Hacettepe University

Computer Engineering Department

BBM 204: Algorithms Lab. - 2020 Fall

Assignment 4 - Trees

February 16, 2021

Student Name:
Eyyüp Ensar Topuz

Student Number:
b21827905

1 Problem Definition

In this assignment we need to implement file compression algorithm based on Huffman coding with trees. Our program should compress given file with Huffman encoding also our program should decompress given file with same algorithm.

Our program has 4 operation:

1. Encoding given file
2. Decoding given file
3. Return Huffman code of character in tree
4. Print Hufmann tree

2 Solution for Encoding

My encoding algorithm based on four parts:

Part 1: Finding frequency of each character

I used map for counting operation because it is easy to implement it and it is more efficient than other structures. The algorithm is so basic. If key exist in map; add 1 to the value , if not; set value to 1. I send all values to vector after this step.

```
1          // -- counting algorithm with hashmaps
2          for (char _c : line) {
3              // if tempMap has a value paired with key c add 1 to value
4              char c = tolower(_c);
5              if (tempMap.count(c)) {
6                  tempMap[c]++;
7              }
8              // if no value paired with key c instantiate it
9              else {
10                 tempMap[c] = 1;
11             }
12         }
```

Part 2: Creating Binary Tree

I divided this part to three parts. If there is 1 or 2 nodes, if there are 3 nodes or if there are more than 3 nodes but I only added more than 3 nodes part in this document because in most case we have more than 3 unique characters in our Huffman Tree. I have vector named as *nodes* which holds Node objects. It will destroy herself after this steps so it's not cost memory. My algorithm is simple it creates new nodes with data and frequency from nodes vector. Firstly it adds first 4 nodes in vector to *nodeNotLeaf1* and *nodeNotLeaf2* nodes. After than that, in line 23; I created another node named as *tempSibling*. I set *nodeNotLeaf1*'s all datas in to *tempSibling* including right node and left node. In line 26 I set *siblingsNotLeaf1*'s left node to *tempSibling* node, In line 27 I set *nodeNotLeaf1*'s right node to node(from nodes vector). In lines 28 and 29 I calculated frequency of nodes which are not leaf. After the line 30 my algorithm also similar.

```
1       Node* nodeNotLeaf1 = new Node('\0', -2); // Node(data, frequency)
2       Node* nodeNotLeaf2 = new Node('\0', -2); // freq. changes in belove
3       // if there is more than 3 unique char
4           for (int i = 0; i < nodes.size(); i++) {
5               char data = nodes.at(i).data;
6               int freq = nodes.at(i).frequency;
7               Node* node = new Node(data, freq);
8               if (i == 0)
9                   nodeNotLeaf1->left = node;
10              else if (i == 1)
11                  nodeNotLeaf1->right = node;
12              else if (i == 2)
13                  nodeNotLeaf2->left = node;
14              else if (i == 3) {
15                  nodeNotLeaf2->right = node;
16                  nodeNotLeaf1->frequency = nodeNotLeaf1->right->frequency +
17          nodeNotLeaf1->left->frequency;
18                  nodeNotLeaf2->frequency = nodeNotLeaf2->right->frequency +
19          nodeNotLeaf2->left->frequency;
20              }
21              else if (i % 2 == 0) {
22                  Node* tempSibling = new Node('\0', nodeNotLeaf1->frequency);
23                  tempSibling->right = nodeNotLeaf1->right;
24                  tempSibling->left = nodeNotLeaf1->left;
25                  nodeNotLeaf1->left = tempSibling;
26                  nodeNotLeaf1->right = node;
27                  nodeNotLeaf1->frequency = nodeNotLeaf1->right->frequency +
28          nodeNotLeaf1->left->frequency;
29              }
30              else if (i % 2 == 1) {
31                  Node* tempSibling = new Node('\0', nodeNotLeaf2->frequency);
32                  tempSibling->right = nodeNotLeaf2->right;
33                  tempSibling->left = nodeNotLeaf2->left;
34                  nodeNotLeaf2->right = tempSibling;
35                  nodeNotLeaf2->left = node;
36                  nodeNotLeaf2->frequency = nodeNotLeaf2->right->frequency +
37           nodeNotLeaf2->left->frequency;
38              }
39          }
40          root->left = nodeNotLeaf1;
41          root->right = nodeNotLeaf2;
42          root->frequency = root->left->frequency + root->right->frequency;
```

