Practical 2(a): Implementation of Huffman encoding algorithm :

1. First procedure Huffman algorithm
2. Minimum-variance Huffman algorithm
3. Extended-Huffman algorithm

1. First procedure Huffman algorithm

```
#include <bits/stdc++.h>
using namespace std;

struct bind
{
	char c;
	float val;
	string code;
	int left,right;
}data[100];

int first=0,n=0;
vector<int> v,avoid;

int isfound(int value)
{
	for(int i=0;i<avoid.size();i++)
	{
		if(avoid[i]==value)
			return true;
	}
	return false;
}
int check(string matchit)
{
	for(int i=0;i<v.size();i++)
	{
		if(matchit==data[v[i]].code)
```

```
                return v[i];
        }
        return -1;
}
void printlist()
{
        cout<<"\nList : ";
        for(int i=0;i<n;i++)
        {
                if(i==first)
                        cout<<" | ";
                cout<<data[i].val<<" ";
        }
}

string getbinary(int n)   //converting int to 8 bit binary
{
        string str;
    for (int i = 7; i >= 0; i--) {
        int k = n >> i;
        if (k & 1)
             str+="1";
        else
            str+="0";
    }
    return str;
}

void assign(int index,string s)
{
        data[index].code=s+data[index].code;
        if(data[index].left!=-1)
                assign(data[index].left,s);
        if(data[index].right!=-1)
```

```
            assign(data[index].right,s);
}

void huffman()
{
      if(data[first+1].val==0 || (data[first].val==0 && first!=0))
            return;
      for(int i=0;i<n-1;i++)          //sorting
      {
            for(int j=0;j<n-i-1;j++)
            {
                  if(data[j].val > data[j+1].val || data[j+1].val==data[j].val
&& data[j].c==NULL && data[j+1].c!=NULL)
                        swap(data[j],data[j+1]);
            }
      }
      printlist();
      float temp;
      cout<<endl<<data[first].val<<" + "<<data[first+1].val<<" = ";
      temp=data[first+1].val+data[first].val; //taking sum of 2 smallest number
      cout<<temp;
      data[n].left=first;
      data[n].right=first+1;
      data[n].val=temp;               //making a new entry in array of struct
      cout<<"\nAppended "<<data[n].val<<" left index : "<<data[n].left<<" right
index : "<<data[n].right<<endl;
      assign(first,"0");              //assign 0s to left part children
      assign(first+1,"1");            //assign 1s to right part children
      first+=2;
      n++;
      huffman();
}

int main()
{
```

```
   ifstream inFile;
   ofstream OutFile;
   std::map<char,int> trace; //dict type data structure
   char text;
   int count=0;
inFile.open("original.txt");
if (!inFile)
   {
    cout << "Unable to open file";
    exit(1); // terminate with error
}
inFile>>std::noskipws;
while (inFile >> text)    //read char and save to text
   {
         count++;
    trace[text]++; //key=text for occcurence count
}
int i=0;
map<char, int>::iterator itr;
cout << " KEY\tOCCURENCE\n";
for (itr = trace.begin(); itr != trace.end(); ++itr) {
   /* cout <<" "<< itr->first
         <<"\t   " << itr->second << '\n'; */
    data[i].left=-1;
         data[i].right=-1;
         data[i].c=itr->first;
         data[i++].val=itr->second;
         n++;
         cout <<" "<< data[i-1].c
         <<"\t   " << data[i-1].val << '\n';
}
inFile.close();
   printlist();
```

```
huffman();
std::map<char,string> findcode;               //for mapping purpose
int k=0;
cout<<"\nCharacter\tCodeword\n";
for(int i=n;i>=0;i--)
{
	if(data[i].left==-1 && data[i].right==-1)
	{
		findcode[data[i].c]=data[i].code;
		cout<<data[i].c<<"\t\t"<<findcode[data[i].c]<<endl;
		v.push_back(i);
	}
}
ofstream outFile;
outFile.open("Compressed.txt");
inFile.open("original.txt");
string str;
int test=0;
while(inFile>>text)
{
	str+=findcode[text];
	if(str.length()>8)
	{
		int sum=0;
		string w=str.substr(0,8);
		cout<<endl<<++test<<"   Chopped string : "<<w;
		for(int i=0;i<8;i++)
		{
			if(w[i]=='1')
				sum+=pow(2,7-i);
		}
		cout<<"\t Decimal : "<<sum;
		if(sum==26)
		{
```

```
                        cout<<"  Skipped";
                        avoid.push_back(test);
                }
                else
                        outFile<<(char)sum;
                str=str.substr(8);
        }
}
int sum=0;   //for remaining last characters which is less than 8
cout<<endl<<++test<<"   Chopped string : "<<str;
int cut=str.length();
for(i=0;i<cut;i++)
        {
                if(str[i]=='1')
                        sum+=pow(2,7-i);
        }
        cout<<"\t Decimal : "<<sum;
outFile<<(char)sum;
inFile.close();
outFile.close();
inFile.open("compressed.txt");
outFile.open("output.txt");
string matchit;
int dlen=0;
int test2=0;
while(inFile>>text)
{

                int deci=(int)text;
                if(deci<0)
                        deci+=256;
                string str=getbinary(deci);
                if(dlen==count-1)
                        str=str.substr(0,cut);
```

```
                cout<<endl<<++test2<<"   Binary decoded : "<<str;
                cout<<"\tDecimal value : "<<deci;
                up:
                for(int i=0;i<8;i++)
                {
                        matchit+=str[i];
                        int index=check(matchit);
                        if(index!=-1)
                                {
                                        outFile<<data[index].c;    dlen++;
                                        matchit.clear();
                                }
                }
                if(isfound(test2+1))
                        {
                         test2++;
                        str="00011010";
                        goto up;                                        }
        }
        cout<<"\nSize : "<<count<<"  Decoded Size : "<<dlen;
        //to check all the characters decoded or not
}
```

