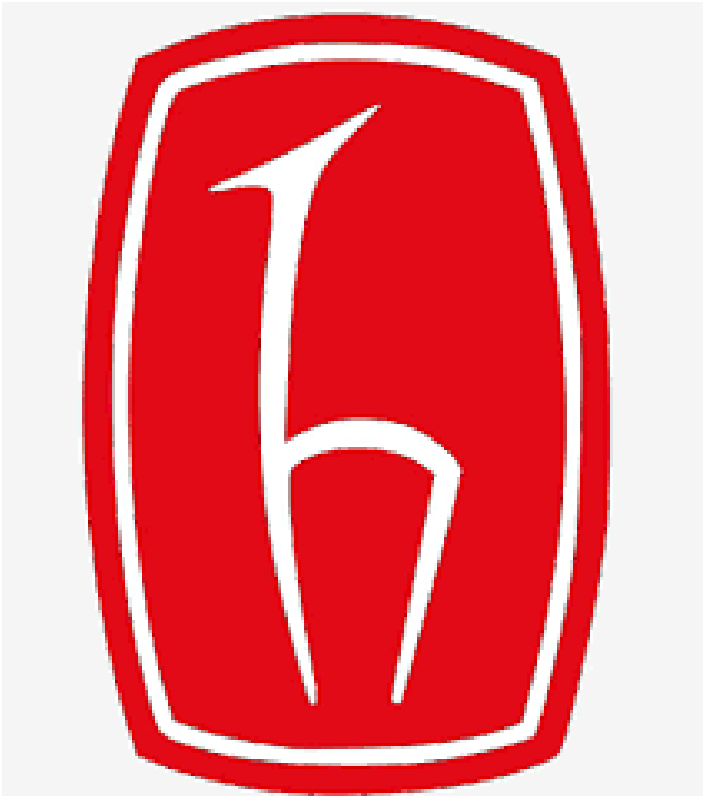

HACETTEPE UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT
DATA STRUCTURES LAB
ASSİGNMENT 4

Mustafa KOLLU - 21627485

AİM:

In this assignment, I will implement a file compression algorithm based on Huffman coding with trees.

```
if(arg1.compare( s: "-i")==0){
    string line;
    string text;
    string arg3= argv[3];
    if(arg3.compare( s: "-encode")==0){
        ofstream fout( s: "temp.txt");
        ifstream readFile( s: argv[2]);
        if (readFile.is_open()) {
            while (getline( &: readFile,  &: line)) {
                text= line;
            }
        }
        int indis=0;
        while(text[indis])
        {
            int deger = text[indis];
            if(deger>64 && deger<91)
                text[indis]+=32;
            indis++;
        }
        work.createHuffmanTree(text, &: fout);
        fout.close();
    }
    if(arg3.compare( s: "-decode")==0){
        ifstream readFile2( s: "temp.txt");
        if (readFile2.is_open()) {
            while (getline( &: readFile2,  &: line)) {
                text= line;
                if(text.substr( pos: 0, n: 13).compare( s: "DecodedString")==0){
                    for (int i = 14; i <text.length() ; i++) {
                        cout<<text[i];
                    }
                }
            }
        }
        cout<<endl;
    }
}
```

```
if(arg1.compare( s: "-s")==0){
    string line;
    string text;
    string arg2=argv[2];
    ifstream readFile2( s: "temp.txt");
    if (readFile2.is_open()) {
        while (getline( &: readFile2,  &: line)) {
            text= line;
            if(text.substr( pos: 0, n: 6).compare( s: "Codes-")==0){
                if(text.substr( pos: 6, n: 1).compare(arg2)==0){
                    for (int j=8;j<text.length();j++){
                        cout<<text[j];
                    }
                    cout<<endl;
                }
            }
        }
    }
if(arg1.compare( s: "-l")==0){
    string line;
    string text;
    ifstream readFile2( s: "temp.txt");
    if (readFile2.is_open()) {
        while (getline( &: readFile2,  &: line)) {
            text= line;
            if(text.substr( pos: 0, n: 4).compare( s: "tree")==0){
                for (int i = 4; i <text.length() ; i++) {
                    cout<<text[i];
                }
                cout<<endl;
            }
        }
    }
}
```

This is my main function reads txt file and print what function needs in Main.cpp.