Part 3: Creating Hufmann Codes of Every Node

In this part starting from root of the tree, I added Hufmann codes for every node. In the start code was empty so code of root is empty. After the root of tree; when node goes left it adds 0 to code, when node goes right it adds 1 to code. I used pre-order traversal for adding codes to nodes. It is so easy to implement and I think it is the most efficient way.

```
1 void BinaryTree::createHufmannCodes(Node* root, std::string code)
2 {
3     if (root == nullptr)
4         return;
5     root->huffmanCode = code;
6     createHufmannCodes(root->left, code+"0"); // goes left, add 0 to code
7     createHufmannCodes(root->right, code+"1"); // goes right, add 1 to code
8 }
```

Part 4: Printing Encoded Message

In this step I used pre-order traversal to find char c in tree it adds Huffman code of char to output. I used this output reference to print decoded message in console. Also I save this encoded message to file named as `encoded_message_b21827905.txt`, it can be useful for decoding operation.

```
1 void BinaryTree::encode(Node* node, char c, std::string& output)
2 {
3     if (node == nullptr)
4         return ;
5     if (c == node->data)
6         output+= node->huffmanCode;
7     encode(node->left ,c,output);
8     encode(node->right ,c,output);
9 }
```

3 Solution for Decoding

In this step, I used iterative method. I loop through code(encoded message), if character c is equal to 0, node goes left; if c is equal to 1, node goes right. In the line 11, it checks whether node is leaf or not. In the other word it checks char in the node. If node has char it adds this char to output also it set our nodes to root of this tree. If node has no char it continue until find any leaf.

```
1  void BinaryTree::decode(Node* node, std::string code, std::string& output)
2  {
3      if (node == nullptr)
4          return;
5      for (char c : code) {
6          if (node) {
7              if (c == '0')
8                  node = node->left;
9              else if (c == '1')
10                 node = node->right;
11             if (node->data != '\0') {
12                 output += node->data;
13                 node = this->root;
14             }
15         }
16     }
17 }
```

4 Solution of printing tree

I used in-order traversal in this step. First it goes right instead of left because writing right node upper looks clear. When we first call this method, cell is equal to 1.This cell is increase when it goes left or right, when it returns back it is normally returns old value. This code prints double whitespaces value of cell times. It prints frequency. During traversal if current node is not leaf it prints 10 dash.

```
1
2       void BinaryTree::printTree(Node* node, int cell)
3   {
4       if (node == NULL)
5           return;
6       printTree(node->right, cell+6);
7       for (int i = 0; i < cell; i++)
8           std::cout << "  ";
9       if (node->data != '\0') // if current node is leaf
10          std::cout << node->frequency << std::endl;
11      else { // if current node is not leaf
12          std::cout << node->frequency;
13          std::cout << "----------";
14          std::cout << std::endl;
15      }
16      std::cout << "\n";
17      printTree(node->left, cell+6);
18  }
```

5 How To Run

Compile: g++ -std=c++11 *cpp -o Main

Also my makefile is working properly you can just type make in terminal.

Commands :

1. ./Main -i `input_file` -encode
2. ./Main -i `encoded_file` -decode
3. ./Main -s char
4. ./Main -l

Notes :

- I create a file named as `encoded_message_b21827905.txt`, it contains encoded message of input file which you give as input to our program. You can use it as input file for decode operation.

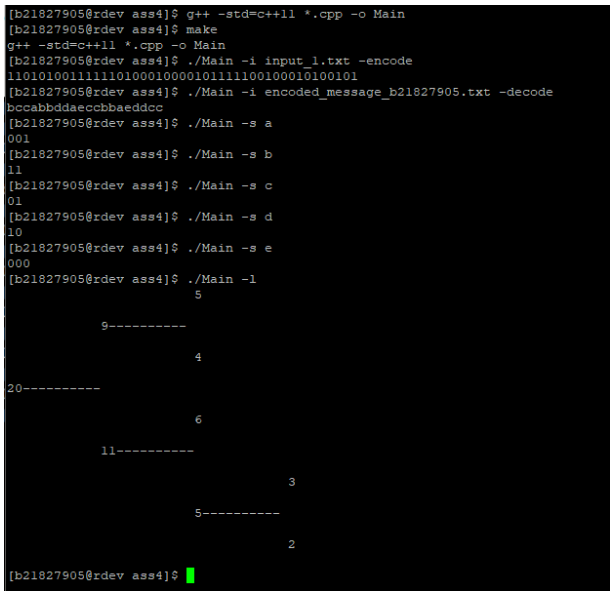

Figure 1: My run results on dev server