OUTPUT:

```
C:\CODING\SEM 6 IT\DCDR\Practical 2 Huffman\firstprocedurehuffman.exe
KEY     OCCURENCE

          4
          198
'         7
,         3
.         10
A         1
G         1
I         6
T         2
a         67
b         14
c         21
d         25
e         117
f         25
g         21
h         33
i         52
j         2
k         9
l         29
m         16
n         47
o         68
p         23
q         1
r         60
s         45
t         78
u         28
v         5
w         25
x         2
y         20

List :  | 4 198 7 3 10 1 1 6 2 67 14 21 25 117 25 21 33 52 2 9 29 16 47 68 23 1 60 45 78 28 5 25 2 20
List :  | 1 1 1 2 2 2 3 4 5 6 7 9 10 14 16 20 21 21 23 25 25 25 28 29 33 45 47 52 60 67 68 78 117 198
1 + 1 = 2
Appended 2 left index : 0 right index : 1

List : 1 1  | 1 2 2 2 2 3 4 5 6 7 9 10 14 16 20 21 21 23 25 25 25 28 29 33 45 47 52 60 67 68 78 117 198
1 + 2 = 3
Appended 3 left index : 2 right index : 3

List : 1 1 1 2  | 2 2 2 3 3 4 5 6 7 9 10 14 16 20 21 21 23 25 25 25 28 29 33 45 47 52 60 67 68 78 117 198
2 + 2 = 4
Appended 4 left index : 4 right index : 5
```

