EXPERIMENT REPORT OF DATA STRUCTURE

Experiment 7

NAME	: ABID ALI
STUDENT ID	: 2019380141
DATE	: 06/18/2021
E-Mail	: abiduu354@gmail.com

Problem Description:

Design a Huffman encoding/decoding system, which can encode and decode the messages to be transmitted. When you build a Huffman tree, you need put the little weight on the left and the large on the right. The right child tree is encoded as 1, the left is encoded as 0.

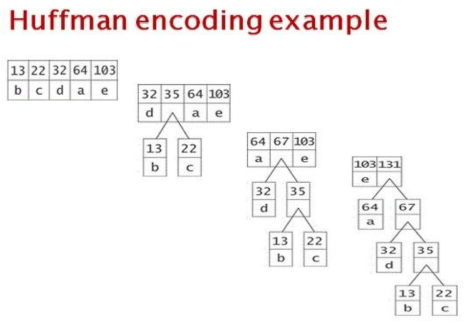
Goal:

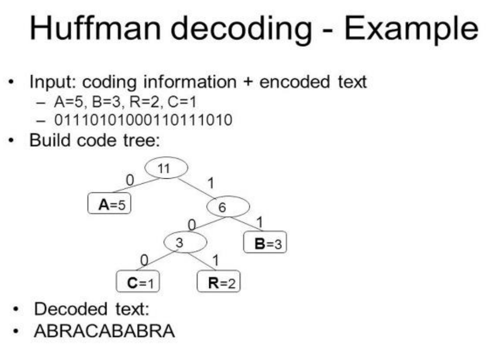
This assignment is designed to help understanding and mastering the use of C++ header files, basic C++ syntax, and stream-based I/O. By successfully completing this assignment we could master the following outcomes:

- **Understanding the of basic C++ syntax.**
- **By using basic C++ we learn implementation of control structures.**
- **Knowing the knowledge about Huffman Encoder/Decoder.**
- **How to Traverse the Huffman Tree**
- **Understanding the construction, the parent node**

General Design:

