming-encoder-and-decoder]