Hacettepe University

Computer Engineering Department

BBM203 SOFTWARE LAB-I 2020 FALL

ASSIGNMENT 4

Name and Surname: Yunus Emre YAZICI
Identity Number: 21827981
Course: BBM203
Experiment Subject: C++ Trees
Due Date: 01.01.2021
Advisor: Yunus Can Bilge
E-Mail: b21827981@cs.hacettepe.edu.tr

Software Design Notes

1- Problem Definition

In this experiment, we are expected to build an Huffman encoding and decoding system. Huffman algorithm is a powerful algorithm that is used for compressing textual data to make file occupy less space with respect to number of bytes. Intuitively, the algorithm works based on a frequency sorted binary tree to encode the input. In order to do it, first of all, we need to build a tree with respect to given text's characters and their frequencies. After that , we should be able to find the character's encoding form from the tree and encode the whole text. For decoding, we should have a encoded string and decode it with respect to previous tree, which is used for encoding the text. At last, we should operate given commands such as -l (display tree) and -s char (encoding form of a particular character).

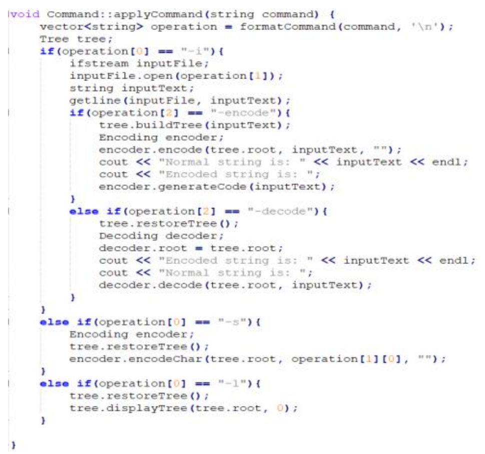
2- Solution

First of all, the codes takes the commands as arguments. For encoding, it reads the text and calculates the characters' frequency table. After calculation, it writes the frequency table to a txt file for next operations such as decoding, displaying tree, encoding a particular character. After writing, it creates a tree with respect to characters frequencies. After that, it operates encoding operation and creates encoded text and prints it. For decoding, first of all, it reads frequency table's txt file and creates the previous tree again. Then, it takes encoded text and inspects it part by part, finds the characters from tree. After inspection, it prints the word character by character. For printing tree and encoding a character commands, again it re-creates the tree and calls the functions needed.

3- Algorithms and Code Design

Command:

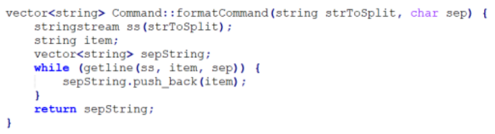
- It operates what to do. Takes arguments and directs the program to the wanted functions.



```
void Command::applyCommand(string command) {
    vector<string> operation = formatCommand(command, '\n');
    Tree tree;
    if(operation[0] == "-i"){
        ifstream inputFile;
        inputFile.open(operation[1]);
        string inputText;
        getline(inputFile, inputText);
        if(operation[2] == "-encode"){
            tree.buildTree(inputText);
            Encoding encoder;
            encoder.encode(tree.root, inputText, "");
            cout << "Normal string is: " << inputText << endl;
            cout << "Encoded string is: ";
            encoder.generateCode(inputText);
        }
        else if(operation[2] == "-decode"){
            tree.restoreTree();
            Decoding decoder;
            decoder.root = tree.root;
            cout << "Encoded string is: " << inputText << endl;
            cout << "Normal string is: ";
            decoder.decode(tree.root, inputText);
        }
    }
    else if(operation[0] == "-s"){
        Encoding encoder;
        tree.restoreTree();
        encoder.encodeChar(tree.root, operation[1][0], "");
    }
    else if(operation[0] == "-l"){
        tree.restoreTree();
        tree.displayTree(tree.root, 0);
    }

}
```

Takes the given arguments as a whole string and use formatCommand() function to give them proper format. After that it reads the arguments step by step. If the arguments starts with '-i', then it reads the input file and sends it to the encoding or decoding part. If it starts with '-s', it sends the particular character to the encodeChar() function to find the encoded form of it. If it starts with '-l', then it calls printing function of the tree. For decoding, encoding a char and displaying tree, it restores the previous tree from frequency table.

