HACETTEPE UNIVERSITY DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

ELE411: Data Structures Course Implementation Project: Huffman Coding

Node Class

```
class Node
{
    public:
    int Freq;
    char data;
    string code;
    Node *begin;
    Node *Next=nullptr;
   Node *left=nullptr;
    Node *right=nullptr;
    Node* End(Node* begin)
    {
      head=begin;
      while(head->Next!=nullptr)
   {
   head=head->Next;
   }
   return head;
    }
    Node* End()
    {
      head=Next;
      while(head->Next!=nullptr)
   {
   head=head->Next;
   }
   return head;
    }
private:
   Node *head;
};
```

Function of Pushing new Node to List

```
void push(Node** head_ref, char new_data)
{
    Node* new_node = new Node();

    new_node->data = new_data;
     new_node->Freq = 1;
    new_node->Next = (*head_ref);

    (*head_ref) = new_node;
}
```

Input String to List Converter

```
Node* insert(string message)
{
    Node *Chars=nullptr;
    Node *tempChars=nullptr;
    bool inserted=true;

    for (int i=0; i<message.length();i++)
    {
        if(Chars==nullptr)//only for first time
        {
            push(&Chars,message[i]);
        }
        else
        {
            tempChars=Chars;
            while(tempChars!=nullptr )
            {
```

Input String to List Converter(cont.)

```
                if(message[i]==tempChars->data)
                {
                    tempChars->Freq=tempChars->Freq+1;
                    inserted=true;
                    break;
                }
                inserted=false;
                tempChars=tempChars->Next;
            }
            if(!inserted)
            {
                push(&Chars,message[i]);
            }
        }
    }
    return Chars;
}
```

Sorting Algorithm for List

```
void sort(Node** Chars)
{
    Node* temp=*Chars;
    Node* temp2=temp;
    Node* Sorted=temp;

    while(Sorted!=nullptr)
    {
        temp=temp2;
        while(temp->Next!=nullptr)
        {
            if(temp->Freq > temp->Next->Freq)
            {
                int tempFreq=temp->Freq;
                temp->Freq=temp->Next->Freq;
                temp->Next->Freq=tempFreq;
```

Sorting Algorithm for List(cont.)

```
    char tempChar=temp->data;
    temp->data=temp->Next->data;
    temp->Next->data=tempChar;


    Node* tempNode=temp->left;
    temp->left=temp->Next->left;
    temp->Next->left=tempNode;

    tempNode=temp->right;
    temp->right=temp->Next->right;
    temp->Next->right=tempNode;




    }
    temp=temp->Next;
    }
    Sorted=Sorted->Next;
    }
    *Chars=temp2;
}
```

Print Functions

```
void printList(Node* node)
{
    cout<<"Data of list:"<<endl;
    Node* n;
    n=node;
    while (n != nullptr) {
        cout << n->data << " ";
        n = n->Next;
    }
    cout<<endl;
}
```

```
void printFreq(Node* node)
{
    cout<<"Frequencies of
list:"<<endl;
    Node* n;
    n=node;
    while (n != nullptr) {
        cout << n->Freq << " ";
        n = n->Next;
    }
    cout<<endl;
}
```

List to Huffman Tree Convertion

```
Node* huffmanTree(Node* Chars)
{
    Node* tChars=nullptr;
    Node* temp=new Node();
    Node* left=new Node();
    Node* right=new Node();

    if(Chars==nullptr||Chars->Next==nullptr )
    {
        sort(&tChars);
        printList(tChars);cout<<"in loop"<<endl;

        return tChars;
    }
    if(Chars->Next->Next==nullptr )
    {
        temp=new Node();
        temp->Next=tChars;
        temp->Freq=Chars->Freq+Chars->Next->Freq;
        temp->left=Chars;
        temp->right=Chars->Next;
        Chars=Chars->Next;
        Chars=Chars->Next;
        tChars=temp;
        sort(&Chars);
        sort(&temp);

        return tChars;
    }
```

List to Huffman Tree Convertion(cont.)

```
    temp=newNode();
    temp->Next=tChars;
    temp->Freq=Chars->Freq+Chars->Next->Freq;
    temp->left=Chars;
    temp->right=Chars->Next;
    Chars=Chars->Next;
    Chars=Chars->Next;
    tChars=temp;
    sort(&Chars);
    sort(&temp);
    Node* end=tChars->End(tChars);//end of tChars
    Node* head=Chars->End(Chars); //head of Chars
    head->Next=tChars;//add tChars to Chars
    end->Next=huffmanTree(Chars);//recursion with new list

    return tChars;
}
```