1. Data structure



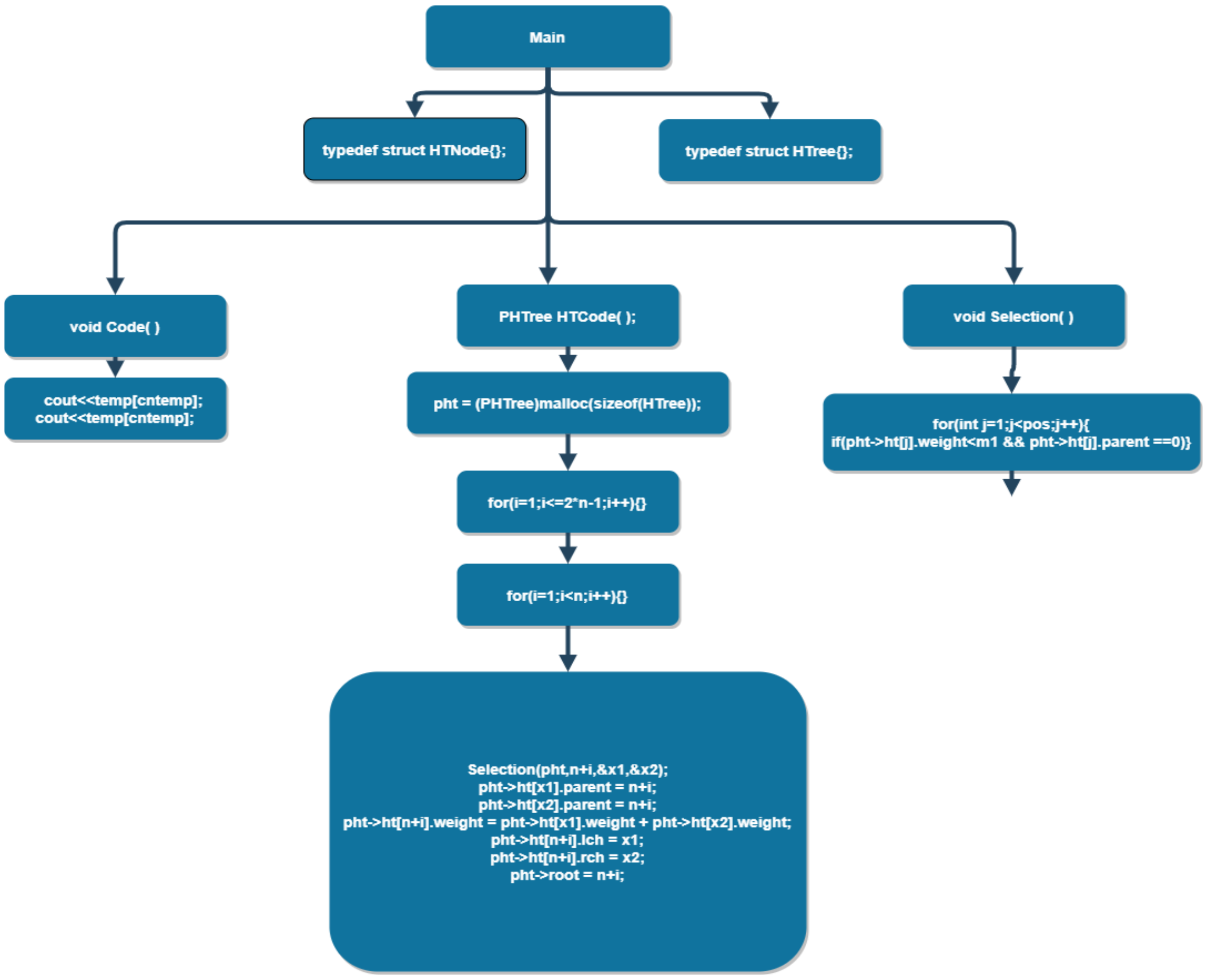

Huffman decoding - Example

- Input: coding information + encoded text
  - A=5, B=3, R=2, C=1
  - 01110101000110111010
- Build code tree:
- Decoded text:
- ABRACABABRA

---

2. Basic operations needed to complete:

Structure Chart:


```
Main

typedef struct HTNode{};
typedef struct HTree{};

void Code( )
cout<<temp[cntemp];
cout<<temp[cntemp];

PHTree HTCode( );
pht = (PHTree)malloc(sizeof(HTree));
for(i=1;i<=2*n-1;i++){}
for(i=1;i<n;i++){}
Selection(pht,n+i,&x1,&x2);
pht->ht[x1].parent = n+i;
pht->ht[x2].parent = n+i;
pht->ht[n+i].weight = pht->ht[x1].weight + pht->ht[x2].weight;
pht->ht[n+i].lch = x1;
pht->ht[n+i].rch = x2;
pht->root = n+i;

void Selection( )
for(int j=1;j<pos;j++){
if(pht->ht[j].weight<m1 && pht->ht[j].parent ==0)}
```

Implementation:

In computer science and information theory, a Huffman code is a particular type of optimal prefix code that is commonly used for lossless data compression. The process of finding or using such a code proceeds by means of Huffman coding. The output from Huffman's algorithm can be viewed as a variable-length code table for encoding a source symbol (such as a character in a file). The algorithm derives this table from the estimated probability or frequency of occurrence (*weight*) for each possible value of the source symbol. As in other entropy encoding methods, more common symbols are generally represented using fewer bits than less common symbols. Huffman's method can be efficiently implemented, finding a code in time linear to the number of input weights if these weights are sorted.[2] However, although optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods - it is replaced with arithmetic coding[3] or asymmetric numeral systems[4] if better compression ratio is required.

Source: https://en.wikipedia.org/wiki/Huffman_coding

There are two major parts of encoding:

1. Building a Huffman Tree from input characters.

2. Traverse the Huffman Tree and assign code to each character

Input: *N* unique characters along with their frequencies of occurrence. Enter a positive integer *n* representing the size of the character set (n <=100) as well as n characters and n weights (weight is a positive integer. It has a larger value, then it has the greater probability of occurrence); the user needs to enter messages whose length is smaller or equal to 100.

I used the PHTNode structure to store a single node of the Huffman tree and the node is stored in a sequential table. By modifying the node information, it represents the interrelationship between nodes. The Huffman tree is established from the bottom down according to the approach of greed, and then the character-by-character search is in the tree, and the path represents the encoding of the character pair.