C:\CODING\SEM 6 IT\DCDR\Practical 2 Huffman\firstprocedurehuffman.exe

```
Character       Codeword
                00
e               011
t               1100
o               1010
a               1001
r               0101
i               11111
n               11100
s               11011
h               10001
l               10000
u               01000
w               111101
f               111100
d               111011
p               111010
g               110100
c               101111
y               101110
m               101100
b               010010
.               1101010
k               1011010
'               0100110
I               11010111
v               10110111

                01001111
,               110101101
x               101101101
j               101101100
T               010011101
q               010011100
G               1101011001
A               1101011000

1   Chopped string : 11010111    Decimal : 215
2   Chopped string : 11000100    Decimal : 196
3   Chopped string : 11011011    Decimal : 219
4   Chopped string : 00111001    Decimal : 57
5   Chopped string : 01011000    Decimal : 88
6   Chopped string : 01010111    Decimal : 87
7   Chopped string : 00100001    Decimal : 33
8   Chopped string : 01110001    Decimal : 113
9   Chopped string : 11101010    Decimal : 234
10   Chopped string : 11111111   Decimal : 255
11   Chopped string : 00011010   Decimal : 26  Skipped
12   Chopped string : 11101100   Decimal : 236
13   Chopped string : 11110110   Decimal : 246
```

original.txt - Notepad

File Edit Format View Help

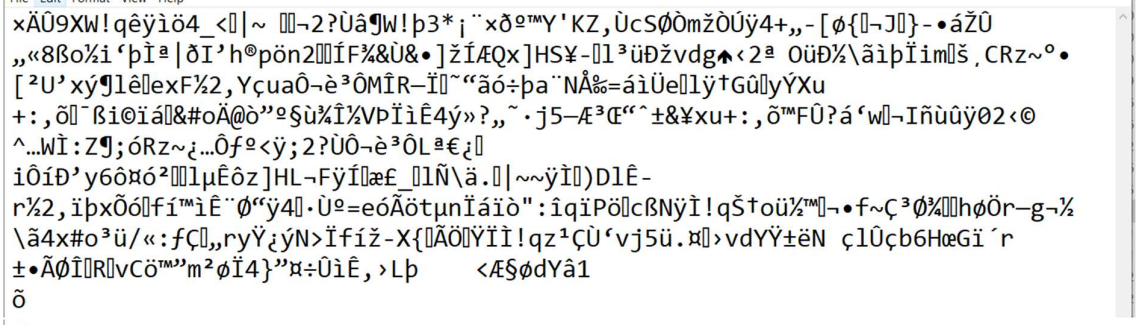
It's not only writers who can benefit from this free online tool. If you're a programmer who's working on a project where blocks of text are needed, this tool can be a great way to get that. It's a good way to test your programming and that the tool being created is working well.

Above are a few examples of how the random paragraph generator can be beneficial. The best way to see if this random paragraph picker will be useful for your intended purposes is to give it a try. Generate a number of paragraphs to see if they are beneficial to your current project.

If you do find this paragraph tool useful, please do us a favor and let us know how you're using it. It's greatly beneficial for us to know the different ways this tool is being used so we can improve it with updates. This is especially true since there are times when the generators we create get used in completely unanticipated ways from when we initially created them. If you have the time, please send us a quick note on what you'd like to see changed or added to make it better in the future.

Compressed.txt - Notepad

File Edit Format View Help



output.txt - Notepad

File Edit Format View Help

It's not only writers who can benefit from this free online tool. If you're a programmer who's working on a project where blocks of text are needed, this tool can be a great way to get that. It's a good way to test your programming and that the tool being created is working well.

Above are a few examples of how the random paragraph generator can be beneficial. The best way to see if this random paragraph picker will be useful for your intended purposes is to give it a try. Generate a number of paragraphs to see if they are beneficial to your current project.

If you do find this paragraph tool useful, please do us a favor and let us know how you're using it. It's greatly beneficial for us to know the different ways this tool is being used so we can improve it with updates. This is especially true since there are times when the generators we create get used in completely unanticipated ways from when we initially created them. If you have the time, please send us a quick note on what you'd like to see changed or added to make it better in the future.