The rest of the functions are all in Program.h.

1. Encoding Algorithm

```
void createHuffmanTree(string text,ofstream& fout){

    unordered_map<char, int> frequency;

    for (char letter: text) {
        frequency[letter]++;
    }

    priority_queue<Node*, vector<Node*>, order> priorityQueue;

    for (auto branch: frequency) {
        priorityQueue.push( x: getNode(branch.first, branch.second, left: nullptr, right: nullptr));
    }

    while (priorityQueue.size() != 1){
        Node *left = priorityQueue.top(); priorityQueue.pop();
        Node *right = priorityQueue.top();  priorityQueue.pop();
        int total = left->frequency + right->frequency;
        priorityQueue.push( x: getNode( letter: '\0', total, left, right));
    }

    Node* root = priorityQueue.top();

    unordered_map<char, string> huffman;
    encode(root, encoding: "", &: huffman);
```

1
2
3
4
5
6

Photo-1

```
void encode(Node* root, string encoding,unordered_map<char, string> &huffman){
    if (root == nullptr){
        return;
    }

    if (!root->left && !root->right) {
        huffman[root->letter] = encoding;
    }
    encode(root->left, encoding: encoding + "0", &: huffman);
    encode(root->right, encoding: encoding + "1", &: huffman);
}
```

1
2
3

Photo-2

- First part of the first code, count frequency and add a map as you can see in Photo-1.
- Second part of the first code, create priority queue for Huffman tree as you can see in Photo-1.
- Third part of the first code, create leaf node and add it to priority queue as you can see in Photo-1.
- Fourth part of the first code, create a new internal node with a frequency equal to the sum of the frequencies of the two nodes. Add the new node in priority queue as you can see in Photo-1.

- Fifth part of the first code, determine the root of the priority queue's top as you can see in Photo-1.
- Sixth part of the first code, create unordered map for store Huffman tree's letter and frequency as you can see in Photo-1.
- First part of the second code, check whether my root is null or not as you can see in Photo-2.
- Second part of the second code, the last step of my recursive function. When the root has no children, it stores Huffman codes in a map as you can see in Photo-2.
- Third part of the second code, it encodes roots with a child by giving 0 to the left and 1 to the right as you can see in Photo-2.

2. Decoding Algorithm

```
while (index < (int)encoding.size() - 2) {
    decode(root, &index, encoding, &fout);
}
```

Photo-3

```
void decode(Node* root, int &index, string decoding,ofstream& fout){
    if (root == nullptr) {
        return;
    }                                   1

    if (!root->left && !root->right)
    {
        fout << root->letter;           2
        return;
    }

    index++;
    if (decoding[index] =='0')                          3
        decode(root->left, &index, decoding, &fout);
    else
        decode(root->right, &index, decoding, &fout);
}
```

Photo-4

- In the first code, decode the encoded string as you can see in Photo-3.
- First part of the second code, check whether my root is null or not as you can see in Photo-4.
- Second part of the second code, find a leaf and print it as you can see in Photo-4.
- Third part of the second code, it decodes roots as you can see in Photo-4.

3. Tree command (-l)

```
void printTree(Node *root,unordered_map<char, string> &huffman, int number,ofstream& fout){
    if (root == NULL){
        return;
    }                                    1

    number += TEN;                       2

    printTree(root->right, &huffman, number, &fout);  3
    fout<<"tree";

    for (int i = TEN; i < number; i++){
        fout<<"-";                                    4
    }

    if(!root->letter){
        fout<<"|";
    }

    fout<<root->letter<<endl;

    printTree(root->left, &huffman, number, &fout);   5
}
```

Photo-5

- First part of the code check whether my root is null or not as you can see in Photo-5.
- Second part of the code, TEN is constant value.

```
#define TEN 10
```

- Third part of the code, process right child by recursive function.
- Fourth part of the code, print the Huffman Tree.
- Fifth part of the code, process left child by recursive function.