1. The design of the node structure is as follows, it contains the weight, weight of the character as WEIGHT, its parent in the position of "parent" in the order table, its left child tree in the order table of the position "lch" and its right child tree in the order table position "rch".
   The code struct is as follows:
   ```
   typedef struct HTNode{
           int weight;
           int parent;
           int lch;
           int rch;
   }HTNode,*PHTNode;
   ```

```
typedef struct HTree{
        HTNode ht[MAXNODE];
        int root;
}HTree,*PHTree;
```

2. The Design of the Huffman tree is as follows, which contains a long HTNode-type array of MAXNODE and the node position of an integer root representing the root of that number.

   ```
   typedef struct HTree{
           HTNode ht[MAXNODE];
           int root;
   }HTree,*PHTree;
   ```

3. After the node data is entered in the main function, the Huffman tree is established through the HTCode function, and each time in the order table, two fatherless nodes with the smallest weights are found in the order table, and a new node is established at the end of the sequential table as the parent node of the two nodes. Until the construction of a new node built n-1 (number of nodes is n) ends.
   The function code is as follows:

   ```
   PHTree HTCode(int n,int *w)
   {

           PHTree pht;
           int i,x1=0,x2=0;
           pht = (PHTree)malloc(sizeof(HTree));

           for(i=1;i<=2*n-1;i++){
                   pht->ht[i].lch = 0;
                   pht->ht[i].rch = 0;
                   pht->ht[i].parent = 0;
                   if(i<=n)pht->ht[i].weight = w[i-1];
                   else pht->ht[i].weight = 0;
           }
           for(i=1;i<n;i++){
                   Selection(pht,n+i,&x1,&x2);
                   pht->ht[x1].parent = n+i;
                   pht->ht[x2].parent = n+i;
                   pht->ht[n+i].weight = pht->ht[x1].weight + pht->ht[x2].weight;
                   pht->ht[n+i].lch = x1;
                   pht->ht[n+i].rch = x2;
                   pht->root = n+i;
           }
   ```

```
        return pht;
}
```

4. The task of selecting two small weights in the HTCode function is done by the function "Selection", which is simpler and posts the code as follows:

```
void Selection(PHTree pht,int pos,int *x1,int *x2)
{

        int m1=MAXINT,m2=MAXINT;
        for(int j=1;j<pos;j++){
                if(pht->ht[j].weight<m1 && pht->ht[j].parent ==0){
                        m2=m1;
                        *x2=*x1;
                        m1 = pht->ht[j].weight;
                        *x1=j;
                }
                else if(pht->ht[j].weight<m2 && pht->ht[j].parent == 0){
                        m2=pht->ht[j].weight;
                        *x2=j;
                }
        }
}
```

Once the numbers are established, the coding process begins In the first n nodes of the order table, that is, the leaves of the Huffman tree, to find the required characters, it has been looking up to find the parent node, and generate the reverse code, it finds the reverse code and output it after the reverse sequence. The decoding can be achieved by searching for the leaves by encoding. Here is the function Code:

```
void Code(PHTree pht,char *taget,int n,int *w,char *chs)
{

        int le=strlen(taget);
        for (int i = 0; i <le ; ++i)
        {
                int j,cntemp=0;
                char temp[100];
                for ( j = 0; j < n; ++j)
                {
                        if(chs[j]==taget[i])break;
                }
                int c=j+1,f;
```

```
            for(f = pht->ht[j+1].parent; f!=0 ;c=f,f=pht->ht[f].parent){
                    if(pht->ht[f].lch == c)temp[cntemp++]='0';
                    else temp[cntemp++]='1';
            }
            while(--cntemp)
    cout<<temp[cntemp];
            cout<<temp[cntemp];
    }
    cout<<endl;

}
```

All The functions above are then called in the main function and user is asked to enter the required input and the later the output is shown on the screen;
Here is the main function:

```
int main()
{

    int n;
    int w[101];
    char chs[101];
    char taget[101];

      cout<<"Input  ."<<endl;
    cin>>n;

    for (int i = 0; i < n; ++i)cin>>chs[i];

    for(int i=0; i<n; ++i)cin>>w[i];

    getchar();
    gets(taget);
    PHTree pht=HTCode(n,w);

    cout<<endl;
    cout<<"Output ."<<endl;
    Code(pht,taget,n,w,chs);
    cout<<taget<<endl;

    getchar();
    getchar();
```

```
                return 0;
        }
```

Source:

1) https://www.techiedelight.com/huffman-coding/
2) https://www.programmersought.com/article/93256208662/

