ITC Coding

Name- Anmol Agrawal
Roll No. - 122CS0300
Date- 12/03/2025

Q.1 Adaptive Huffman Coding:

```
#include<bits/stdc++.h>

class node{
public:
	std::string ch;
	int weight;
	int no;
	int left;
	int right;
	int parent;
	std::string code;
};

// std::string data="aaaaaa";
std::string data;
std::string output;
std::vector<node>tree;
int NYT=0;
int nodeNo=51;
int currNode;

void createNode(std::string,int,int,int,std::string);
void encode();
```

```
bool first(int);
std::string retCodeData(std::string);
std::string givenCode(char);
void update(bool,std::string);
void gotoParent(int);
int findNodeMax(int);
void switchNodes(int,int);
void reNumCode(int);
void display();

int main(){
	std::cout<<"\nEnter the string to encode:";
	std::cin>>data;

	createNode("NYT",51,0,-1,"");
	encode();
	std::cout<<"\n\nAdaptive huffman code for "<<data<<"
is:\n"<<output<<"\n\n";

	system("pause");
}

//create a node and push it in tree
void createNode(std::string str,int num,int freq,int p,std::string c){
	node temp;
	temp.ch=str;
	temp.no=num;
	temp.weight=freq;
	temp.left=-1;
	temp.right=-1;
	temp.parent=p;
	temp.code=c;
```

```
        tree.push_back(temp);
}

void encode(){
        bool firstFlag;
        for(int a=0;a<data.length();a++){
                firstFlag=false;
                if(first(a)){
                        firstFlag=true;
                        output+=tree[NYT].code;
                        output+=givenCode(data[a]);
                } else {
                        output+=retCodeData(std::string(1,data[a]));
                }

                update(firstFlag,std::string(1,data[a]));
        }
}

//return true if symbol is first appearance
bool first(int n){
        for(int a=0;a<n;a++){
                if(data[a]==data[n]){
                        return false;
                }
        }
        return true;
}

//return code for the data element
std::string retCodeData(std::string str){
```

```
        for(int a=0;a<tree.size();a++){
                if(str==tree[a].ch){
                        currNode=a;
                        return tree[a].code;
                }
        }
        return "";
}

//return code for specific data element
std::string givenCode(char str){
        switch(str){
                case 'a':return "00000";break;
                case 'b':return "00001";break;
                case 'c':return "00010";break;
                case 'd':return "00011";break;
                case 'e':return "00100";break;
                case 'f':return "00101";break;
                case 'g':return "00110";break;
                case 'h':return "00111";break;
                case 'i':return "01000";break;
                case 'j':return "01001";break;
                case 'k':return "01010";break;
                case 'l':return "01011";break;
                case 'm':return "01100";break;
                case 'n':return "01101";break;
                case 'o':return "01110";break;
                case 'p':return "01111";break;
                case 'q':return "10000";break;
                case 'r':return "10001";break;
                case 's':return "10010";break;
                case 't':return "10011";break;
```

```
            case 'u':return "1010";break;
            case 'v':return "1011";break;
            case 'w':return "1100";break;
            case 'x':return "1101";break;
            case 'y':return "1110";break;
            case 'z':return "1111";break;
            default:return "";
        }
}

//update the tree
void update(bool flag,std::string str){
        if(flag){
            tree[NYT].ch="-";
            tree[NYT].left=tree.size();
            tree[NYT].right=tree.size()+1;


createNode("NYT",tree[NYT].no-2,0,NYT,tree[NYT].code+"0");
            createNode(str,tree[NYT].no-1,1,NYT,tree[NYT].code+"1");

            tree[NYT].weight++;
            NYT=tree[NYT].left;

            currNode=tree[NYT].parent;
        } else {
            int nodeMax=findNodeMax(currNode);
            if(nodeMax>0){
                switchNodes(currNode,nodeMax);
            }
            tree[currNode].weight++;
        }
```

```
        gotoParent(currNode);
}

//goto parent of node
void gotoParent(int n){
        int nodeMax;
        while(tree[n].parent!=-1){
                n=tree[n].parent;
                nodeMax=findNodeMax(n);

                if(nodeMax<0){
                        tree[n].weight++;
                } else {
                        switchNodes(n,nodeMax);
                        reNumCode(0);
                        tree[n].weight++;
                        nodeNo=51;
                }
        }
}

//return the node index with max node no. in block else -1
int findNodeMax(int n){
        int w=tree[n].weight;
        int index=n;

        for(int a=0;a<tree.size();a++){
                if(tree[a].weight==w&&tree[a].no>tree[index].no){
                        index=a;
                }
        }
```

```
        if(index==n){
                return -1;
        }

        return index;
}

//switch the nodes
void switchNodes(int a,int b){

        //swap parent's child
        int parent_a=tree[a].parent;
        int parent_b=tree[b].parent;
        int parent_a_left=tree[parent_a].left;
        int parent_a_right=tree[parent_a].right;
        int parent_b_left=tree[parent_b].left;
        int parent_b_right=tree[parent_b].right;


        //swap no.
        int temp=tree[a].no;
        tree[a].no=tree[b].no;
        tree[b].no=temp;

        //swap code
        std::string str=tree[a].code;
        tree[a].code=tree[b].code;
        tree[b].code=str;

        //swap parent
        temp=tree[a].parent;
```

```
        tree[a].parent=tree[b].parent;
        tree[b].parent=temp;

        // swap parents chid
        if(parent_a_left==a){
                tree[parent_a].left=b;
        } else {
                tree[parent_a].right=b;
        }

        if(parent_b_left==b){
                tree[parent_b].left=a;
        } else {
                tree[parent_b].right=a;
        }
}

//rearrange node no. and codes
void reNumCode(int n){
        if(tree[n].left!=-1&&tree[n].right!=-1){
                tree[tree[n].right].no=(--nodeNo);
                tree[tree[n].left].no=(--nodeNo);

                tree[tree[n].left].code=tree[n].code+"0";
                tree[tree[n].right].code=tree[n].code+"1";

                reNumCode(tree[n].right);
                reNumCode(tree[n].left);
        }
}

//display
```

```
void display(){
	for(int a=0;a<tree.size();a++){
		std::cout<<"\n"<<a<<" ch:"<<tree[a].ch<<"
weight:"<<tree[a].weight<<" no:"<<tree[a].no<<" l:"<<tree[a].left<<"
r:"<<tree[a].right<<" p:"<<tree[a].parent<<" code:"<<tree[a].code;
	}
	std::cout<<"\n";
	system("pause");
}
```

OUTPUT:

```
Output

Enter the string to encode:aardvark


Adaptive huffman code for aardvark is:
00000101000100000110001011010110001010



=== Code Execution Successful ===
```