HACETTEPE UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BBM203 SOFTWARE LABORATORY 1

ASSIGNMENT 4

Subject: Trees

Deadline: 31.12.2020

Programming Language: C++

İsmail ATEŞ – 21626953

Defining Problem

In this assignment, we will implement a file compression algorithm based on Huffman coding with trees. At the end of this assignment:

1. Our program will compress given file using Huffman algorithm for encoding,

2. Our program will decompress given file using the same algorithm for decoding.

Explanation of My Approach

In this assignment, I learnt the Huffman coding with trees. Then I design "HuffmanCoding.h" header file, "HuffmanCoding.cpp" source file and main.cpp. I used unordered_map, priority_queue and structs for implementing. Decoding, listing three and returning an ASCII character commands can not executed without encoding operation. I created config file for checking encoding operation were realized and every run for program I build Huffman tree with tree.txt input which includes user's encoding file content.

Detailed explanation will be given later. We can summarize the program as follows:

1. Creating HuffmanCoding class object as "system".
2. Taking program arguments:
   2.1 if(argv[1]=="-i")
      2.1.1 if(argv[3]=="-encode")
            system.executeEncoding(argv[2])
      2.1.2 else if(argv[3]=="-decode")
            system.executeDecoding(argv[2])
   2.2 else if(argv[1]=="-s")
      system.executeReturn(argv[2])
   2.3 else if(argv[1]=="-l")
      system.executeListing()
3. return 0

Explanation of Encoding Algorithm

```
void HuffmanCoding::executeEncoding(string input_file) {
    encode_file.open(input_file,std::ios::in);
    string line;
    string total="";
    if (encode_file.is_open()){
        while (getline( &: encode_file, &: line)){
            line.erase( first: std::remove(line.begin(), line.end(), value: ' '), line.end());
            total=total+line+'\n';

        }
        total=total.substr( pos: 0, n: total.length()-1);
        config_file.open( s: "config.txt",std::ios::out);
        config_file<<"true";
        config_file.close();
        tree_file.open( s: "tree.txt",std::ios::out);
        tree_file<<total;
        tree_file.close();
        buildHuffmanTree(total);
        encodeString();
    }
    else{
        cout<<"hata";
    }
}
```

When executeEncoding operation were executed it first open the input file. It checks encode_file is opened. Then it reads the all lines in encode_file, erase spaces in line and adds chars to total string with for everyline “\n”. In while loop there is extra “\n” added so I substring the total string. Then I set config.txt file and tree.txt file. Then I call buildHuffmanTree() for building tree and encodeString() for encoding string and printing encoding.

BuildHuffmanTree():

```
void HuffmanCoding::buildHuffmanTree(string text)
{
    unordered_map<char, int> freq;
    for (char ch: text) {
        freq[ch]++;
    }

    priority_queue<Node*, vector<Node*>, comp> pq;

    for (auto pair: freq) {
        pq.push( x: getNode(pair.first, pair.second, left: nullptr, right: nullptr));
    }

    while (pq.size() != 1)
    {
        Node *left = pq.top(); pq.pop();
        Node *right = pq.top(); pq.pop();

        int sum = left->freq + right->freq;
        pq.push( x: getNode( ch: '\0', sum, left, right));
    }
    root = pq.top();
}
```

I add unoredered_map<char,int> freq for count frequency of appearance of each character and store it in a map. Then I create a priority queue to store live nodes of Huffman tree. Then I create a leaf node for each character and add it to the priority queue. Then I remove the two nodes of highest priority(lowest frequency) from the queue and create a new internal node with these two nodes as children with frequency equal to the sum of the two nodes' frequencies and add the new node to the priority queue. I did till there is more than one in the queue. Then I store the pointer of root Huffman tree.

