Practical 3: Implementation of Adaptive Huffman Algorithm.

```
#include<bits/stdc++.h>
#include<iostream>
#include<vector>
using namespace std;
typedef struct node
{
      int weight,num;
      char ch;
      struct node *left;
      struct node *right;
}node;

//GLOBAL DEFINATIONS
typedef node* branch;
branch root=NULL,save=NULL,up=NULL;       //save and up pointer used to
store parent pointer so we can easily swap value by change left and right
of parent nodes
string encoded,input,decoded,output;
vector<char> v;         //to save discovered char
int flag=0,child=-1,side=-1;  //(save pointer)child =0 left child  child=1
right child same goes to (up pointer)side
char lastch;
bool entry=false;

//FUNCTION DECLARATION
branch new_branch(char,int,int);
void NodePath(branch,char , string &);
string getbinary(int,int);
string findcode(char);
char findchar(int,int);
void createtree();
int isfound(char);
void travel(branch);
void Rebalance(branch);
char getChar(int &);
void gotopath(string &,branch,int &);
void deletetree(branch);
void largestInBlock(branch,int,int&);
void swap2(branch,branch);
void reset()
{
      save=NULL;up=NULL;side=-1;child=-1;
}
```

```
int main()
{
      ifstream inFile;
      inFile.open("Originaladaptive.txt");
      char t;
      while(inFile>>t)
      {
            input+=t;
      }
      createtree();
      for(int i=1;i<input.length();i++)
      {
            flag=0;
            string strcode;
            if(isfound(input[i]))
            {
                  NodePath(root,input[i],strcode);
                  encoded+=strcode;
                  cout<<"\nPath to "<<input[i]<<" : "<<strcode;
            }
            else
            {
                  lastch=input[i];
                  v.push_back(input[i]);
                  NodePath(root,'#',strcode);
          cout<<"\nPath to NYT for finding "<<input[i]<< " : "<<strcode;
                  encoded+=strcode;
                  string charcode=findcode(input[i]);
                  encoded+=charcode;
          cout<<"\nAdd new character "<<input[i]<<" of code : "<<charcode;

