BLG 223E - Data Structures
Assignment #2

Due: December, 16th 23:59

Introduction

For this assignment, you are expected to implement Huffman algorithm (Loseless data comprassion algorithm) to create an encryption system. For any issues regarding the assignment, you can ask your questions on Ninova Message Board. Before writing your question please check whether your question has been asked. You can also contact T.A. Barış Bilen (**bilenb20@itu.edu.tr**).

Implementation Notes

The implementation details below are included in the grading:

1. Please follow a consistent coding style (indentation, variable names, etc.) with comments otherwise you will get **minus** points.
2. You are not allowed to use STL containers.
3. You may implement functions of your own if they are needed.
4. Make sure that there is no memory leak in your code otherwise you will get **minus** points.
5. Since your code will be tested automatically, make sure that your outputs match with sample scenario for given inputs. You will be provided with a calico file to check your assignment output.

Submission Notes

1. Your program should compile and run on Linux environment. You can (and should) code and test your program on Docker's Virtual Environment. Do not use any pre-compiled header files or STL commands. Be sure that you have submitted all of your files.
2. Submissions are made through **only** the Ninova system and have a strict deadline. Assignments submitted after the deadline will not be accepted.
3. You can discuss general ideas but this is not a group assignment do not work together. Getting involved in any kind of cheating is subject to disciplinary actions. Your assignment should not include any copy-paste material (from the Internet or from someone else's paper/thesis/project).

Implementation Details

1 Structs Header File

1.1 Token

This class is for you to hold the symbols (characters) and their corresponding values. Do not add or change any variable in this class. (e.g.: Symbol: C, Value: 2) (Look for step 2 in figure 1)

1.2 Node

This is your node class. Use this class both in priority queue and tree to create the structures. Do not add or change any variable in this class.

1.3 Priority Queue

Use priority queue to arrange the tokens order. Lower node values has higher priorities. This means the node with the lowest value should be the head of the queue. Implement the given methods down below. If you find it necessary, you can implement functions of your own.

Methods:

1. **PriorityQueue:** Initializes priority queue structure.
2. **enque:** En-queue the given node depending on its token value. If the token values of the nodes are equal then en-queue the new node after the one inside the queue (e.g.: Lets say token value is **4** and queue is ordered in 1-3-4-5. En-queue the 4 between 4 and 5. Your final queue should look like 1-3-4-**4**-5.).
3. **dequeue:** De-queue the head node and delete the node from the queue.

1.4 Tree

Use tree structure to create the Huffman tree. Implement the given methods down below. If you find it necessary, you can implement functions of your own.

Methods:

1. **Tree:** Initializes tree structure.
2. **~Tree:** Deletes the tree nodes.
3. **mergeNodes:** Merges two nodes token symbols and values in a tree structure way. Add token symbols and values of the nodes and assign these nodes left and right side of the new node. (e.g.: Node1(symbol:a, val:1), Node2(symbol:b, val:1) **after merge** Node(symbol:ab, val:2) Node->left = Node1, Node->right = Node2) (Look for step 2 in figure 1)

2 Huffman Header File

Huffman algorithm is used for lossless data compression. Algorithm creates a tree structure depending on the symbols (characters) repetitions (lets say there are 5 "A"'s in the key so value of a is 5) and assign a binary value for every unique symbol. By doing so symbols are represented with fewer bits (e.g.: a char is represented with 8 bits but after Huffman it might be represented with 3-4 bits).

