北京師範大學
BEIJING NORMAL UNIVERSITY

一、实验要求

1. 上机之前应做好充分准备，认真思考所需的上机题目，提高上机效率。

2. 独立上机输入和调试自己所编的程序，切忌抄袭、拷贝他人程序。

3. 上机结束后，整理出实验报告。书写报告时，重点放在实验的方法、思路以及总结反思上，以达到巩固课堂学习、提高动手能力的目的。

二、实验内容

霍夫曼编码问题

三、实验步骤（写出问题分析或者算法思路）

霍夫曼树创建过程中不同之处在于通过小根堆寻找最小和次小，找到之后将最小和次小从小根堆中删除，插入最小和次小之和，霍夫曼树的结构定义和小根堆的结构定义和课本相同。然后是得到密码的过程，在霍夫曼树中从叶节点向上回溯指导根节点，获得逆序密码，然后将密码颠倒顺序得到正确密码。解密过

程是简单的 BM 模式匹配过程。

四、程序清单（源程序代码等）

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define leafNumber 5

#define totalNumber 9 //totalNumber=2*leafNumber-1

#define heapSize 20

#define keylenth 10

#define max_keynumber 10


//Huffman tree defination

typedef struct

{

    char data;

    int weight;

    int parent,lchild,rchild;

}HTNode;

typedef struct

{

    HTNode elem[totalNumber];

    int num,root;

}HFTree;

//minheap defination

typedef int HElemType;

typedef struct

{

    HElemType elem[heapSize];

    int curSize;
```

```
}minHeap;
//decipher keys defination
typedef struct
{
    char chararter;
    char decipher_string[keylenth];
}key;
typedef struct
{
    key elem[max_keynumber];
    int cur_key_num;
}Keys;
//function of minHeap
void ShiftDown(minHeap& H,int i,int m)
{
    HElemType temp=H.elem[i];
    for(int j=2*i+1;j<=m;j=2*j+1)
    {
        if(j<m&&H.elem[j]>H.elem[j+1]) j++;
        if(temp<=H.elem[j])break;
        else
        {
            H.elem[i]=H.elem[j];
            i=j;
        }
    }
    H.elem[i]=temp;

}
```

```
void CreateMinHeap(minHeap& H,int arr[],int n)
{
    for(int i=0;i<n;i++) H.elem[i]=arr[i];
    H.curSize=n;
    for(int i=(H.curSize-2)/2;i>=0;i--)
        ShiftDown(H,i,H.curSize-1);
}

void ShiftUp(minHeap& H,int start)
{
    HElemType temp=H.elem[start];
    int j=start,i=(j-1)/2;
    while(j>0)
    {
        if(H.elem[i]<=temp) break;
        else
        {
            H.elem[j]=H.elem[i];
            j=i;
            i=(i-1)/2;
        }
    }
    H.elem[j]=temp;
}

bool Insert(minHeap& H,HElemType x)
{
    if(H.curSize==heapSize) return false;
    H.elem[H.curSize]=x;
    ShiftUp(H,H.curSize);
```

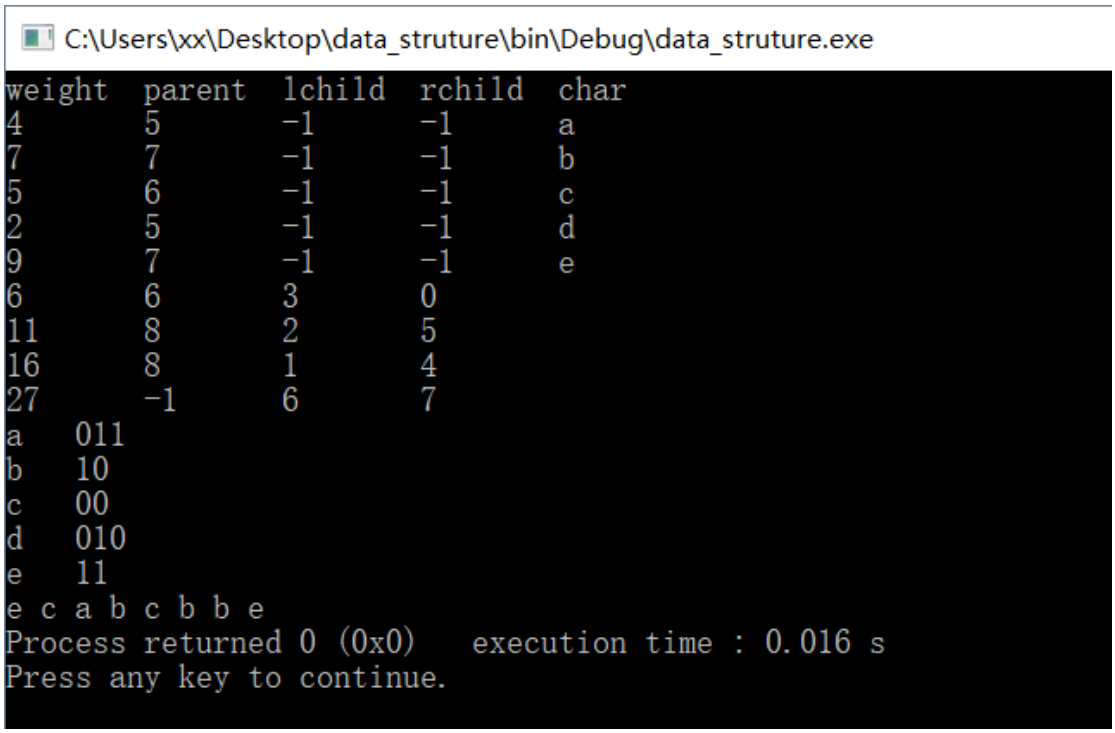
```
    H.curSize++;
    return true;
}