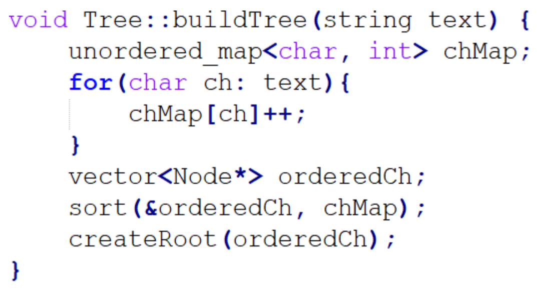


```
vector<string> Command::formatCommand(string strToSplit, char sep) {
    stringstream ss(strToSplit);
    string item;
    vector<string> sepString;
    while (getline(ss, item, sep)) {
        sepString.push_back(item);
    }
    return sepString;
}
```

Separates the given string with respect to given separator and returns it as a vector.

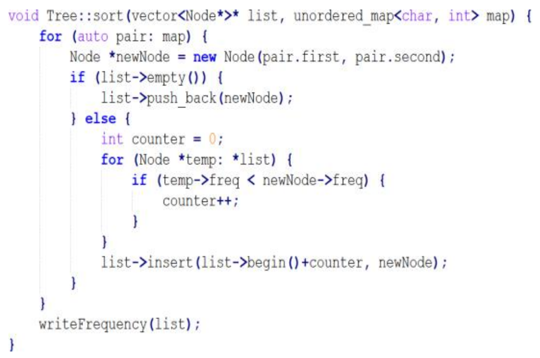
Encoding:

- First, the code reads the input file and takes the text from it. After that, it creates a tree.



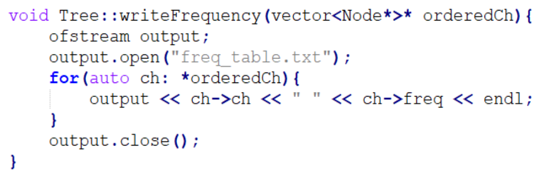
```
void Tree::buildTree(string text) {
    unordered_map<char, int> chMap;
    for(char ch: text){
        chMap[ch]++;
    }
    vector<Node*> orderedCh;
    sort(&orderedCh, chMap);
    createRoot(orderedCh);
}
```

In order to create a tree, it takes the characters from text one by one and assigns them into a map. The key of the map is the characters and the value is their frequencies. After the loop is done, it creates a local vector and calls sorting function. After sorting it calls the createRoot() function.



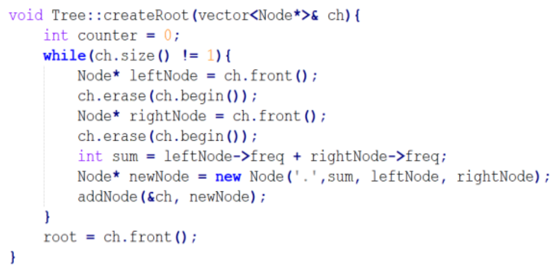
```
void Tree::sort(vector<Node*>* list, unordered_map<char, int> map) {
    for (auto pair: map) {
        Node *newNode = new Node(pair.first, pair.second);
        if (list->empty()) {
            list->push_back(newNode);
        } else {
            int counter = 0;
            for (Node *temp: *list) {
                if (temp->freq < newNode->freq) {
                    counter++;
                }
            }
            list->insert(list->begin()+counter, newNode);
        }
    }
    writeFrequency(list);
}
```

The sorting algorithm, takes an empty vector and character's map. For every character in the map, it creates a node and pushes them into the vector with respect to their frequencies. The frequencies are ordered lower to higher. After sorting operation, it writes the frequency table to a txt file. For next usages, such as decoding, displaying tree and encoding particular char.