2. Minimum-variance Huffman algorithm

```
#include <bits/stdc++.h>
using namespace std;

struct bind
{
                        char c;
                        float val;
                        string code;
                        int left,right;
}data[100];

int first=0,n=0;
vector<int> v,avoid;

int isfound(int value)
{
                        for(int i=0;i<avoid.size();i++)
                        {
                          if(avoid[i]==value)
                                  return true;
                        }
                        return false;
}
int check(string matchit)
{
                        for(int i=0;i<v.size();i++)
                        {
                          if(matchit==data[v[i]].code)
                                  return v[i];
                        }
                        return -1;
}
```

```
void printlist()
{
                        cout<<"\nList : ";
                        for(int i=0;i<n;i++)
                        {
                          if(i==first)
                                cout<<" | ";
                          cout<<data[i].val<<" ";
                        }
}

string getbinary(int n)   //converting int to 8 bit binary
{
                        string str;
    for (int i = 7; i >= 0; i--) {
        int k = n >> i;
        if (k & 1)
                        str+="1";
        else
            str+="0";
    }
    return str;
}

void assign(int index,string s)
{
                        data[index].code=s+data[index].code;
                        if(data[index].left!=-1)
                          assign(data[index].left,s);
                        if(data[index].right!=-1)
                          assign(data[index].right,s);
}

void huffman()
```

```
{
if(data[first+1].val==0 || (data[first].val==0 && first!=0))
                          return;
                        for(int i=0;i<n-1;i++)             //sorting
                        {
                          for(int j=0;j<n-i-1;j++)
                          {
                                 if(data[j].val > data[j+1].val )
                                        swap(data[j],data[j+1]);
                          }
                        }
                        printlist();
                        float temp;
                        cout<<endl<<data[first].val<<" + "<<data[first+1].val<<" =
";
                        temp=data[first+1].val+data[first].val;
                        cout<<temp;
                        data[n].left=first;
                        data[n].right=first+1;
                        data[n].val=temp;  //making a new entry in array of struct
                        cout<<"\nAppended "<<data[n].val<<" left index :
"<<data[n].left<<" right index : "<<data[n].right<<endl;
                        assign(first,"0");    //assign 0s to left part children
                        assign(first+1,"1");        //assign 1s to right part
children
                        first+=2;
                        n++;
                        huffman();
}

int main()
{

                        ifstream inFile;
                        ofstream OutFile;
```

```
                        std::map<char,int> trace;    //dict type data structure
                        char text;
                        int count=0;
    inFile.open("originalmin.txt");
    if (!inFile)
                        {
        cout << "Unable to open file";
        exit(1); // terminate with error
    }
    inFile>>std::noskipws;
    while (inFile >> text)    //read char and save to text
                        {
                           count++;
        trace[text]++;     //key=text for occcurence count (increment for every
occurence)
    }
    int i=0;
    map<char, int>::iterator itr;
    cout << " KEY\tOCCURENCE\n";
    for (itr = trace.begin(); itr != trace.end(); ++itr) {
       /* cout <<" "<< itr->first
            <<"\t   " << itr->second << '\n'; */
        data[i].left=-1;
                           data[i].right=-1;
                           data[i].c=itr->first;
                           data[i++].val=itr->second;
                           n++;
                           cout <<" "<< data[i-1].c
            <<"\t   " << data[i-1].val << '\n';
    }
    inFile.close();
                        printlist();
                        huffman();
                        std::map<char,string> findcode;                 //for
mapping purpose
```

```
int k=0;
cout<<"\nCharacter\tCodeword\n";
for(int i=n;i>=0;i--)
{
  if(data[i].left==-1 && data[i].right==-1)
  {
        findcode[data[i].c]=data[i].code;

cout<<data[i].c<<"\t\t"<<findcode[data[i].c]<<endl;
        v.push_back(i);
  }
}
ofstream outFile;
outFile.open("Compressedmin.txt");
inFile.open("originalmin.txt");
string str;
int test=0;
while(inFile>>text)
{
  str+=findcode[text];
  if(str.length()>8)
  {
        int sum=0;
        string w=str.substr(0,8);
        cout<<endl<<++test<<"   Chopped string : "<<w;
        for(int i=0;i<8;i++)
        {
              if(w[i]=='1')
                    sum+=pow(2,7-i);
        }
        cout<<"\t Decimal : "<<sum;
        if(sum==26)
              {
                    cout<<"   Skipped";       //avoid
adding this element because it will cause end of file later on
```

```
                                        avoid.push_back(test);
                                }
                        else
                                outFile<<(char)sum;
                        str=str.substr(8);
                  }
                }
                int sum=0;   //for remaining last characters which is less
than 8
                cout<<endl<<++test<<"   Chopped string : "<<str;
                int cut=str.length();
                for(i=0;i<cut;i++)
                  {
                        if(str[i]=='1')
                                sum+=pow(2,7-i);
                  }
                  cout<<"\t Decimal : "<<sum;
                outFile<<(char)sum;
                inFile.close();
                outFile.close();
                inFile.open("compressedmin.txt");
                outFile.open("outputmin.txt");
                string matchit;
                int dlen=0;
                int test2=0;
          while(inFile>>text)
        {
                        int deci=(int)text;
                        if(deci<0)
                                deci+=256;
                        string str=getbinary(deci);
                        if(test-1==test2)
                                str=str.substr(0,cut);
                        cout<<endl<<++test2<<"   Binary decoded : "<<str;
```

```
                                cout<<"\tDecimal value : "<<deci;
                                up:
                                for(int i=0;i<8;i++)
                                {
                                       matchit+=str[i];
                                       int index=check(matchit);
                                       if(index!=-1)
                                              {
                                       outFile<<data[index].c;    dlen++;
                           //if found then add into output using the index returned
from funcion
                                                     matchit.clear();
                                              }
                                }
                                if(isfound(test2+1))                    //check
that is it need to add the avoided character to the string
                                       {
                                              test2++;
                                       str="00011010";
                           //binary string of 26
                                       goto up;
                           //jump to the finding logic with new string
                                       }
                           }
                           cout<<"\nSize : "<<test<<"  Decoded Size : "<<test2;
                             //to check all the characters decoded or not
}
```