EncodeString():

```
void HuffmanCoding::encodeString(string text) {
    unordered_map<char, string> huffmanCode;
    encode(root, str: "", &: huffmanCode);

    string str = "";
    for (char ch: text) {
        str += huffmanCode[ch];
    }
    cout << "\nEncoded string is :\n" << str << '\n';

}
```

This function creates unordered_map for store the values of every char and calls encode function. After encoding the string, it prints the encoded value.

Encode():

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

    if (!root->left && !root->right) {
        huffmanCode[root->char1] = str;
    }

    encode(root->left, str: str + "0", &: huffmanCode);
    encode(root->right, str: str + "1", &: huffmanCode);
}
```

This function adds chars strings values in ordered map, 0 or 1. İf char exist in the left of tree it adds “0” otherwise it adds “1”.

Explanation of Decoding Algorithm

```
void HuffmanCoding::executeDecoding(string input_file) {
    ifstream temp;
    temp.open( s: "config.txt",std::ios::in);
    string line;
    if (temp.is_open()){
        while (getline( &: temp, &: line)){
            if (line=="true"){
                encode_file.open( s: "tree.txt",std::ios::in);
                string line;
                string total;
                if (encode_file.is_open()) {
                    while (getline( &: encode_file,  &: line)) {
                        line.erase( first: std::remove(line.begin(), line.end(),  value: ' '), line.end());
                        total = total + line+'\n';

                    }
                    total=total.substr( pos: 0, n: total.length()-1);
                }
                buildHuffmanTree(total);
                decode_file.open(input_file,std::ios::in);
                if (decode_file.is_open()){
                    while (getline( &: decode_file, &: line)){
                        decodeString(line);
                    }
                }
            }else{
                cout<<"This command must not be performed without encoding operation"<<'\n';
            }
        }
    }else{
        cout<<"hata";
```

When executeDecoding operation were executed it first open the config file. It checks config file is opened.Then it checks config file content equals to “true”. True means before decoding encoding operation were done. Then it opens tree file for building huffmanTree. Tree.txt file includes encode_input before were used for encoding. Then it reads the all lines in tree file and adds chars to total string with for everyline “\n”. In while loop there is extra “\n” added so I substring the total string. Then I call buildHuffmanTree() for building tree and decodeString() for decoding string and printing decoded string.

BuildHuffmanTree():

```
void HuffmanCoding::buildHuffmanTree(string text)
{
    unordered_map<char, int> freq;
    for (char ch: text) {
        freq[ch]++;
    }

    priority_queue<Node*, vector<Node*>, comp> pq;

    for (auto pair: freq) {
        pq.push( x: getNode(pair.first, pair.second,  left: nullptr,  right: nullptr));
    }

    while (pq.size() != 1)
    {
        Node *left = pq.top(); pq.pop();
        Node *right = pq.top(); pq.pop();

        int sum = left->freq + right->freq;
        pq.push( x: getNode( ch: '\0', sum, left, right));
    }
    root = pq.top();
}
```

I add unoredered_map<char,int> freq for count frequency of appearance of each character and store it in a map. Then I create a priority queue to store live nodes of Huffman tree. Then I create a leaf node for each character and add it to the priority queue. Then I remove the two nodes of highest priority(lowest frequency) from the queue and create a new internal node with these two nodes as children with frequency equal to the sum of the two nodes' frequencies and add the new node to the priority queue. I did till there is more than one in the queue. Then I store the pointer of root Huffman tree.

DecodeString():

```
void HuffmanCoding::decodeString(string decodeString) {
    unordered_map<char, string> huffmanCode;
    encode(root,  str: "",  &: huffmanCode);

    int index = -1;
    cout << "\nDecoded string is: \n";
    while (index < (int)decodeString.size() - 2) {
        decode(root,  &: index, decodeString);
    }
}
```

This function creates unordered_map for store the values of every char and calls encode function. After encoding the string, it calls decode function.

Decode():

```
void HuffmanCoding::decode(Node* root, int &index, string str)
{
    if (root == nullptr) {
        return;
    }
    if (!root->left && !root->right)
    {
        cout << root->char1;
        return;
    }
    index++;

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

It is a recursive function. If it finds the char encode value it returns. And cout the char corresponding the value.