            }
            Rebalance(root);
            entry=false;
      }
      cout<<endl<<"Traversal In Tree(node->num) : ";  travel(root);
      cout<<"\nEncoded : "<<encoded;
      int i=0;
      inFile.close();
      ofstream outFile;
      outFile.open("Compressedadaptive.txt");
      int repeat=encoded.size()/8;
      int len=repeat;
      int remained=encoded.size()%8;
      while(repeat--)
      {
            int sum=0;
```

```
		string chopped=encoded.substr(i,8);
		for(int j=0;j<8;j++)
		{
			if(chopped[j]=='1')
				sum+=pow(2,7-j);
		}
		i+=8;				//for next 8 bits
		cout<<"\nChopped String : "<<chopped<<" Decimal : "<<sum;
		char c=sum;
		outFile<<c;
	}
	int sum=0;
	string chopped=encoded.substr(i);
	for(int j=0;j<remained;j++)
	{
		if(chopped[j]=='1')
			sum+=pow(2,7-j);
	}
	cout<<"\nChopped String : "<<chopped<<" Decimal : "<<sum;
	outFile<<(char)sum;
	cout<<"\n--------------------End of encoding--------------------";
	outFile.close();
	inFile.close();
	inFile.open("Compressedadaptive.txt",ios::in | ios::binary);
	outFile.open("Outputadaptive.txt");
	inFile>>std::noskipws;
	while(!inFile.eof())
	{
		char x;
		int y;
		inFile>>x;
		if(inFile.eof())
			break;
		if((int)x<0)
			y=(int)x+256;
		else if(x==13)
		{
			if(inFile.peek()==10)
				inFile>>x;
			y=(int)x;
		}
		else
			y=(int)x;
		cout<<"\nChopped String : "<<getbinary(y,8)<<" Decimal : "<<y;
			decoded+=getbinary(y,8);
	}
	cout<<"\n--------------------End of Decoding--------------------";
	deletetree(root);
	decoded=decoded.substr(0,8*len+remained);
	cout<<endl<<"Decoded : "<<decoded;
	i=0;
	char x=getChar(i);
	branch nn=new_branch('*',0,101);	//alloting memory for new node
	nn->weight=0;
	nn->ch='#';
	branch par=new_branch('*',1,103);
	branch child=new_branch(x,1,102);
	par->left=nn;
	par->right=child;
	root=par;
	string path;
	while(1)
	{
		if(input.length()==output.length())
			break;
		path+=decoded[i];
		gotopath(path,root,i);
		if(i>=decoded.length())
			break;
	}
	cout<<endl<<"Traversal In Tree(node->num) : ";
	travel(root);
	cout<<endl<<"Decoded : "<<output;