OUTPUT:

originalmin.txt - Notepad

File Edit Format View Help

```
It's not only writers who can benefit from this free online tool. If you're
a programmer who's working on a project where blocks of text are needed,
this tool can be a great way to get that. It's a good way to test your
programming and that the tool being created is working well.

Above are a few examples of how the random paragraph generator can be
beneficial. The best way to see if this random paragraph picker will be
useful for your intended purposes is to give it a try. Generate a number of
paragraphs to see if they are beneficial to your current project.

If you do find this paragraph tool useful, please do us a favor and let us
know how you're using it. It's greatly beneficial for us to know the
different ways this tool is being used so we can improve it with updates.
This is especially true since there are times when the generators we create
get used in completely unanticipated ways from when we initially created
them. If you have the time, please send us a quick note on what you'd like
to see changed or added to make it better in the future.s
```

Compressedmin.txt - Notepad

File Edit Format View Help

```
×ÄÛ9XW!qêÿìö4_<|~ ¬2?ÙâWþ3*¡¨×ðº™Y'KZ,ÙcSØÒmžÒÚÿ4+„-[ø{¬J}-•áŽÛ
„«8ßo¼i‘þÌª|ðI’h®pön2ÍF¾&Ù&•]žÍÆQx]HS¥-l³üÐžvdg↟‹2ª OüÐ½\ãìþÏimš¸CRz~°•
[²U’xý¶lêexF½2,YçuaÔ¬è³ÔMÎR–Ï˜“ãó÷þa¨NÅ‰=áìÜelÿ†GûyÝXu
+:,õ¯ßi©ïál&#oÄ@ò”º§ù¼Î½VÞÏìÊ4ý»?„˜·j5–Æ³Œ“^±&¥xu+:,õ™FÛ?á‘w¬Iñùûÿ02‹©
^…WÌ:Z¶;óRz~¿…Ôƒº<ÿ;2?ÙÔ¬è³ÔLª€¿
iÔíÐ’y6ô¤ó²lµÊôz]HL¬FÿÍæ£_lÑ\ä.|~~ÿÌ)DlÊ-
r½2,ïþxÕófí™ìÊ¨Ø“ÿ4·Ùº=eóÃötµnÏáïò":îqïPölcßNÿÌ!qŠ†oü½™¬•f~Ç³Ø¾høÖr–g¬½
\ã4x#o³ü/«:ƒÇ„ryŸ¿ýN>Ïfíž-X{ÃÖŸÌ!qz¹ÇÙ‘vj5ü.¤›vdYŸ±ëN çlÛçb6HœGï´r
±•ÃØÎRvCö™”m²øÏ4}”¤÷ÛìÊ,›Lþ    <Æ§ødYâ1
õl
```

outputmin.txt - Notepad

File Edit Format View Help

```
It's not only writers who can benefit from this free online tool. If you're
a programmer who's working on a project where blocks of text are needed,
this tool can be a great way to get that. It's a good way to test your
programming and that the tool being created is working well.

Above are a few examples of how the random paragraph generator can be
beneficial. The best way to see if this random paragraph picker will be
useful for your intended purposes is to give it a try. Generate a number of
paragraphs to see if they are beneficial to your current project.

If you do find this paragraph tool useful, please do us a favor and let us
know how you're using it. It's greatly beneficial for us to know the
different ways this tool is being used so we can improve it with updates.
This is especially true since there are times when the generators we create
get used in completely unanticipated ways from when we initially created
them. If you have the time, please send us a quick note on what you'd like
to see changed or added to make it better in the future.
```

```
557   Binary decoded : 10011011 Decimal value : 155
558   Binary decoded : 01001100 Decimal value : 76
559   Binary decoded : 11111110 Decimal value : 254
560   Binary decoded : 00001001 Decimal value : 9
561   Binary decoded : 00111100 Decimal value : 60
562   Binary decoded : 11000110 Decimal value : 198
563   Binary decoded : 10100111 Decimal value : 167
564   Binary decoded : 11111000 Decimal value : 248
565   Binary decoded : 01100100 Decimal value : 100
566   Binary decoded : 01011001 Decimal value : 89
567   Binary decoded : 11100010 Decimal value : 226
568   Binary decoded : 00110001 Decimal value : 49
569   Binary decoded : 00001010 Decimal value : 10
570   Binary decoded : 11110101 Decimal value : 245
571   Binary decoded : 011011   Decimal value : 108
Size : 571  Decoded Size : 571
--------------------------------
Process exited after 2.485 seconds with return value 0
Press any key to continue . . .
```