Print Functions

```
void printTreeLeftAndRight(Node* node)
{
	Node* n;
	Node* n2;
	n=node;
	while (n != nullptr) {

		if(n->right!=nullptr)
		{
		  n2=n->right;
		  cout <<"right of "<<n->data<<": "<< n2->data << " "<<endl;;
		}
		if(n->left!=nullptr)
		{
		  n2=n->left;
		  cout <<"left of "<<n->data<<": "<< n2->data << " "<<endl;
		}
		n = n->Next;
	}
}
```

Generate Huffman Code Function

```
void comCode(Node** head)
{
    Node* temp=*head;

    if(temp->left!=nullptr)
    {
        temp->right->code=temp->code+'1';
        temp->left ->code=temp->code+'0';
        comCode(&temp->right);
        comCode(&temp->left);

    }
    else return;
}
```

Print Functions

```
void printTreeCode(Node* node)
{
	cout<<endl<<"Codes of Leaves:"<<endl<<"---------------";
	Node* n;
	n=node;
	while (n != nullptr)
	{

		cout <<endl<<n->data<<": "<<n->code;
	    n = n->Next;
	}
	cout<<endl<<"---------"<<endl;
}
```

Compress Function

```
string compress(string message,Node* Chars)
{
    Node* temp=Chars;
    string turn;
    for (int i=0;i<message.length();i++)
    {
    temp=Chars;
    while(temp->Next!=nullptr)
    {
    if(message[i]==temp->data)
    {
    turn+=temp->code;
    break;
    }
    temp=temp->Next;
    }
    }
    return turn;
}
```

Decompress Function

```
string decompress(string message,Node* head)
{
     string ans = "";
    Node* curr = head;
    for (int i=0;i<message.size();i++)
    {
        if (message[i] == '0')
           curr = curr->left;
        else
           curr = curr->right;

        // reached leaf node
        if (curr->left==nullptr and curr->right==nullptr)
        {
            ans += curr->data;
            curr = head;
        }
    }
    return ans+'\0';
}
```

Main Function

```
string message="Hacettepe University Departmen of Electrical and Electronics Engineering ELE-411 Data Structure Course Implementation Project";
    cout<<"Input:"<<message<<endl;
    Node* Chars=insert(message);
    cout<<endl<<"Inserted Chars:"<<endl;
    printList(Chars);
    printFreq(Chars);

    sort(&Chars);
    cout<<endl<<"Sorted list:"<<endl;
    printList(Chars);
    printFreq(Chars);

    Node* head=Chars->End();//end node of list for adding new chars to the Chars
    head->Next=huffmanTree(Chars);
    cout<<"Data and Frequencies in Tree:"<<endl;
    printList(Chars);
    printFreq(Chars);
```

Main Function Out

Input:Hacettepe University Departmen of Electrical and Electronics Engineering ELE-411 Data Structure Course Implementation Project

Inserted Chars:
Data of list:
j P l C u S 1 4 - L g d l E f o m D y s r v i n U p t e c a H
Frequencies of list:
1 1 1 1 3 1 2 1 1 1 2 1 4 5 1 5 3 2 1 3 9 1 7 9 1 13 3 12 15 7 7 1

Sorted list:
Data of list:
j P l C S 4 - L d f y v U H 1 g D u m s p l E o i c a r n t e
Frequencies of list:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 3 3 3 3 4 5 5 7 7 7 9 9 12 13 15
Data and Frequencies in Tree:
Data of list:
j P l C S 4 - L d f y v U H 1 g D ...
Frequencies of list:
1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3 3 4 4 4 4 4 5 5 6 6 7 7 7 8 8 8 9 9 10 12 12 13 14 15 15 16 18 22 25 29 31 40 54 71 125

Main Function(cont.)

```
//cout<<endl<<"Left and Rights of Each Leaves :"<<endl;
//printTreeLeftAndRight(Chars); cout<<endl;

head=Chars->End();

comCode(&head);

//printTreeCode(Chars) ;

cout<<"message   :"<<endl<<message<<endl<<endl;

string compressed=compress(message,Chars);
cout<<"Compressed:"<<endl<<compressed<<endl<<endl;

cout<<"length of compressed message  :"<<compressed.length()<<endl;
cout<<"length of uncompressed message:"<<message.length()*8<<endl;

string decompressed=decompress(compressed,head);
cout<<endl<<"Decompressed:"<<endl<<decompressed<<endl<<endl;
```