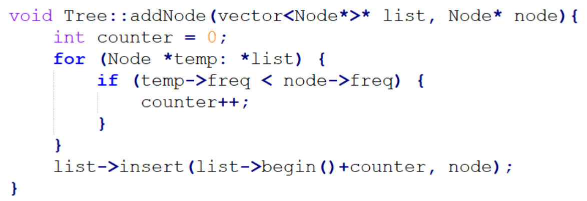
```
void Tree::writeFrequency(vector<Node*>* orderedCh){
    ofstream output;
    output.open("freq_table.txt");
    for(auto ch: *orderedCh){
        output << ch->ch << " " << ch->freq << endl;
    }
    output.close();
}
```

After sorting, it writes freq_table.txt file with proper format for re-creation of the tree. The format is [char][space][frequency].



```
void Tree::createRoot(vector<Node*>& ch){
    int counter = 0;
    while(ch.size() != 1){
        Node* leftNode = ch.front();
        ch.erase(ch.begin());
        Node* rightNode = ch.front();
        ch.erase(ch.begin());
        int sum = leftNode->freq + rightNode->freq;
        Node* newNode = new Node('.',sum, leftNode, rightNode);
        addNode(&ch, newNode);
    }
    root = ch.front();
}
```

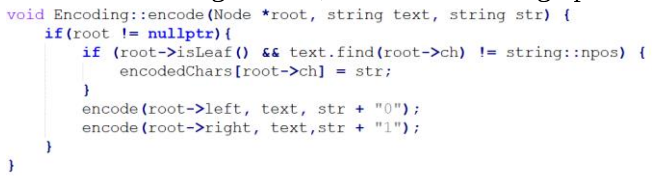
The root creation functions, takes ordered vector and creates a new node with its elements. It takes elements two by two and creates a new node, where these two nodes become leftNode and rightNode of the parent. Then, it adds new node to vector back with addNode() function. This operation continues, until the size of the vector is 1 and there is not any second node to create another one. After that, it assigns the one node to the root.



```
void Tree::addNode(vector<Node*>* list, Node* node){
    int counter = 0;
    for (Node *temp: *list) {
        if (temp->freq < node->freq) {
            counter++;
        }
    }
    list->insert(list->begin()+counter, node);
}
```

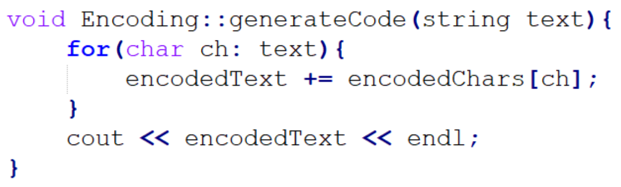
It takes ordered vector and the node that should be added in the vector. Then, counts how many frequency there are that are bigger than node's frequency. After that, it adds the new node to the vector without changing the order.

- After creating the tree, the main encoding operation starts



```
void Encoding::encode(Node *root, string text, string str) {
    if(root != nullptr){
        if (root->isLeaf() && text.find(root->ch) != string::npos) {
            encodedChars[root->ch] = str;
        }
        encode(root->left, text, str + "0");
        encode(root->right, text,str + "1");
    }
}
```