3. Extended-Huffman algorithm

```
#include <bits/stdc++.h>
using namespace std;

struct bind
{
                        string c;
                        float val;
                        string code;
                        int left,right;
}data[400];

int first=0,n=0;
vector<int> v,avoid;

int isfound(int value)
{
                        for(int i=0;i<avoid.size();i++)
                        {
                          if(avoid[i]==value)
                                  return true;
                        }
                        return false;
}
int check(string matchit)
{
                        for(int i=0;i<v.size();i++)
                        {
                          if(matchit==data[v[i]].code)
                                  return v[i];
                        }
                        return -1;
```

```
}
void printlist()
{
                        cout<<"\nList : ";
                        for(int i=0;i<n;i++)
                        {
                          if(i==first)
                                 cout<<" | ";
                          cout<<data[i].val<<" ";
                        }
}

string getbinary(int n)   //converting int to 8 bit binary
{
                        string str;
    for (int i = 7; i >= 0; i--) {
        int k = n >> i;
        if (k & 1)
                        str+="1";
        else
            str+="0";
    }
    return str;
}

void assign(int index,string s)
{
                        data[index].code=s+data[index].code;
                        if(data[index].left!=-1)
                          assign(data[index].left,s);
                        if(data[index].right!=-1)
                          assign(data[index].right,s);
}
```

```
void huffman()
{
                        if(data[first+1].val==0 || (data[first].val==0 &&
first!=0))
                          return;
                        for(int i=0;i<n-1;i++)             //sorting
                        {
                          for(int j=0;j<n-i-1;j++)
                          {
                                 if(data[j].val > data[j+1].val)
                                        swap(data[j],data[j+1]);
                          }
                        }
                        printlist();
                        float temp;
                        cout<<endl<<data[first].val<<" + "<<data[first+1].val<<" =
";
                        temp=data[first+1].val+data[first].val; //taking sum of 2
smallest number
                        cout<<temp;
                        data[n].left=first;
                        data[n].right=first+1;
                        data[n].val=temp;                         //making a new
entry in array of struct
                        cout<<"\nAppended "<<data[n].val<<" left index :
"<<data[n].left<<" right index : "<<data[n].right<<endl;
                        assign(first,"0");                //assign 0s to left
part children
                        assign(first+1,"1");        //assign 1s to right part
children
                        first+=2;
                        n++;
                        huffman();
}

int main()
```

```
{

                        ifstream inFile;
                        ofstream OutFile;
                        std::map<string,int> trace;  //dict type data structure
                        char text;
                        string s;
                        int count=0;
    inFile.open("originalext.txt");
    if (!inFile)
                        {
        cout << "Unable to open file";
        exit(1); // terminate with error
    }
    inFile>>std::noskipws;

                        while (inFile >> text)    //read char and save to text
                        {
                          s+=text;
                          count++;
                          if(count%2==0)
                          {
                        trace[s]++;     //key=text for occcurence count (increment
for every occurence)
                          s.clear();
                          }
    }
    if(count%2==1)
                        {
                                trace[s]++;
                                count=count/2+1;
                          }
                        else
                          count/=2;
```

```
    int i=0;
    map<string, int>::iterator itr;
    cout << " KEY\tOCCURENCE\n";
    for (itr = trace.begin(); itr != trace.end(); ++itr) {
       /* cout <<" "<< itr->first
            <<"\t   " << itr->second << '\n'; */
