Hacettepe University

Computer Engineering Department

BBM203 Data Structures Lab - 2020 Fall

Assignment 4

January 2, 2021

Student name:
Sümeyye Meryem Taşyürek

Student Number:
b21827871

1 Building the Huffman Tree

- I have used pointer-based implementation to build the tree. So basically I have created linked list of nodes in the tree.
- After reading the given line I have counted the frequencies of the letters then I created a leaf node for each character in the string and add them to the priority queue.
- With the usage of priority queue the highest priority is given to the item with the lowest frequency.
- While there is more than one node in the queue, I remove the two nodes with the highest priority lowest frequency) and created an internal node with these two nodes as children and with frequency equal to the sum of two nodes frequencies. And add this new node to the priority queue.
- The remaining node in the priority queue is became the root of the tree.
- Also I implemented functions to store and retrieve the tree for the next usages.

2 Encoding Algorithm and Code

- I have gave the root node as parameter to the encoding function and used it to traverse the Huffman Tree and store Huffman Codes in a map. I have called the encoding function recursively for the right and left sub trees. After reaching a leaf node, function returns to the parent nodes.

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

    // if it finds a leaf node, map the code to the char
    if (isLeaf(root)) {
        huffmanCode[root->ch] = (str != EMPTY_STRING) ? str : "1";
    }

    //go left and add 0 to the code
    encode(root->left, str + "0", huffmanCode);
    //go right and add 1 to the code
    encode(root->right, str + "1", huffmanCode);
}
```

3 Decoding Algorithm and Code

- For decoding a given code we need the Huffman tree again so I read the data of the tree and build the tree again to keep the root.
- Decoding function takes the root node as a parameter, and also the decoded string to keep the track of the traversing.
- According to bits in the string it visits the right and left sub trees of the nodes with recursive calls and when it reaches a leaf node it prints the char value and starts to return to its parent nodes.

```
1  void decode(Node* root, int &index, string str)
2  {
3
4      if (root == nullptr) {
5          return;
6      }
7
8      // if it finds a leaf node print it out
9      if (isLeaf(root)) {
10         if(root->ch == '*'){//* represents the space
11             cout << "␣";
12         }
13         else{
14             cout << root->ch;
15         }
16         return;
17     }
18
19     index++;
20
21     if (str[index] == '0') { //get 0 go left in the tree
22         decode(root->left, index, str);
23     }
24     else { // get 1 go right in the tree
25         decode(root->right, index, str);
26     }
27 }
```

4 Listing the Tree

- Again I firstly retrieved the tree with the read function and kept the root. I have used a print function and give the root node as a parameter.
- Function visits the left and right nodes of the given node with the recursive calls of itself. When it reaches a leaf node it prints the char value and returns to the parent nodes. If a node is not a leaf it prints the frequency of the node.
- Example of the tree representation:

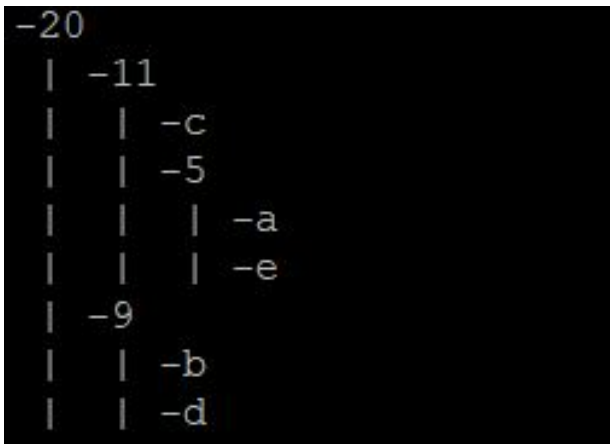

Figure 1: Output

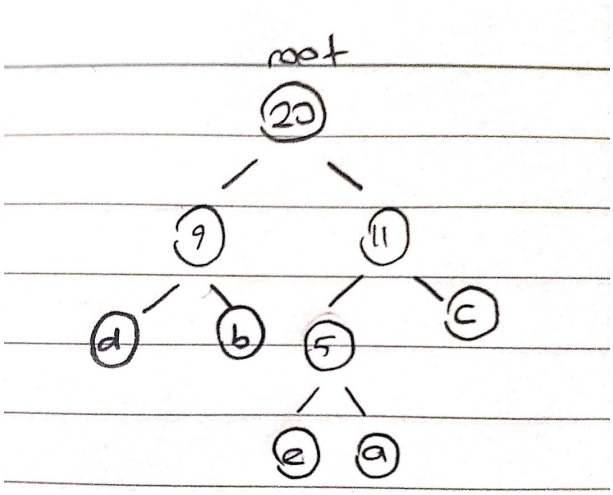

Figure 2: Tree