Main Function Out(cont.)

message :
Hacettepe University Departmen of Electrical and Electronics Engineering ELE-411 Data Structure Course Implementation Project

Compressed:
10111111000010101111111111011000110110011011110110101001011101011110000010
01001111101110000110100001100011100011001111000010111101001111011011011001
11100100100110101111111100010001011000100100011000110110110100011110010010
0110101111111001110111010100010100010001111001101100111010011010110111100
01001101100111001111001011001111001011000101011110011010011000110100010001
11110000011010110111111000000000101111100000110001100110101011110100000110
00001001100110101000000100011100100110000101111011111100011110100111011101
0011010011110011101101001001101011111

length of compressed message :552
length of uncompressed message:1000

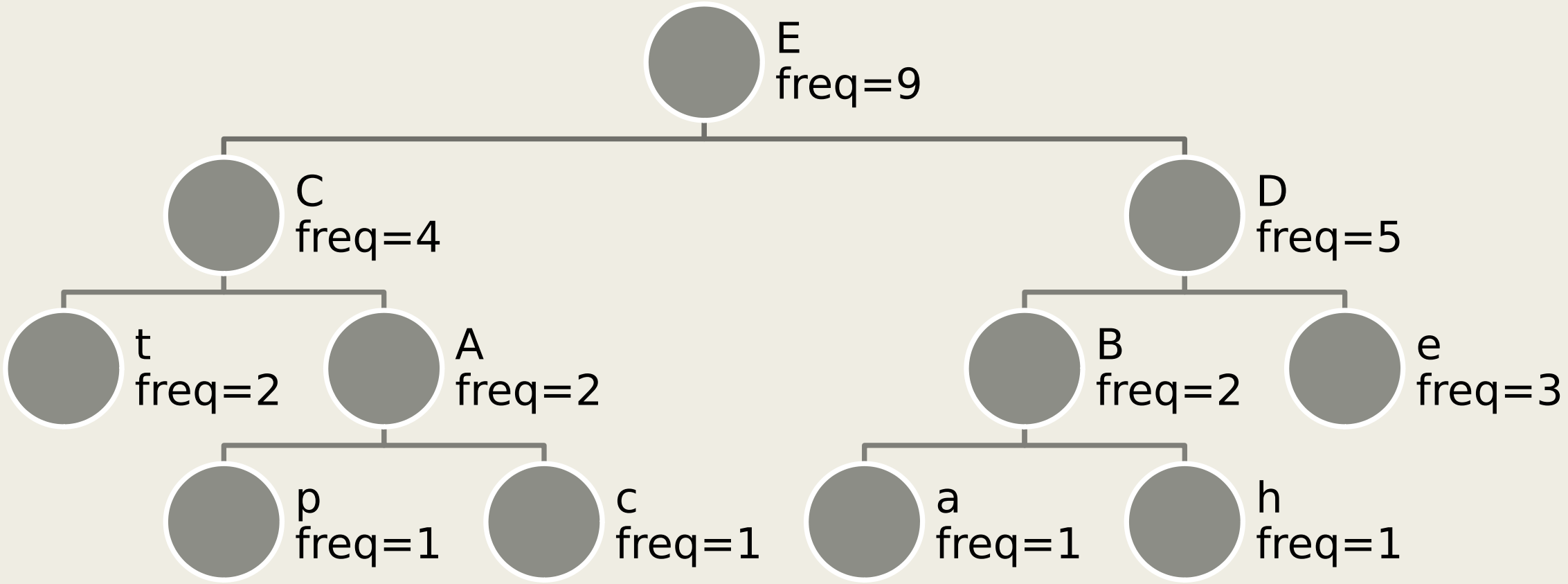
Decompressed:
Hacettepe University Departmen of Electrical and Electronics Engineering ELE-411 Data Structure Course Implementation Project

Basic Example(message=«hacettepe»)

Data and Frequencies in Tree:
Data of list:
p c a h t A B e C D E
Frequencies of list:
1 1 1 1 2 2 2 3 4 5 9

Left and Rights of Each Leaves :
right of A: c
left of A: p
right of B: h
left of B: a
right of C: A
left of C: t
right of D: e
left of D: B
right of E: D
left of E: C

Basic Example(message=«hacettepe»)



Basic Example(message=«hacettepe»)

message :
hacettepe

Compressed:
10110001111000011010 11

length of compressed message :22
length of uncompressed message:72

Decompressed:
hacettepe