        data[i].left=-1;
                          data[i].right=-1;
                          data[i].c=itr->first;                    //assigning it
to structure defined for process
                          data[i++].val=itr->second;
                          n++;
                          cout <<" "<< data[i-1].c
            <<"\t   " << data[i-1].val << '\n';
    }
    inFile.close();
                        printlist();
                        huffman();
                        std::map<string,string> findcode;               //for
mapping purpose
                        int k=0;
                        cout<<"\nCharacter\tCodeword\n";
                        for(int i=n;i>=0;i--)
                        {
                          if(data[i].left==-1 && data[i].right==-1)
                          {
                                 findcode[data[i].c]=data[i].code;

                        cout<<data[i].c<<"\t\t"<<findcode[data[i].c]<<endl;
                          //saving code in index as character so we can use later
for retreival purpose
                                 v.push_back(i);
                        //saving indexes of leaf nodes so we can easily traverse
using vector
                          }
                        }
```

```
ofstream outFile;
outFile.open("Compressedext.txt");
inFile.open("originalext.txt");
string str;
int test=0;
int h=0;
bool entry=false;
s.clear();
while(inFile>>text)
{
  h++;
  if(h%2!=0)
  {
        s+=text;
  }
  else
  {
  s+=text;
  str+=findcode[s];
//add corrosponding code of character to string(binary)
  s.clear()    ;
  if(str.length()>8)
//if length exceeds 8 then start processing data
  {
        int sum=0;
        string w=str.substr(0,8);
//take first 8 characters of binary
        cout<<endl<<++test<<"   Chopped string : "<<w;
        for(int i=0;i<8;i++)
        {
              if(w[i]=='1')
                    sum+=pow(2,7-i);
        }
```

```
                cout<<"\t Decimal : "<<sum;
                if(sum==26)
        //avoid adding this element because it will cause end of
          file later on
                {
                        avoid.push_back(test);
                        cout<<"  Skipped";
                }
                else
                        outFile<<(char)sum;
                str=str.substr(8);
          }
          }
        }
        int sum=0;   //for remaining last characters which is less
than 8
        cout<<endl<<++test<<"   Chopped string : "<<str;
        int cut=str.length();
        for(i=0;i<cut;i++)
          {
                if(str[i]=='1')
                        sum+=pow(2,7-i);
          }
          cout<<"\t Decimal : "<<sum;
        outFile<<(char)sum;
        inFile.close();
        outFile.close();
        inFile.open("compressedext.txt");
        outFile.open("outputext.txt");
        string matchit;
        int dlen=0;
        int test2=0;
        while(inFile>>text)
        {
```

```
                        int deci=(int)text;
                        if(deci<0)
                              deci+=256;
                        string str=getbinary(deci);
                        if(dlen==count-1)
                              str=str.substr(0,cut);
                        cout<<endl<<++test2<<"   Binary decoded : "<<str;
                        cout<<"\tDecimal value : "<<deci;
                        up:
                        for(int i=0;i<8;i++)
                        {
                              matchit+=str[i];
                              int index=check(matchit);
                              if(index!=-1)
                                    {
                                    outFile<<data[index].c;    dlen++;
                                          matchit.clear();
                                    }
                        }
                        if(isfound(test2+1))
      //check that is it need to add the avoided character to the string
                              {
                                    test2++;
                                    str="00011010";
                                    //binary string of 26
                                    goto up;
                              //jump to the finding logic with new string
                              }
                  }
                  cout<<"\nSize : "<<count<<"  Decoded Size : "<<dlen;
                    //to check all the characters decoded or not
}
```