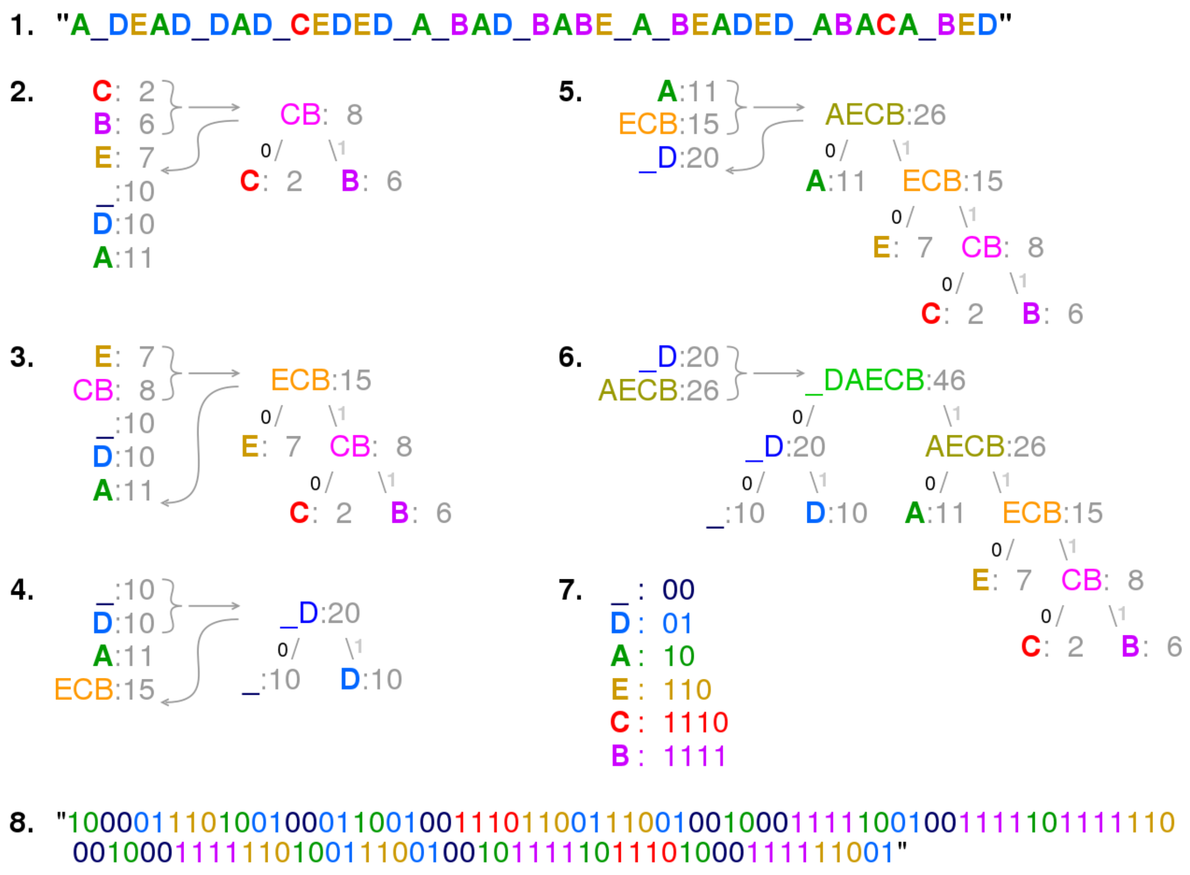

Figure 1: Example of how the Huffman algorithm works. (See **Huffman Coding, Wikipedia**)

2.1 Huffman

You have been provided with a key reader and a key sorter functions. Implement the given methods down below. If you find it necessary, you can implement functions of your own.

Methods:

1. **findFrequency:** Finds the frequency of the symbols (characters).
2. **createHuffmanTree:** Creates huffman tree.
3. **getTokenBinary:** Finds the binary value of a given symbol.
4. **encodePassword:** Encodes the given password.
5. **decodePassword:** Decodes the given encoded password.
6. **decodeToken:** Finds the symbol of the given binary.

Notice: You are already given a skeleton code. Please examine it before writing your own codes. Skeleton code should includes all necessary classes and variables you need. You are expected to implement methods of these classes. If you find it necessary, you can implement functions of your own.

3 Encryption Logic

What you will be doing is to implement Huffman algorithm in a way to create an encryption system. Using a key (e.g.: Step 1 in figure 1) you will create a Huffman tree. Then by using this tree you will be encoding and decoding passwords. An encoded password has 2 contents: binary and value. Binary holds the binary output of the password and value holds the depths of the symbols inside the tree. Lets say your password is "ACE". To encode this password you need to find the binary values and depths of these symbols (a,c,e). By looking from the above figure (Look for step 7 in figure 1) you can find out their binary outputs and depths.
For A -> Depth: 2 and Binary: 10
For C -> Depth: 4 and Binary: 1110
For E -> Depth: 3 and Binary: 110
With this knowledge you can create the encoded password. First content of the password (binary) will be 101110110 and second content of the password (value) is 243.
Binary Password: 101110110
Value Password: 243

And to decode this password, what you need to do is to take the value elements (2,4,3) and split the binary password according to those elements (10(2), 1110(4), 110(3)). Then you will find the symbol they represent by following the binary output inside the Huffman tree.

Notice: Did anything catch your attention in the keys? If so, write it down inside the Question.txt file and you might get a bonus point. Be aware though i am looking for a specific answer. :)

4 How to compile, run and test your code

If you want to compile and run the provided code on terminal, you can use these commands:

```
g++ -Wall -Werror src/main.cpp src/structs.cpp src/huffman.cpp -I include -o main
./main key1.txt
```

You can (and should) run the calico file on terminal to check your assignment with the command:

```
calico hw2_test.yaml --debug
```

5 Submission

Zip your project folder and submit the zip file to Ninova. Before submitting please make sure you are passing all cases in calico file. Calico file you received is not the grading file so taking 100 does not corresponds to full grade for your assignment. Passing all cases will show you that your output is compatible with the grading file. Otherwise you might not receive a full grade from the assignment.
Your zip file should look like this:

```
/assignment2
    /bin
        main
    /include
        structs.h
        huffman.h
    /src
        structs.cpp
        huffman.cpp
        main.cpp
    hw2_test.yaml
    key1.txt
    key2.txt
    Question.txt
```

Notice: Do not change any folder names or file names or file locations. Check your project folder and make sure everything is in the same order as provided with skeleton folder.