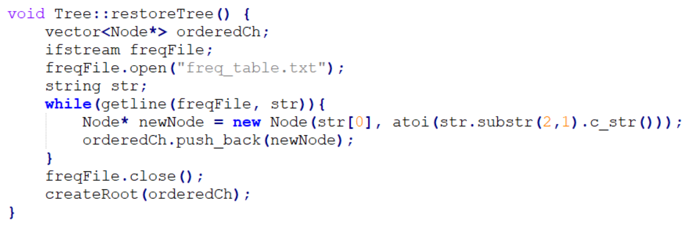
If the given command includes an input file with a whole text, then, this function is called. This function starts with the root and searches for leaf which is a character from the text. Firstly, it goes to the leftist side and looks for the leaf, if there is one, then it assigns the character as key, and its encoded form as value. After that it returns leaf's parent node and looks for the right-hand side. If there is a leaf, again assigns it. Otherwise, it returns current node's parent as well. It continues recursively until the whole tree and assigning operation is finished. Every time, the code goes to the left, it adds 0 to the encoded str, and every time it goes to the right, it adds 1 to the encoded str. For instance, let's say the code goes 2 left and 1 right orderly and finds a leaf, then the encoding is 001.



```
void Encoding::generateCode(string text){
    for(char ch: text){
        encodedText += encodedChars[ch];
    }
    cout << encodedText << endl;
}
```

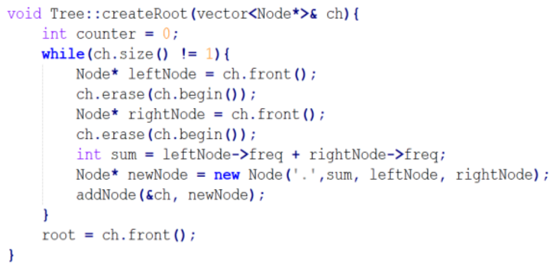
To print a whole text's encoded form, this function is called. It takes every character from the text and adds it to an initially empty string. After the loop is done, it prints the encoded form

- **An important step for Decoding, Displaying Tree and Encoding a character:**
  Restoring tree from previous frequency table:



```
void Tree::restoreTree() {
    vector<Node*> orderedCh;
    ifstream freqFile;
    freqFile.open("freq_table.txt");
    string str;
    while(getline(freqFile, str)){
        Node* newNode = new Node(str[0], atoi(str.substr(2,1).c_str()));
        orderedCh.push_back(newNode);
    }
    freqFile.close();
    createRoot(orderedCh);
}
```

It uses a local vector and reads previous freq_table.txt line by line. In every line there is a character and its frequency [char][space][frequency]. The function creates a new node with given information and pushes it to the vector. After the loop finishes it calls createRoot() function.

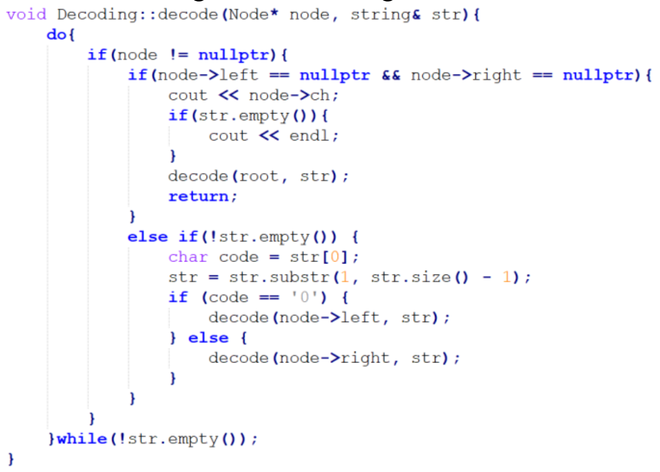


```
void Tree::createRoot(vector<Node*>& ch){
    int counter = 0;
    while(ch.size() != 1){
        Node* leftNode = ch.front();
        ch.erase(ch.begin());
        Node* rightNode = ch.front();
        ch.erase(ch.begin());
        int sum = leftNode->freq + rightNode->freq;
        Node* newNode = new Node('.',sum, leftNode, rightNode);
        addNode(&ch, newNode);
    }
    root = ch.front();
}
```

The root creation functions, takes ordered vector and creates a new node with its elements. It takes elements two by two and creates a new node, where these two nodes become leftNode and rightNode of the parent. Then, it adds new node to vector back with addNode() function. This operation continues, until the size of the vector is 1 and there is not any second node to create another one. After that, it assigns the one node to the root.