	outFile<<output;
	if(decoded==encoded)
		cout<<"\nSTATUS[OK]	: The Decoded binary 100% matches 
with Encoded binary.......";
	else
		cout<<"\nSTATUS[FAIL] : Decoded binary does match with encoded 
binary........";
	if(input==output)
		cout<<"\nSTATUS[SUCCESS] : The Decoded string 100% matches 
with Encoded string.......";
	else
		cout<<"\nSTATUS[FAIL] : Decoded string does match with encoded 
string........";
}

branch new_branch(char c,int w,int n)
{
	branch nn;
	nn=new node();
	nn->left=NULL;
	nn->right=NULL;
	nn->weight=w;
	nn->num=n;
	nn->ch=c;
	return nn;
}
void NodePath(branch temp,char c, string &str)
{
	if(c==temp->ch)
	{
		if(c=='#')
		{
			branch l=new_branch('#',0,temp->num-2);
			branch r=new_branch(lastch,1,temp->num-1);
			temp->left=l;
			temp->right=r;
		   temp->ch='*'; //overwrite the old nyt(#) with *(inter node)
		}
		else
		{
			temp->weight++;
			int largest=-1;
			largestInBlock(root,temp->weight-1,largest);
			if(largest>temp->num &&temp!=root )
			{
				cout<<"\nCheck for : "<<temp->weight-1;
				cout<<"\nLARGEST FOUND : "<<largest;
				cout<<"\nSTATUS[REFRESHING] : Swapify the nodes
"<<temp->weight<<"("<<temp->num<<")  ";
			if(child==0 && side==0)
			{
			cout<<save->left->weight<<"("<<save->left->num<<")";
			swap2(save->left,up->left);
			}
			else if(child==0 && side==1)
			{
			cout<<save->left->weight<<"("<<save->left->num<<")";
			swap2(save->left,up->right);
			}
			else if(child==1 && side==0)
			{
			cout<<save->right->weight<<"("<<save->right->num<<")";
			swap2(save->right,up->left);
			}
			else if(child==1 && side==1)
			{
			cout<<save->right->weight<<"("<<save->right->num<<")";
			swap2(save->right,up->right);
```

```
                        }
                    }
                    reset();
                }
                flag=1;
            }
            if(temp->left!=NULL && flag==0)
            {
                str+="0";
                up=temp;
                side=0;
                NodePath(temp->left,c,str);
            }
            if(temp->right!=NULL && flag==0)