bool Remove(minHeap& H,HElemType& x)
{
    if(H.curSize==0) return false;
    x= H.elem[0];
    H.elem[0]=H.elem[H.curSize-1];
    H.curSize--;
    ShiftDown(H,0,H.curSize-1);
    return true;
}
void CreateHFTree(HFTree& HT,char value[],int fr[],int n)
{
    int i,min1_index,min2_index,min1_weight,min2_weight;
    for(i=0;i<n;i++) {HT.elem[i].data=value[i];HT.elem[i].weight=fr[i];}
    for(i=0;i<2*n-1;i++)
        HT.elem[i].parent=HT.elem[i].lchild=HT.elem[i].rchild=-1;
    minHeap H;
    CreateMinHeap(H,fr,n);
    for(i=n;i<2*n-1;i++)
    {
        Remove(H,min1_weight);
        Remove(H,min2_weight);
        Insert(H,min1_weight+min2_weight);
        for(int j=0;j<i;j++)
        {
            if(HT.elem[j].weight==min1_weight) min1_index=j;
```

```
                if(HT.elem[j].weight==min2_weight) min2_index=j;
            }
            HT.elem[min1_index].parent=HT.elem[min2_index].parent=i;
            HT.elem[i].lchild=min1_index;
            HT.elem[i].rchild=min2_index;
            HT.elem[i].weight=min1_weight+min2_weight;
            HT.elem[i].data=' ';
        }
        HT.num=n;
        HT.root=2*n-2;
    }
    void Getkeys(Keys& K,HFTree& H,int keynumber)
    {   K.cur_key_num=keynumber;
        for(int i=0;i<keynumber;i++)
        {
            int j=i,k,n=0;
            int key_length=0;
            K.elem[i].chararter=H.elem[i].data;
            while(H.elem[j].parent!=-1)
            {
                k=H.elem[j].parent;
                if(H.elem[k].lchild==j)
{K.elem[i].decipher_string[key_length]='0';key_length++;}
                if(H.elem[k].rchild==j)
{K.elem[i].decipher_string[key_length]='1';key_length++;}
                j=k;
            }
            K.elem[i].decipher_string[key_length]='\0';
            //先得到逆序密码，再改变顺序得到正常密码
            char temp_str[keylenth];
```

```
            strcpy(temp_str,K.elem[i].decipher_string);
            for(int m=strlen(temp_str)-1;m>=0;m--)
            {
                K.elem[i].decipher_string[n]=temp_str[m];
                n++;
            }
        }
    }
    void Decrypt(Keys& mykeys,char ciphertext[])
    {
        int i=0;
        while(i<strlen(ciphertext))
        {
            for(int q=0;q<mykeys.cur_key_num;q++)
            {
                int n=0;
                for(int m=i;n<strlen(mykeys.elem[q].decipher_string);m++,n++)
                {
                    if(ciphertext[m]!=mykeys.elem[q].decipher_string[n]) break;
                }
                if(n==strlen(mykeys.elem[q].decipher_string))
                {
                    printf("%c ",mykeys.elem[q].chararter);
                    i=i+strlen(mykeys.elem[q].decipher_string);
                    break;
                }

            }
        }
    }
```

```
int main()
{
    int weight[5]={4,7,5,2,9};
    char code[5]={'a','b','c','d','e'};
    char *ciphertext="11000111000101011";
    HFTree testtree;
    CreateHFTree(testtree,code,weight,5);
    printf("weight\tparent\tlchild\trchild\tchar\n");
    for(int i=0;i<9;i++)
    {

printf("%d\t%d\t%d\t%d\t%c\n",testtree.elem[i].weight,testtree.elem[i].parent,testtr
ee.elem[i].lchild,testtree.elem[i].rchild,testtree.elem[i].data);
    }
    Keys mykeys;
    Getkeys(mykeys,testtree,testtree.num);
    for(int i=0;i<mykeys.cur_key_num;i++)

printf("%c   %s\n",mykeys.elem[i].chararter,mykeys.elem[i].decipher_string);
    Decrypt(mykeys,ciphertext);
}
```

五、运行结果（程序运行时的结果说明或运行截图等）



六、总结（实验中遇到的问题、取得的经验、感想等）

小根堆中可以直接存放霍夫曼树的结点，这样的话得到密码可能会给简便一些，不需要逆序，但是小根堆的一系列操作和创建霍夫曼树的操作需要重写，可能有些难度。