BBM203: Software Laboratory 2020 Fall Assignment-4 Report

Mert Tazeoğlu, 21946606

January 1, 2021

Table Of Contents:

Part-1: Encoding Algorithm Explanation

Step-1: Read input file. **(ReadLineByLine Method)**
Step-2: For every character in input file: **(calcFreq Method)**
Step-2.1: Lowercase character. **(main Method)**
Step-2.2: If character exists, add +1 to its frequency. **(main Method)**
Step-2.3: Else create a frequency variable for it. **(main Method)**
Step-3: Create a leaf node for each item and it to the priority queue. **(HuffmanCodes Method)**
Step-4: While there are more than one nodes in queue: **(HuffmanCodes Method)**
Step-4.1: Remove the two nodes of highest priority from the queue. **(HuffmanCodes Method)**
Step-4.2: Create a new node with these two nodes as children and with their frequency equal of the sum of the two nodes' frequency. **(HuffmanCodes Method)**
Step-4.3: Add the new node to the queue. **(storeCodes Method)**
Step-5: Remaining node is root node and Huffman tree is completed. **(storeCodes Method)**
Step-6: Create a new string. For every character in input file: add characters' encoded values to string. **(encode Method)**
Step-7: Print result to the console. **(encode Method)**
Step-8: Write and save input files components in "**backup.txt**" for next steps.

This algorithms functions' names in my code are written in parantheses of steps. An example of output of this algoritm is at part-4. This is classic Huffman Encoding Algorithm. This algorithm encodes strings with binary values. Algorithm calculates these values using with frequencies of components of input file. Then with using priority queue, it saves these values in Huffman Tree data structure. In the last step, we replace input files characters with their encoded values and we print that encoded string.

Part-2: Decoding Algorithm Explanation

Step-1: Read input file. **(ReadLineByLine Method)**
Step-2: For every character in input file lowercase character.
Step-3: Read "**backup.txt**" file. **(ReadLineByLine Method)**
Step-4: Recreate Huffman tree and other components. **(HuffmanCodes and storeCodes Methods)**
Step-5: For every character in input file: **(decode_file Method)**
Step-5.1: Traverse in Huffman Tree. **(decode_file Method)**
Step-5.2: Return final value. **(decode_file Method)**
Step-6: Print final value in the console. **(main Method)**

This algorithms functions' names in my code are written in parantheses of steps. An example of output of this algoritm is at part-4. This is classic Huffman Decoding Algorithm. This algorithm decodes strings with using "backup.txt" which we generated at encoding step. Decoding is reverse function of encoding function. So we can say that, these steps are logically reverse of encoding algorithm at part-1 of report. For encoding, we traverse at Huffman Tree and for every character in input.txt we replace input files characters with their decoded values and we print that encoded string.