OUTPUT:

```
C:\CODING\SEM 6 IT\DCDR\Practical 2 Huffman\extendedhuffman.exe
KEY     OCCURENCE


          1

I         1
  G       1
  I       1
  T       1
  a       6
  b       5
  c       4
  d       2
  e       1
  f       4
  g       4
  h       2
  i       3
  l       1
  n       3
  o       6
  p       6
  q       1
  s       2
  t       20
  u       3
  w       13
  y       5
 'r       2
 's       3
 ,        2
 .        1
 .
          1
 .        4
 Ab       1
 If       2
 It       2
 Th       1
 a        7
 ag       3
 al       2
 am       1
 an       5
 ap       1
 ar       5
 at       4
 av       2
 ay       3
```

```
 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
16 16 16 16 16 16 16 16 16 16 16 16 16 17 18 19 20 20 20 22 24 24 24 24 25
229 + 304 = 533
Appended 533 left index : 354 right index : 355

Character       Codeword
e               0001
 t              11001
 w              00101
o               110100
d               110001
s               101111
ra              101110
in              101101
th              010010
te              010001
se              010000
re              001111
er              001110
a               001101
y               1111011
us              1111010
t               1111001
or              1111000
is              1110111
ea              1110110
be              1110101
 p              1110100
 o              1110011
 a              1110010
ou              1110001
n               1110000
it              1101111
he              1101110
en              1101101
ar              1101100
an              1101011
 y              1101010
 b              1100001
to              0110101
r               0110100
pr              0110011
pl              0110010
om              0110001
ne              0110000
nd              0101111
l               0101110
hi              0101101
f               0101100
et              0101011
```