            {
                if(str.substr(str.size()-1)=="1")
                    str=str.substr(0,str.size()-1);
                str=str.substr(0,str.size()-1);
                str+="1";
                up=temp;
                side=1;
                NodePath(temp->right,c,str);
            }
        }

string getbinary(int n,int bit)   //converting int to 8 bit binary
{
        string str;
    for (int i=(bit-1);i>=0;i--)
    {
        int k=n>>i;
        if(k&1)
            str+="1";
        else
            str+="0";
    }
    return str;
}
string findcode(char c)
{
        int x=c;
        string code;
        if(x>=65 && x<=92)
        {
            x=x-65;
            code=getbinary(x,6);
        }
```

```
	else if(x>=97 && x<=110 )
	{
		x=x-71;
		code=getbinary(x,6);
	}
	else
	{
		x=x-91;
		code=getbinary(x,5);
	}
	return code;
}
char findchar(int v,int k)
{
	 if(k==5)
		v+=91;
	else if(v<=25 && k==6)
		v+=65;
	else if(v>=26 && k==6)
		v+=71;
	return char(v);
}
void createtree()
{
	branch nn=new_branch('#',0,101);	//alloting memory for new node
	encoded+=findcode(input[0]);
	branch par=new_branch('*',1,103);
	branch child=new_branch(input[0],1,102);
	par->left=nn;
	par->right=child;
	root=par;
	cout<<"New Tree Created with node characters : '*'(Internal Node) ,
'#'(NYT Node), "<<input[0]<<"(First character Node)";
	cout<<"\nFor Next character...";
	v.push_back(input[0]);		//saving discovered char in vector
}
int isfound(char c)
{
	for(int i=0;i<v.size();i++)
	{
		if(v[i]==c)
			return true;
	}
	return false;
}
void Rebalance(branch temp)
{
	if(temp->left->left!=NULL )  //only go to the node if it have childs
	{
		up=temp;
		side=0;
		Rebalance(temp->left);
	}
	if(temp->right->left!=NULL )  //in other words dont go to leaf nodes
	{
		up=temp;
		side=1;
		Rebalance(temp->right);
	}
	if(temp->left->ch=='#')      //NYT found so start summing the nodes
		entry=true;
	if(entry)
	{
		temp->weight=temp->left->weight+temp->right->weight;
		cout<<"\n\t\t\t\t\t\tSUM of "<<temp->left->ch<<" ("<<temp->left-