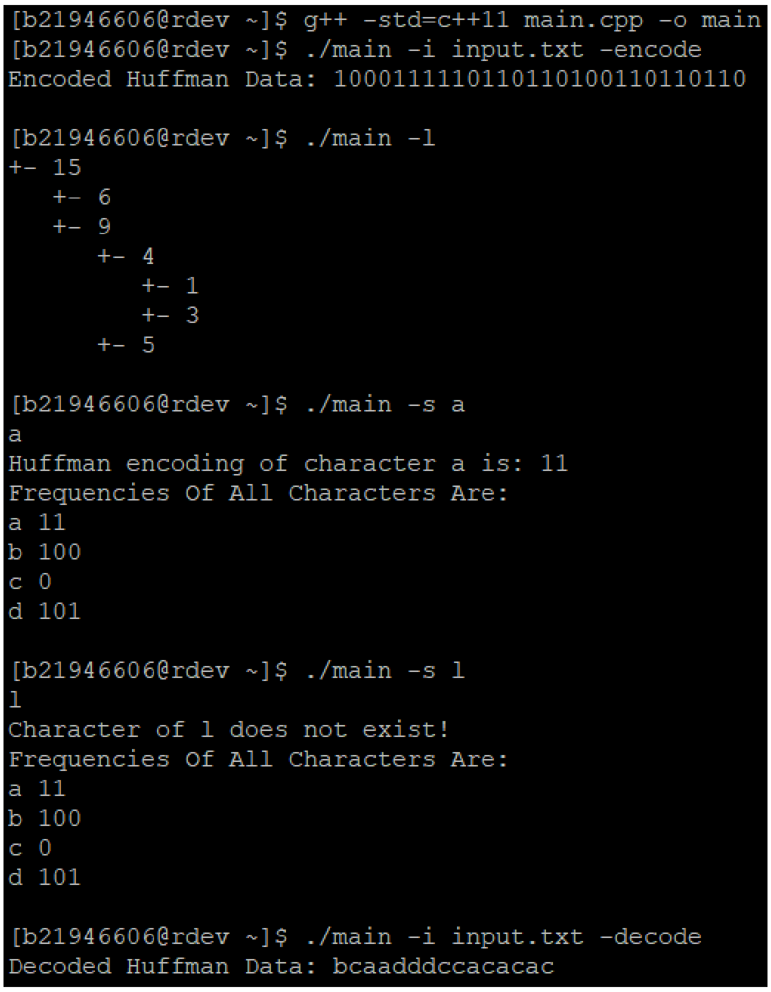
Part-3: List Tree Algorithm Explanation

Step-1: Set height as 0. **(PrintHuffmanTree Method)**
Step-2: If node doesn't exit return.
Step-3.1: Else print " " for 3x times of height.
Step-3.2: Then print value of of given node. (Value is an integer)
Step-4: Call same function for left and right nodes and with +1 height.

Notes: This algorithm works only with 1 function. This function is an recursive function. This function prints values every nodes and it's also careful for identation for realistic imagination. An example of output of this algoritm is at part-4. Working principle of this algorithm is simple: We start at root node and traverse all nodes with going deeper at both of left and right sides. In every step we print values with being careful of tree levels. (For example 3x space in 1st level, 6x space in 2nd level etc.)

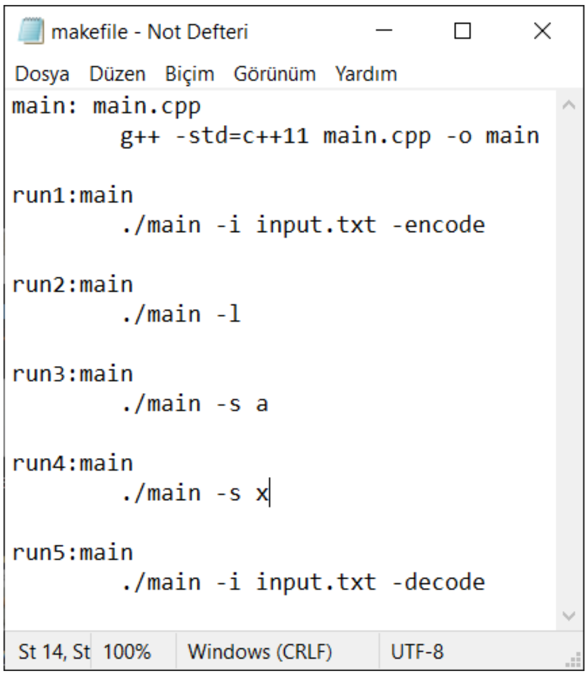
Part-4: How To Run Program?

As we can see at the image below, my program works on DEV server.



```
[b21946606@rdev ~]$ g++ -std=c++11 main.cpp -o main
[b21946606@rdev ~]$ ./main -i input.txt -encode
Encoded Huffman Data: 1000111110110110100110110110

[b21946606@rdev ~]$ ./main -l
+- 15
   +- 6
   +- 9
      +- 4
         +- 1
         +- 3
      +- 5

[b21946606@rdev ~]$ ./main -s a
a
Huffman encoding of character a is: 11
Frequencies Of All Characters Are:
a 11
b 100
c 0
d 101

[b21946606@rdev ~]$ ./main -s l
l
Character of l does not exist!
Frequencies Of All Characters Are:
a 11
b 100
c 0
d 101

[b21946606@rdev ~]$ ./main -i input.txt -decode
Decoded Huffman Data: bcaadddccacacac
```

I also send a makefile as stated in below:


makefile - Not Defteri
Dosya Düzen Biçim Görünüm Yardım
St 14, St 100% Windows (CRLF) UTF-8

```
main: main.cpp
	g++ -std=c++11 main.cpp -o main

run1:main
	./main -i input.txt -encode

run2:main
	./main -l

run3:main
	./main -s a

run4:main
	./main -s x

run5:main
	./main -i input.txt -decode
```

But i'm not sure makefile working properly. I tried to use makefiles in Windows OS, but i wasn't able to do it. Program works on dev server properly, but i dont know my makefile is correct or wrong.

Part-5: References

1. Assignment PDF File (For Report and Codes)
https://d1b10bmlvqabco.cloudfront.net/paste/k4seb9z1p641z/f38a117d1e70fd9e12fa7ac6d860abfa3ce5bfee38d26ccc05e6d5cd590bcdee/assignment_4.pdf

2. CodeSpeedy Huffman Page (For Report and Some Parts Of Codes)
https://www.codespeedy.com/huffman-decoding-in-cpp/