Decoding:

- First, it reads frequency table's txt file and creates previous tree with it. Then, it reads the input file and takes encoded string from it. To decode the encoded string, before the operation, tree's root is assigned as decoding class' root node, for next usages in the function.

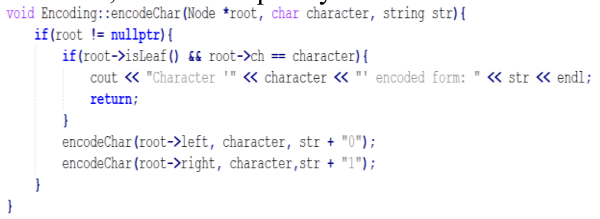


```cpp
void Decoding::decode(Node* node, string& str){
    do{
        if(node != nullptr){
            if(node->left == nullptr && node->right == nullptr){
                cout << node->ch;
                if(str.empty()){
                    cout << endl;
                }
                decode(root, str);
                return;
            }
            else if(!str.empty()) {
                char code = str[0];
                str = str.substr(1, str.size() - 1);
                if (code == '0') {
                    decode(node->left, str);
                } else {
                    decode(node->right, str);
                }
            }
        }
    }while(!str.empty());
}
```

To decode the text, it takes current node, and the encoded string as parameter. Searches the tree for given encoded string character by character. If the current character from tree is 0, then the function goes to the left side of the tree. If it is 1, then the function goes to the right side of the tree until it finds a leaf. Every time it takes a character from string, it deletes it from the string as well. In every step the function also checks if the node is a leaf, if it is prints the character of the node. After finding a leaf, it starts from tree's root again and searches for the next character. The operation continues until the encoded string is empty and there is nothing to print.

Encoding for a particular character:

- First, it reads frequency table's txt file and creates previous tree with it.

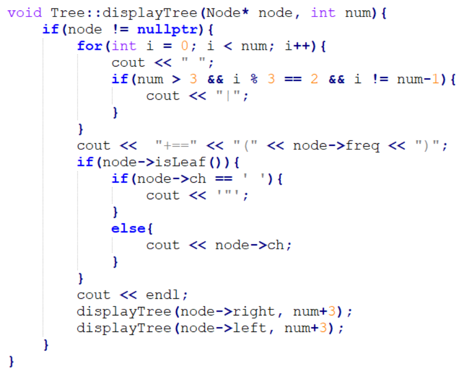


```cpp
void Encoding::encodeChar(Node *root, char character, string str){
    if(root != nullptr){
        if(root->isLeaf() && root->ch == character){
            cout << "Character '" << character << "' encoded form: " << str << endl;
            return;
        }
        encodeChar(root->left, character, str + "0");
        encodeChar(root->right, character,str + "1");
    }
}
```

This function searches for a particular leaf that has the wanted character. It similarly repeats the steps from encode() function. The difference is that, when it finds the character it prints its encoded form and returns, instead of storing its data in a map.

Listing a tree

- To print whole tree, the previous tree should be created again. So, it reads frequency table and re-creates the tree. Then calls this function.



```cpp
void Tree::displayTree(Node* node, int num){
    if(node != nullptr){
        for(int i = 0; i < num; i++){
            cout << " ";
            if(num > 3 && i % 3 == 2 && i != num-1){
                cout << "|";
            }
        }
        cout <<  "+==" << "(" << node->freq << ")";
        if(node->isLeaf()){
            if(node->ch == ' '){
                cout << '"';
            }
            else{
                cout << node->ch;
            }
        }
        cout << endl;
        displayTree(node->right, num+3);
        displayTree(node->left, num+3);
    }
}
```

The function starts from the rightest side of the tree, and prints it step by step. When there are 2 leaf or parents at the same level, the upper side is always right side of the tree and the down side is always left side of the tree. The | indicator shows belongings, if there is a '|' symbol under '+==' until the another part, the elements between belongs to the upper '+=='. If a node is not a leaf, it just prints its frequency. If it is a leaf, it prints its frequency and its character.

