ITC Coding

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

Q.2 Adaptive Huffman Decoding:

```
#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="00000101000100000110001011010110001010";
std::string data;
std::string output;
std::vector<node>tree;
int NYT=0;
int nodeNo=51;
int currNode=0;

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

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

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

	createNode("NYT",51,0,-1,"");

	decode();
	std::cout<<"\n\nAdaptive huffman decode for "<<data<<"
is:\n"<<output<<"\n\n";
	system("pause");
}

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

```
        tree.push_back(temp);
}

void decode(){
        int a=0,p;
        std::string code;
        bool firstFlag;

        do{
                if(tree[currNode].left==-1&&tree[currNode].right==-1){
                        if(currNode==NYT){
                                code=data.substr(a,4);
                                a=a+4;
                                p=b_to_d(code);
                                if(p<10){
                                        code+=data[a++];
                                        p=b_to_d(code);
                                        output+=givenCode(p,5);
                                } else {
                                        output+=givenCode(p,4);
                                }
                        } else {
                                output+=tree[currNode].ch;
                        }

                        firstFlag=first(output.substr(output.length()-1,1));
                        update(firstFlag,output.substr(output.length()-1,1));

                        currNode=0;
                } else {
                        if(data[a++]=='0'){
```

```
                        currNode=tree[currNode].left;
                    } else {
                        currNode=tree[currNode].right;
                    }
                }

        }while(a<=data.length());
}

//convert binary to decimal
int b_to_d(std::string bit){
        int dec=0;
        for(int a=bit.length()-1;a>=0;a--){
                if(bit[a]=='1'){
                        dec+=pow(2,bit.length()-a-1);
                }
        }
        return dec;
}

//return code for specific data element
std::string givenCode(int x,int bit){
        if(bit==5){
                if(x==0){
                        return "a";
                } else if(x==1){
                        return "b";
                } else if(x==2){
                        return "c";
                } else if(x==3){
                        return "d";
                } else if(x==4){
```

```
        return "e";
} else if(x==5){
        return "f";
} else if(x==6){
        return "g";
} else if(x==7){
        return "h";
} else if(x==8){
        return "i";
} else if(x==9){
        return "j";
} else if(x==10){
        return "k";
} else if(x==11){
        return "l";
} else if(x==12){
        return "m";
} else if(x==13){
        return "n";
} else if(x==14){
        return "o";
} else if(x==15){
        return "p";
} else if(x==16){
        return "q";
} else if(x==17){
        return "r";
} else if(x==18){
        return "s";
} else if(x==19){
        return "t";
}
```

```
        } else if(bit==4){
                if(x==10){
                        return "u";
                } else if(x==11){
                        return "v";
                } else if(x==12){
                        return "w";
                } else if(x==13){
                        return "x";
                } else if(x==14){
                        return "y";
                } else if(x==15){
                        return "z";
                }
        }
        return "";
}

bool first(std::string str){
        for(int a=0;a<tree.size();a++){
                if(tree[a].ch==str){
                        return false;
                }
        }
        return true;
}

//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;
                int 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_b_left=tree[parent_b].left;


        //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:00000101000100000110001011010110001010


Adaptive huffman decode for 00000101000100000110001011010110001010 is:
aardvark



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