>num<<") and "<<temp->right->ch<<" ("<<temp->right->num<<") is "<<temp-
>weight;
		if(temp->weight>1 && temp!=root)
		{
			int largest=-1;
			largestInBlock(root,temp->weight-1,largest);
			if(largest>temp->num )
			{
				cout<<"\nCheck for : "<<temp->weight-1;
				cout<<"\nLARGEST FOUND : "<<largest;
				cout<<"\nSTATUS[REFRESHING] : Swapify the nodes
				"<<temp->weight<<"("<<temp->num<<")   ";
			if(child==0 && side==0)
			{
			cout<<save->left->weight<<"("<<save->left->num<<")";
			swap2(save->left,up->left);
			}
			else if(child==0 && side==1)
			{
			cout<<save->left->weight<<"("<<save->left->num<<")";
			swap2(save->left,up->right);
			}
			else if(child==1 && side==0)
			{
			cout<<save->right->weight<<"("<<save->right->num<<")";
			swap2(save->right,up->left);
			}
			else if(child==1 && side==1)
			{
```

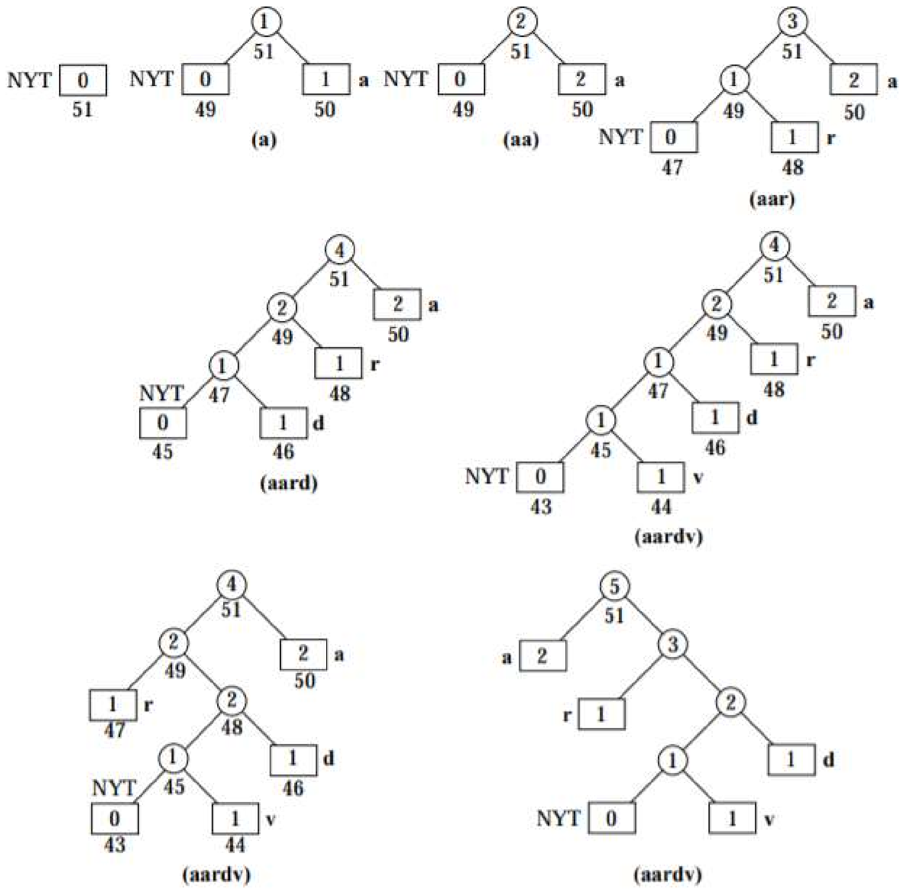
```
                        cout<<save->right->weight<<"("<<save->right->num<<")";
                        swap2(save->right,up->right);
                        }
                        reset();
                }
        }
        }
}
void travel(branch temp)
{
        if(temp!=NULL)
        {
                printf("%d  ",temp->num);
                travel(temp->right);
                travel(temp->left);
        }
}
char getChar(int &i)
{
        int sum=0,k=6;
        if(decoded[i]=='1' && (decoded[i+1]=='1'||decoded[i+2]=='1'))
                k=5;
        string code=decoded.substr(i,k);
        i+=k;
        for(int j=0;j<k;j++)
        {
                if(code[j]=='1')
                        sum+=pow(2,k-1-j);
        }
        char c=findchar(sum,k);
        cout<<endl<<"STATUS[MATCHED] : "<<code<<" matches with "<<c<<" where
k = "<<k;
        output+=c;
        return c;
}
void gotopath(string &path,branch temp,int &i)
{
        cout<<"\nPath : "<<path;
        for(int j=0;j<path.size();j++)
        {
                if(path[j]=='1')
                        temp=temp->right;
                else if(path[j]=='0')
                        temp=temp->left;
        }
        if(temp->ch=='#')
        {