- The format of displaying a tree:

```
+== root
   +== right branch
   |   +== right sibling
   |   |   +== right grandchild
   |   |   +== left grandchild
   |   +== left sibling
   |   |   +== right grandchild
   |   |   +== left grandchild
   |   +== left sibling
   |   |   +== right grandchild
   |   |   +== left grandchild
   +== left branch
   |   +== right sibling
   |   +== left sibling
```

4- *Program Compile and Command Line Arguments*

To compile the code the user can either one of these commands

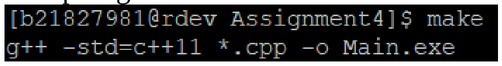
- make
- g++ -std=c++11 *.cpp -o Main.exe

To run the program, these commands can be used:

- ./Main.exe -i input_file.txt -encode (encodes the given input file)
- ./Main.exe -i input_file.txt -decode (decodes the given input file)
- ./Main.exe -s character (encodes a particular character)
- ./Main.exe -l (display whole tree)

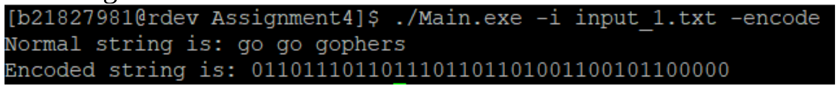
5- *Example usage and outputs*

- Compiling the code



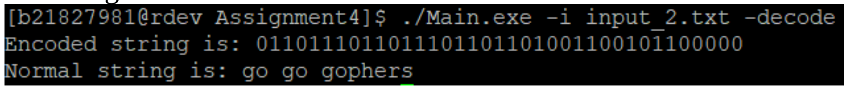
```
[b21827981@rdev Assignment4]$ make
g++ -std=c++11 *.cpp -o Main.exe
```

- Encoding



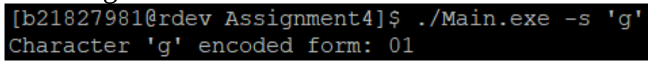
```
[b21827981@rdev Assignment4]$ ./Main.exe -i input_1.txt -encode
Normal string is: go go gophers
Encoded string is: 01101110110111011011010011001011100000
```

- Decoding



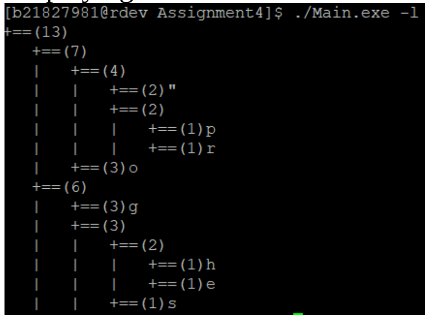
```
[b21827981@rdev Assignment4]$ ./Main.exe -i input_2.txt -decode
Encoded string is: 01101110110111011011010011001011100000
Normal string is: go go gophers
```

- Encoding a character



```
[b21827981@rdev Assignment4]$ ./Main.exe -s 'g'
Character 'g' encoded form: 01
```

- Displaying the tree



```
[b21827981@rdev Assignment4]$ ./Main.exe -l
+==(13)
   +==(7)
   |   +==(4)
   |   |   +==(2)"
   |   |   +==(2)
   |   |   |   +==(1)p
   |   |   |   +==(1)r
   |   +==(3)o
   +==(6)
   |   +==(3)g
   |   +==(3)
   |   |   +==(2)
   |   |   |   +==(1)h
   |   |   |   +==(1)e
   |   |   +==(1)s
```

Important Note:

- I have right to use 2 days extension and I want to use it. So, I kindly request my assignment to be evaluated over 100.