Assignment

Design Analysis and Algorithms

**Huffman Coding | Greedy Algorithm**

*Prefix Codes, means the codes (bit sequences) are assigned in such a way that the code assigned to one character is not the prefix of code assigned to any other character. This is how Huffman Coding makes sure that there is no ambiguity when decoding the generated bitstream.*

**Submitted To**

*Mr. Ahsan Humayum*

**Submitted By**

*Usama Sarwar*

*FA17-BCS-090-B*

#include<iostream>

#include<vector>

#include<string>

using namespace std;

struct node

{

node \* leftChild;

node \* rightChild;

double frequency;

string content;

string code;

};

vector<node> nodeArray;// Use nodeArray to record all the nodes that may be created in the whole process

node extractMin()

{

double temp = (double) INT\_MAX;

vector<node>::iterator i1,pos;

for(i1 = nodeArray.begin();i1!=nodeArray.end();i1++)

{

if(temp>(\*i1).frequency)

{

pos = i1;

temp = (\*i1).frequency;

}

}

node tempNode = (\*pos);

nodeArray.erase(pos);

return tempNode;

}

node getHuffmanTree()

{

while(!nodeArray.empty())

{

node \* tempNode = new node;

node \* tempNode1 = new node;

node \* tempNode2 = new node;

\*tempNode1 = extractMin();

\*tempNode2 = extractMin();

tempNode->leftChild = tempNode1;

tempNode->rightChild = tempNode2;

tempNode->frequency = tempNode1->frequency+tempNode2->frequency;

nodeArray.push\_back(\*tempNode);

if(nodeArray.size() == 1)//only the root node exsits

{

break;

}

}

return nodeArray[0];

}

void BFS(node \* temproot,string s)

{

node \* root1 = new node;

root1 = temproot;

root1->code = s;

if(root1 == NULL)

{

}

else if(root1->leftChild == NULL && root1->rightChild == NULL)

{

cout<<"the content is "<<root1->content<<endl;

cout<<"and its corresponding code is "<<root1->code<<endl;

}

else

{

root1->leftChild->code = s.append("0");

s.erase(s.end()-1);

root1->rightChild->code = s.append("1");

s.erase(s.end()-1);

BFS(root1->leftChild,s.append("0"));

s.erase(s.end()-1);

BFS(root1->rightChild,s.append("1"));

s.erase(s.end()-1);

}

}

void getHuffmanCode()

{

int size,i;

double tempDouble;

string tempString = "";

cout<<"please input the number of things you want to encode!"<<endl;

cin>>size;

for(i = 0;i<size;i++)

{

cout<<"please input the things you want to encoded and their frequencies!"<<endl;

node tempNode;

cin>>tempString;

cin>>tempDouble;

tempNode.frequency = tempDouble;

tempNode.content = tempString;

tempNode.leftChild = NULL;

tempNode.rightChild = NULL;

nodeArray.push\_back(tempNode);

}

node root = getHuffmanTree();

BFS(&root,"");

}

int main()

{

vector<int> test;

test.push\_back(1);

test.push\_back(2);

test.push\_back(3);

test.push\_back(4);

vector<int>::iterator i1 = test.begin();

test.erase(i1);

getHuffmanCode();

return 0;

}

**Output:**

**The sample input is like:**

**6**

**a 45**

**b 13**

**c 12**

**d 16**

**e 9**

**f 5**

**and the output is like:**

**please input the number of things you want to encod**

**6**

**please input the things you want to encoded and the**

**a 45**

**please input the things you want to encoded and the**

**b 13**

**please input the things you want to encoded and the**

**c 12**

**please input the things you want to encoded and the**

**d 16**

**please input the things you want to encoded and the**

**e 9**

**please input the things you want to encoded and the**

**f 5**

**the content is a**

**and its corresponding code is 0**

**the content is c**

**and its corresponding code is 100**

**the content is b**

**and its corresponding code is 101**

**the content is f**

**and its corresponding code is 1100**

**the content is e**

**and its corresponding code is 1101**

**the content is d**

**and its corresponding code is 111**

**Press any key to continue**