```

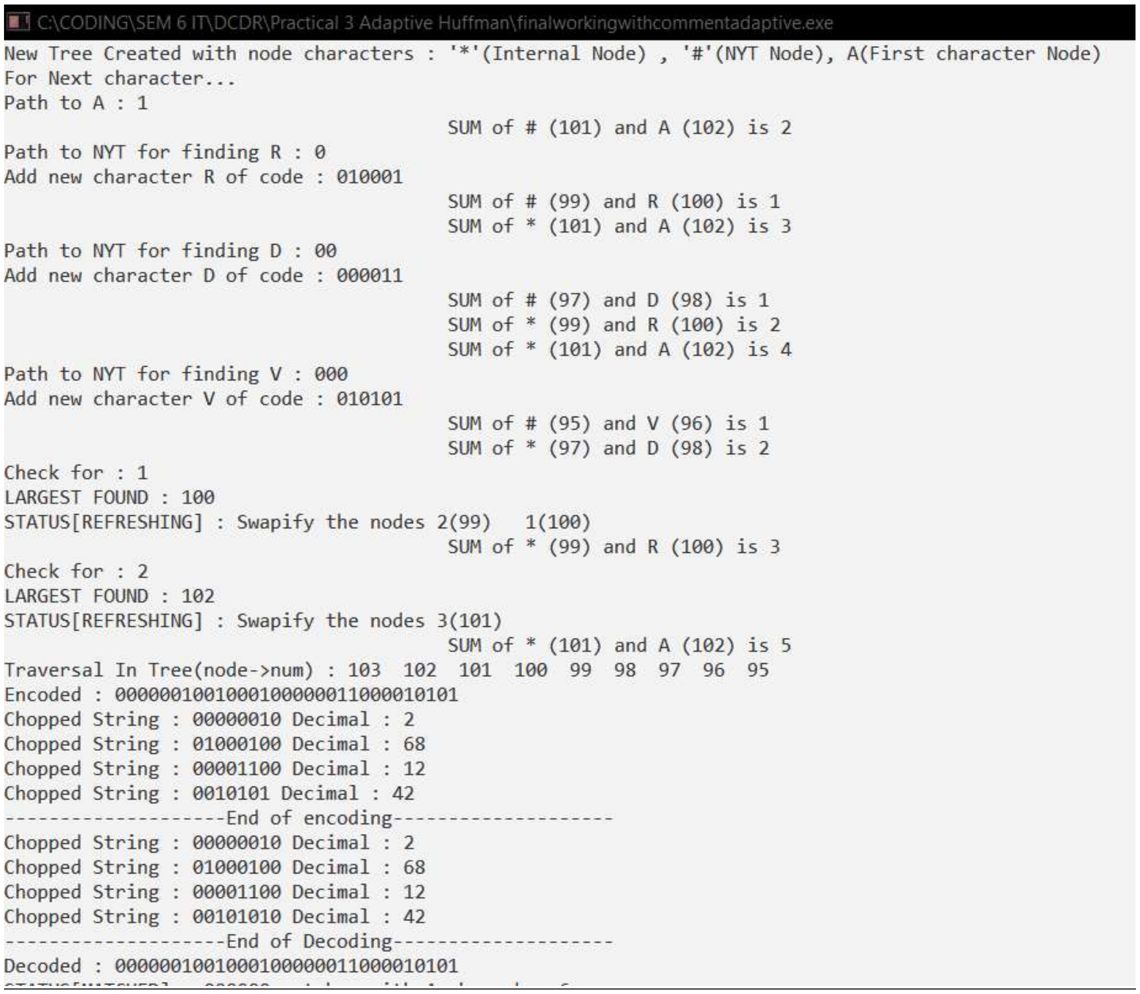
```
    cout<<" reaches to NYT meoutput the next k bits shows new character.";
            i++;
            branch l=new_branch('#',0,temp->num-2);
            branch r=new_branch(getChar(i),1,temp->num-1);
            temp->left=l;
            temp->right=r;
            temp->ch='*';  //overwrites old nyt data to make internal node
            path.clear();
      }
      else if(temp->ch!='*')        //if it not a internal node
      {
            i++;
            temp->weight++;
      cout<<" reached to Leaf Node where character saved is "<<temp->ch
           <<"  and weight increaded to the "<<temp->weight;
            output+=temp->ch;
            cout<<"\nSTATUS[CLEANING] : Clear Old Path Data";
            path.clear();
      }
      else
      {
            cout<<" leads to internal node.\nSTATUS[TRY AGAIN] :  by
adding next bit in the path string.";
            i++;
      }
      Rebalance(root);
}
void largestInBlock(branch temp,int v,int &largest)
{
      if(temp->left!=NULL)
      {
            if(v==temp->left->weight && largest < temp->left->num )
            {
                  largest=temp->left->num;
                  save=temp;
                  child=0;
            }
            if(v==temp->right->weight && largest < temp->right->num )
            {
                  largest=temp->right->num;
                  save=temp;
                  child=1;
            }
            largestInBlock(temp->left,v,largest);
            largestInBlock(temp->right,v,largest);
      }
}
```

```
void deletetree(branch tree)
{
	if(tree!=NULL)
	{
		deletetree(tree->left); deletetree(tree->right);free(tree);
	}
}
void swap2(branch a,branch b)
{
	branch t; t=a; a=b; b=a;
}
```

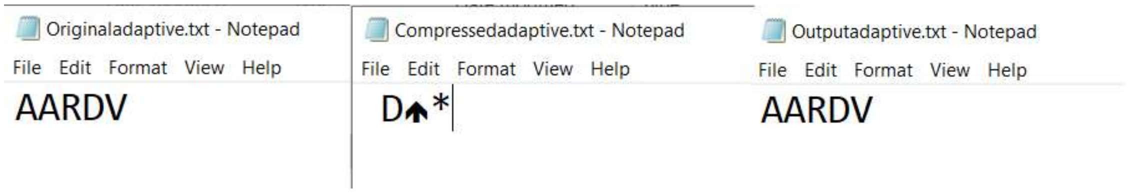
OUTPUT:

Example:




C:\CODING\SEM 6 IT\DCDR\Practical 3 Adaptive Huffman\finalworkingwithcommentadaptive.exe

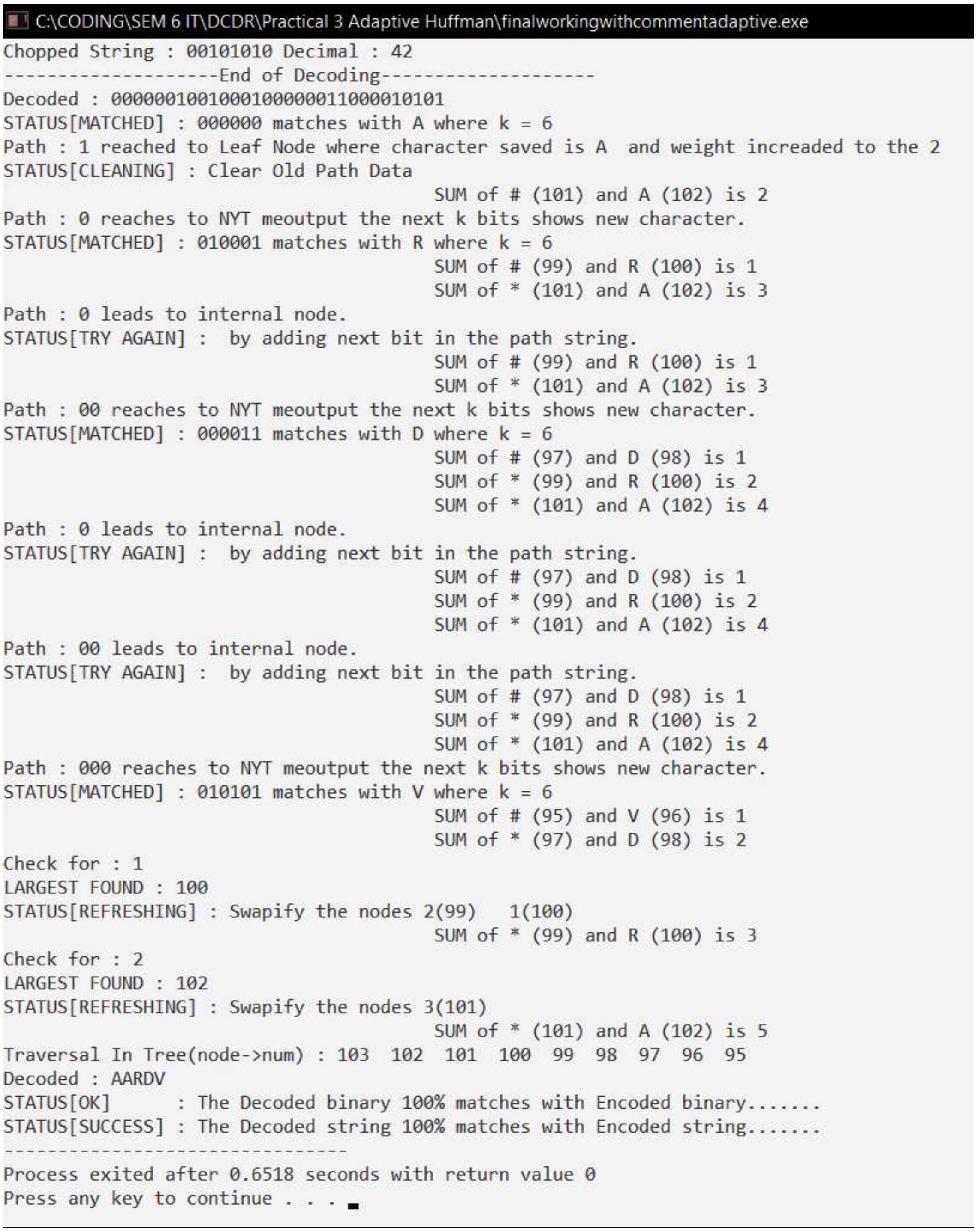
```
New Tree Created with node characters : '*'(Internal Node) , '#'(NYT Node), A(First character Node)
For Next character...
