Huffman Trees

String Compressing Alogrithm
Documentation

January 3, 2021

Student name: Student Number: Ozan Can Akın b21988988

1 Encoding Algorithm and Code

Listing 1: C++ code using listings

```
            
            i f ( situ==0){
            // f i l ereading and backup tree map
            ofstream fout ;
            fout . open(”prev . txt” );
            letters=f . read (path );
            items=l . count ( letters );
            // l is a LetterCounter object it iterates the
            // letters and count the frequency
            for ( dictItem y : items ){//backup for next uses
            fout<<y . letter <<”/−/”<<y . count<<endl ;
            }
            BinaryTree b;
            //Creating Huffman tree
            b. makeHuffman( items );

           //encode relating to my huffman tree
           for ( string y : letters ) {
           cout << b. findKey (b. getMainNode () , y );
```

```
        }
        cout<<endl ;
        }
```

1.1 *If (situation==0)*

”situ=0” Indicates we are in situation zero then we will encode in this step.Program first of all reads the txt file which is being at path position.Than uses l member of LetterCounter class.This class iterate the letters and return vector of dictItem(shown down page), dictItem is a struct which contains letter and that letters frequency .And now program writes this dict items to prev.txt for next usages of program.Lastly my program uses BinaryTree class for creating Huffman Tree.After the creation it prints binary types of each letter.I will explain the BinaryTree class on next page. I used ”/-/” for separating my copied map file.So we can use spaces easily.

Listing 2: dictItem

```
struct    dictItem{
int  count ;
string    letter ;
};
```

Listing 3: BinaryTree class

```
//node structs of binary tree struct TreeNode{
    TreeNode( const    string &letter ,    int    weight );
   TreeNode ∗ l e f t= nullptr ; TreeNode ∗
   right= nullptr ; string    letter ;
int    weight=0;
};
class BinaryTree { public :
    //when you add a new node to a parent node
    // you need to use  this    function    for    setting weight
    void setWeight (TreeNode ∗);
    //add a node to other node
    void addTree(TreeNode∗ ,TreeNode ∗);
    // this    function    for    sorting the    vectors of TreeNode
    static    bool comparebyWeight(TreeNode∗ ,TreeNode ∗);
    // this    is main function which sorts   others and
```

```
// indicates which node will connect to other
void makeHuffman( vector<dictItem >);
// i f a TreeNode f u l l returns 1
bool nodeFull (TreeNode ∗);
//This is a recursive function which finds the binary key of a letter
string findKey (TreeNode ∗treeNode , string letter );
// function of printing tree
void printTree (TreeNode∗ , int , vector<int >,int );
// recursive funciton for printing tree
// it indicates the count of spaces strict lines etc .
void printSpaces ( int , vector<int >,int );
// this func iterates the binary tree relating to given binary code
// when it finds a leaf returns that leafs letter .
void decode ( vector<string >);
private :
TreeNode ∗mainNode ;
int childs ;
public :
TreeNode ∗getMainNode ()const ;
};
```

Listing 4: makeHuffman function

```
void BinaryTree : : makeHuffman( vector<dictItem> letters ) { childs=letters . size ();
    vector<TreeNode*> nodes ;
    // all map items transed to TreeNode ' s for ( auto x :
    letters ){
        TreeNode *t=new TreeNode(x . letter , x . count );
        nodes . push_back ( t );
        }
        // sorts and adds TreeNodes whom weight is lowest
        while ( nodes . size ()>1){
        TreeNode* nP=new TreeNode("" ,0);
        sort ( nodes . begin () , nodes . end () , comparebyWeight );
        addTree( nodes [0] ,nP);
        addTree( nodes [1] ,nP);
        nodes . push_back (nP);
        nodes . erase ( nodes . begin ());
        nodes . erase ( nodes . begin ());
        }
        mainNode=nodes [ 0 ] ;
        }
```

1.2 *makeHuffman()*

First of all this function creates TreeNode's from dictItem's and put these TreeNode's a vector.Then a while loop starts.In this while loop vector of TreeNode's is being sorted when each joining to while loop.After sorting, two TreeNodes ,whom fraquencies are lowest,are being added to new TreeNode and this Tree node is being added to our vector.Of course program deletes the lowest two nodes from the vector.