Explanation of Listing Tree Algorithm

```
void HuffmanCoding::executeListing() {
    ifstream temp;
    temp.open( s: "config.txt",std::ios::in);
    string line;
    if (temp.is_open()){
        while (getline( &: temp, &: line)){
            if (line=="true"){
                encode_file.open( s: "tree.txt",std::ios::in);
                string line;
                string total;
                if (encode_file.is_open()) {
                    while (getline( &: encode_file,  &: line)) {
                        line.erase( first: std::remove(line.begin(), line.end(),  value: ' '), line.end());
                        total = total + line+'\n';
                    }
                    total=total.substr( pos: 0, n: total.length()-1);
                }
                buildHuffmanTree(total);
                unordered_map<char, string> huffmanCode;
                encode(root,  str: "",  &: huffmanCode);
                printList(root, space: 0);

            }else{
                cout<<"This command must not be performed without encoding operation"<<'\n';
```

When executeListing operation were executed it first open the config file. It checks config file is opened.Then it checks config file content equals to “true”. True means before decoding encoding operation were done. Then it opens tree file for building huffmanTree. Tree.txt file includes encode_input before were used for encoding. Then it reads the all lines in tree file and adds chars to total string with for everyline “\n”. In while loop there is extra “\n” added so I substring the total string. Then I call buildHuffmanTree() for building tree and encode(). Then I call printList() function.

BuildHuffmanTree():

```
void HuffmanCoding::buildHuffmanTree(string text)
{
    unordered_map<char, int> freq;
    for (char ch: text) {
        freq[ch]++;
    }

    priority_queue<Node*, vector<Node*>, comp> pq;

    for (auto pair: freq) {
        pq.push( x: getNode(pair.first, pair.second,  left: nullptr,  right: nullptr));
    }

    while (pq.size() != 1)
    {
        Node *left = pq.top(); pq.pop();
        Node *right = pq.top(); pq.pop();

        int sum = left->freq + right->freq;
        pq.push( x: getNode( ch: '\0', sum, left, right));
    }
    root = pq.top();
}
```

I add unoredered_map<char,int> freq for count frequency of appearance of each character and store it in a map. Then I create a priority queue to store live nodes of Huffman tree. Then I create a leaf node for each character and add it to the priority queue. Then I remove the two nodes of highest priority(lowest frequency) from the queue and create a new internal node with these two nodes as children with frequency equal to the sum of the two nodes' frequencies and add the new node to the priority queue. I did till there is more than one in the queue. Then I store the pointer of root Huffman tree.

Encode():

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

    if (!root->left && !root->right) {
        huffmanCode[root->char1] = str;
    }

    encode(root->left,  str: str + "0",  &: huffmanCode);
    encode(root->right,  str: str + "1",  &: huffmanCode);
}
```

This function adds chars strings values in ordered map, 0 or 1. İf char exist in the left of tree it adds "0" otherwise it adds "1".

PrintList():

```
void HuffmanCoding::printList(Node *root1, int space) {
    if (root1 == NULL)
        return;

    space += COUNT;

    printList(root1->right, space);

    cout<<endl;
    for (int i = COUNT; i < space; i++)
        cout<<"_";
    if (root1->char1==NULL){
        cout<<"**"<<root1->frequence<<"\n";

    }else{
        cout<<root1->char1<<"->"<<root1->frequence<<"\n";

    }
    printList(root1->left, space);
}
```

It’s a recursive fuction. Base case is the root1 is null or not. In this function before it inrecase distance between levels with COUNT value. It defined global. Then it calls printList for right children of tree first then cout. If node have char value it represets with char-> frequence otherwise it represents with only frequence value “**value”. Then it calls printList() function for left children of tree.