Path to A : 1
                                        SUM of # (101) and A (102) is 2
Path to NYT for finding R : 0
Add new character R of code : 010001
                                        SUM of # (99) and R (100) is 1
                                        SUM of * (101) and A (102) is 3
Path to NYT for finding D : 00
Add new character D of code : 000011
                                        SUM of # (97) and D (98) is 1
                                        SUM of * (99) and R (100) is 2
                                        SUM of * (101) and A (102) is 4
Path to NYT for finding V : 000
Add new character V of code : 010101
                                        SUM of # (95) and V (96) is 1
                                        SUM of * (97) and D (98) is 2
Check for : 1
LARGEST FOUND : 100
STATUS[REFRESHING] : Swapify the nodes 2(99)   1(100)
                                        SUM of * (99) and R (100) is 3
Check for : 2
LARGEST FOUND : 102
STATUS[REFRESHING] : Swapify the nodes 3(101)
                                        SUM of * (101) and A (102) is 5
Traversal In Tree(node->num) : 103  102  101  100  99  98  97  96  95
Encoded : 00000010010001000000110000010101
Chopped String : 00000010 Decimal : 2
Chopped String : 01000100 Decimal : 68
Chopped String : 00001100 Decimal : 12
Chopped String : 0010101 Decimal : 42
--------------------End of encoding-------------------
Chopped String : 00000010 Decimal : 2
Chopped String : 01000100 Decimal : 68
Chopped String : 00001100 Decimal : 12
Chopped String : 00101010 Decimal : 42
--------------------End of Decoding-------------------
Decoded : 00000010010001000000110000010101
```


Originaladaptive.txt - Notepad
File Edit Format View Help
Compressedadaptive.txt - Notepad
File Edit Format View Help
Outputadaptive.txt - Notepad
File Edit Format View Help

```
AARDV
```

```
D↑*
```

```
AARDV
```



```
C:\CODING\SEM 6 IT\DCDR\Practical 3 Adaptive Huffman\finalworkingwithcommentadaptive.exe
Chopped String : 00101010 Decimal : 42
--------------------End of Decoding-------------------
Decoded : 00000010010001000000110000101 01
STATUS[MATCHED] : 000000 matches with A where k = 6
Path : 1 reached to Leaf Node where character saved is A  and weight increaded to the 2
STATUS[CLEANING] : Clear Old Path Data
                                        SUM of # (101) and A (102) is 2
Path : 0 reaches to NYT meoutput the next k bits shows new character.
STATUS[MATCHED] : 010001 matches with R where k = 6
                                        SUM of # (99) and R (100) is 1
                                        SUM of * (101) and A (102) is 3
Path : 0 leads to internal node.
STATUS[TRY AGAIN] :  by adding next bit in the path string.
                                        SUM of # (99) and R (100) is 1
                                        SUM of * (101) and A (102) is 3
Path : 00 reaches to NYT meoutput the next k bits shows new character.
STATUS[MATCHED] : 000011 matches with D where k = 6
                                        SUM of # (97) and D (98) is 1
                                        SUM of * (99) and R (100) is 2
                                        SUM of * (101) and A (102) is 4
Path : 0 leads to internal node.
STATUS[TRY AGAIN] :  by adding next bit in the path string.
                                        SUM of # (97) and D (98) is 1
                                        SUM of * (99) and R (100) is 2
                                        SUM of * (101) and A (102) is 4
Path : 00 leads to internal node.
STATUS[TRY AGAIN] :  by adding next bit in the path string.
                                        SUM of # (97) and D (98) is 1
                                        SUM of * (99) and R (100) is 2
                                        SUM of * (101) and A (102) is 4
Path : 000 reaches to NYT meoutput the next k bits shows new character.
STATUS[MATCHED] : 010101 matches with V where k = 6
                                        SUM of # (95) and V (96) is 1
                                        SUM of * (97) and D (98) is 2
Check for : 1
LARGEST FOUND : 100
STATUS[REFRESHING] : Swapify the nodes 2(99)   1(100)
                                        SUM of * (99) and R (100) is 3
Check for : 2
LARGEST FOUND : 102
STATUS[REFRESHING] : Swapify the nodes 3(101)
                                        SUM of * (101) and A (102) is 5
Traversal In Tree(node->num) : 103  102  101  100  99  98  97  96  95
Decoded : AARDV
STATUS[OK]       : The Decoded binary 100% matches with Encoded binary.......
STATUS[SUCCESS] : The Decoded string 100% matches with Encoded string.......
--------------------------------
Process exited after 0.6518 seconds with return value 0
Press any key to continue . . .
```