Listing 5: findKey()

```
string BinaryTree : : findKey (TreeNode *treeNode , string key) { i f ( treeNode ==
        NULL) return NULL;
        // i f we find the key letter returns i f
        (treeNode−>letter==key){ return ”” ;
        }
        // i f return is different than ”none” it adds 0 for l e f t child
        // and adds 1 for right child
        i f (treeNode−>l e f t != nullptr ) {
        string res1 = findKey (treeNode−>left , key );
        i f ( res1 !=”none” ) return ”0” + res1 ;
        }
        i f (treeNode−>right != nullptr ) {

        string res2 = findKey (treeNode−>right , key );
        i f ( res2 != ”none”) return ”1” + res2 ;
        }
```

[1] 1.**3***findKey()*

This is a iterative function which calls a TreeNode’s left and right childs.And if it find the true key as letter returns cooperative string like if our key in root-¿left-¿right node the leaf node returns empty string.And the parrents adds 1 or 0 from left of that returning string.Until the returning iteration reaches root node.If detected node in left it returns 0+(previous returned string) but detected node in right it returns 1+(previous returned string).

```
// if key didn't matched any item in the tree returns none
return "none";
}
```

2 Decoding Algorithm and Code

Listing 6: Decode in Main Using

```
else if (situ==1 or situ==2 or situ==3){
        // situ==1 =>decoding / situ==2 => character searching /
    // situ==3 => Tree printing
        //previous copied tree writing
        items=f.Prev();
        BinaryTree b;
        //Creating Huffman tree from saved txt
        b.makeHuffman(items);
        //encode relating to my huffman tree
        if (situ==1){//decode code
        letters=f.read(path);
        b.decode(letters);
        cout<<endl; 14 }
          if (situ==2){//encoded version of a char
          string output=b.findKey(b.getMainNode(), searching);
          if (output!="none") {
          cout << "encoded bits of " << searching << " is : " << output << en
          } else {
          cout<<searching <<" is invalid character."<<endl;
          }
          }
          if (situ==3) {// plotting huffman tree
```

```
cout<<"huffman tree        for       previous encoded input"<<endl ;
vector<int> a ;
b. printTree (b. getMainNode () ,  0 , a ,    0);
cout << endl ;
}
```

situ is my situation determining variable if we are doing decoding,character searching or Tree printing we have common steps.Firstly we need to write file which we saved previously("prev.txt").Then we create the binary tree by the method ,mentioned on page two *makeHuffman()*.After the common steps we join the **if(situ==1)** because our decoding situation is 1.In this if structure we read given decoded txt file. After the read operation we use a method from BinaryTree class *decode()* I will explain this method on next page.

2.1 *decode() Function*

Listing 7: Decode

```
void BinaryTree : : decode ( vector<string> encoded) { TreeNode
    *node=new TreeNode("" ,0); node=mainNode ;
        for ( int i = 0; i < encoded . size (); i++) {
        i f ( stoi (encoded [ i ])==1){
        node=node−>right ;
        } else i f ( stoi (encoded [ i ])==0){
        node=node−>l e f t ;
        }
        i f (node−>letter != ""){
        cout<<node−>letter ;
        node=mainNode ; 13 }
    }
    }
```

This encoded vector includes zeros and ones as string.This function creates a new tree node and equates this tree node to root node.Then it starts to read that vector I mentioned first.If initial member of vector is 1.Right node of the initial node is visited.If initial member of vector is 0.Left node of the initial node is visited.If any leaf node is reached prints that nodes letter and initial node back to root node.

3 List Tree Command (-l)

Listing 8: Decode

```
i f (p != NULL) { i f ( indent )
        {
            // i f indent >0
                // print spacecs according to indent and separator vector
                printSpaces ( indent , separator , stat );
                i f (p−>letter !="") {
                // i f leafnode => print letter and weight ( frequency )
                cout << p−>letter <<"("<<p−>weight<<")"<<endl ;
                indent++;
```

```
} else {
// i f    ! leafnode −> print    weight ( frequency )
cout << p−>weight<<endl ;
}
} else {
string    line="   " ;
i f  (p−>letter !="") {
// i f    leafnode => print    letter and weight ( frequency )
cout <<line<<p−>letter <<"("<<p−>weight<<")" ;
cout<<endl ;
} else {
// i f    ! leafnode −> print    weight ( frequency )
cout<<line << p−>weight ;
cout<<endl ;
}
}
i f (p−>right != nullptr ) {
// after we printing    the    i n i t i a l    nodes values
//    we iterates    the    right node as  i n i t i a l    node
separator . push_back ( indent ); //adding separator
printTree (p−>right ,    indent+1,separator ,    2);
};

    i f (p−>l e f t    != nullptr    ) {
    // after we iterates    right node ,we iterates
    // the    l e f t    node as    i n i t i a l    node
    i f (p−>left −>letter !=""){
    separator . pop_back (); //removing separator
    printTree (p−>left ,    indent+1,separator ,0);
    } else {
    separator . pop_back (); //removing separator
    printTree (p−>left ,    indent+1,separator ,    1);
    }
    }
    }
    }
```

```
void BinaryTree : : printSpaces ( int count , vector<int> separator , int stat ) {
    // print the lines for printing binary tree
    string vertical=" ";
    for ( int i = 0; i < count ; ++i ) {
        if ( std : : count ( separator . begin () , separator . end () , i −1)){
            cout<<vertical <<" ";
        } else {
            cout<<" " ;
        }
    }

    string l e f t=" " ;
    string right=" " ;
    if ( stat==0 or stat==1) {
        cout << right ;
    } else {
        cout << l e f t ;
    }
}
```

3.1 *printTree() and printSpaces()*

This function is a recursive function.This function iterates firstly left nodes and then right nodes.When it goes through left node indent will be increased,And after right node printed indent will be decreased.You can think indent as depth of that node.And seperator vector collects the queue of seperator positions for each line.Finally *printTree()* uses *printSpaces()* print spaces called for every printed line. *printSpaces()* uses that indent and separators and print lines.Assume that indent=4 separator=[0,1,2,3] the printed spaces and separators will be like that ”l l l l—-a(5)” because every indent has separators in separator vector.

4 Showing Compiling and Some Debugging

```
[b21988988@rdev src]$ make
g++ -c -std=c++11 BinaryTree.cpp
g++ -c -std=c++11 Filer.cpp
g++ -c -std=c++11 LetterCounter.cpp
g++ -std=c++11 main.cpp BinaryTree.o Filer.o LetterCounter.o -o main
[b21988988@rdev src]$ echo "" > input.txt
[b21988988@rdev src]$ ./main -i input.txt -encode
This file is empty.We can't encode this file.
Segmentation fault
[b21988988@rdev src]$ echo "Happy New Years" > input.txt
[b21988988@rdev src]$ ./main -i input.txt -encode
0011110101101100011001001000010111000101100000111
[b21988988@rdev src]$ ^C
[b21988988@rdev src]$ echo "0011110101101100011001001000010111000101100000111" > decode_input.txt
[b21988988@rdev src]$ ./main -i decode_input.txt -decode
happy new years
[b21988988@rdev src]$ ./main -s h
encoded bits of h is: 0011
[b21988988@rdev src]$ ./main -s q
q is invalid character.
[b21988988@rdev src]$ ./main -l
huffman tree for previous encoded input
────15
    ├────7
    │    ├────3
    │    │    ├────s(1)
    │    │    └────a(2)
    │    └────4
    │         ├────p(2)
    │         └────y(2)
    └────8
         ├────4
         │    ├──── (2)
         │    └────e(2)
         └────4
              ├────2
              │    ├────h(1)
              │    └────n(1)
              └────2
                   ├────w(1)
                   └────r(1)

[b21988988@rdev src]$
```

```
[b21988988@rdev src]$ ./main -l
wrong file
please use encode command before this command
Segmentation fault
[b21988988@rdev src]$ ./main -s c
wrong file
please use encode command before this command
Segmentation fault
[b21988988@rdev src]$
```