Test Description and Results:

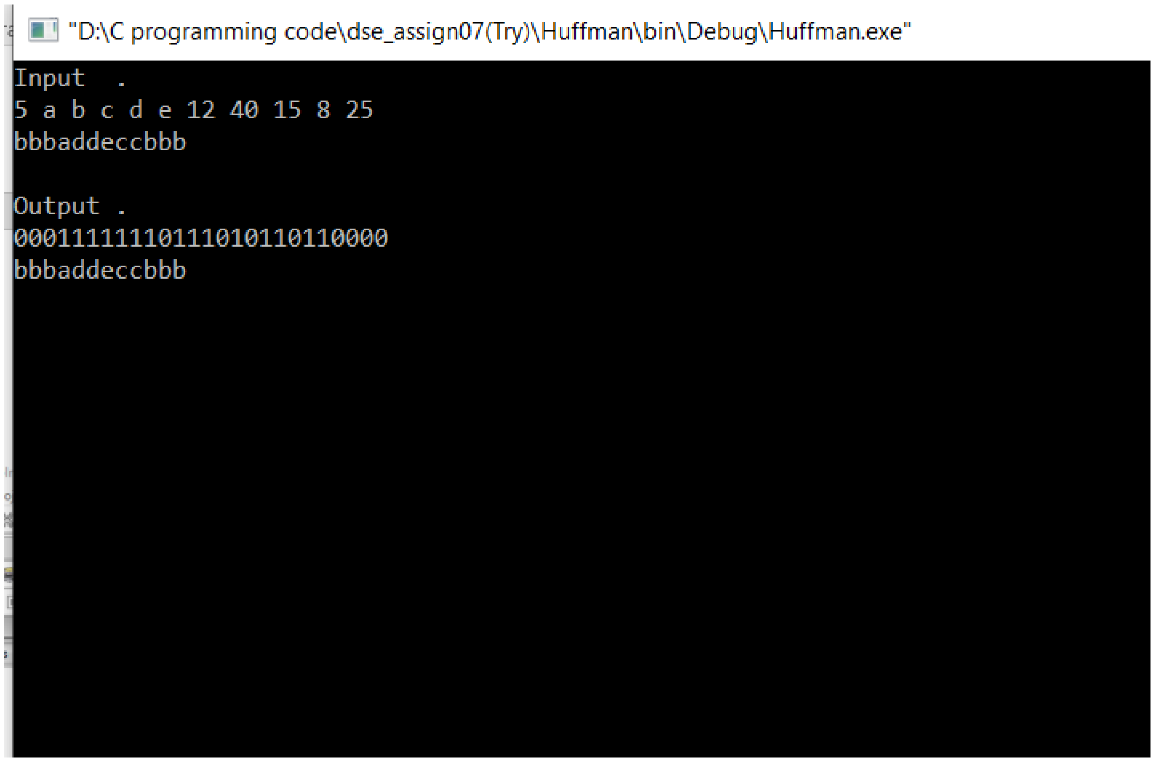
The output of the system is the encoding binary code taking a line; and the corresponding messages taking a line; Finally, a carriage return is output.

Input sample

5 a b c d e 12 40 15 8 25
bbbaddeccbbb

Output sample

0001111111011101011011000 0
bbbaddeccbbb



"D:\C programming code\dse_assign07(Try)\Huffman\bin\Debug\Huffman.exe"

```
Input  .
5 a b c d e 12 40 15 8 25
bbbaddeccbbb

Output .
00011111110111010110110000
bbbaddeccbbb
```

Epilogue:

While doing this assignment I had many bugs which had influence on program working and many troubles with compiling the program, because of syntax .Fixing those bugs was a challenge for me.I followed teacher practical lecture note,use those idea to solve the problem.Internet was a big help for this program.This program helped me understand the concept of **Huffman Encoder/Decoder**.

Attachments:

1)Huffman.cpp

2)Huffman.txt

3) Assignment 6_ABID ALI_2019380141.pdf

Acknowledgement:

I complete this assignment by myself by using online videos and different books discussing about Algorithm and Data structure. Also,used my knowledge about Object Oriented Programming to solve this problem.It was very useful and helpful for me to increase knowledge for solving complex problem.

Remarks and Grade (by the instructor)

Instructor Signature:
Grading Date: