**2019秋季学期**

**数据结构实验报告**

**实验二 树实验**

**班级：\_\_\_\_计实验19\_\_\_\_\_\_\_\_**

**学号：\_\_\_\_\_19101130122\_\_\_\_\_\_\_**

**姓名：\_ 陈旻杰\_\_\_\_\_\_\_\_\_\_**

|  |
| --- |
| **评语：**  **日期：** |

**实验目的：**

熟悉非线性结构的特点 , 掌握非线性结构的存储方式及各种操作的实现方法，同时对自顶向下的程序设计方法、应用程序界面的设计、非线性结构的文件存储方法等方面的辑程技术进行训练。

1、树的遍历

**问题描述**：

二叉树一般有三种遍历方式，即先序遍历、中序遍历、后序遍历，相应得到对应的三个遍历序列，分别称作先序序列、中序序列、后序序列。当给定我们先序序列和中序序列或者后序序列和中序序列，就可以确定这棵二叉树。

现给定我们某二叉树先序序列和中序序列，要求你给出该二叉树的后序序列。

**基本要求:**

构建二叉树，然后对二叉树进行后续遍历。

**Input**

输入包括两行，每行一个字符串（字符串长度最大为30），第一行字符串为二叉树的先序序列，第二行字符串为二叉树的中序序列，二叉树结点数据为一个字符。

**Output**

输出包括一行，为二叉树的后续序列。

**Sample Input**

**ABDEC**

**DBEAC**

**Sample Output**

**DEBCA**

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

分析：由先序遍历和中序遍历确定树，然后对该树后序遍历

解决：由先序和中序唯一确定出这棵树，然后在对确定出的这个树进行后序遍历

方法：先序遍历的访问顺序是根节点左孩子右孩子，中序遍历的特点是左孩子根节点和右孩子，由先序遍历可以确定根节点一定是第一个，由中序遍历可以确定的是根节点的左子树。

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

typedef struct node

{

char str;

node\* l;

node\* r;

}\*shu;

shu solve(string s1,string s2)；

void dfs(shu point)；

**基本操作的实现：**（对各基本操作实现的描述）

#include <iostream>

#include <algorithm>

#include <string>

#include <cstdio>

using namespace std;

const int maxn = 1000010;

typedef struct node

{

char str;

node\* l;

node\* r;

}\*shu;

int x = 0;

shu solve(string s1,string s2)

{

int m = s2.length();

if(m == 1)

{

shu point = new node();

point->str = s2[0];

point->l = NULL;

point->r = NULL;

return point;

}

string x1, x2;//构建左右子树

int i;

for(i = 0; i<= m;i++)

{

if(s2[i] != s1[x])

{ //找根节点

x1 += s2[i];

}

else break;

}

i++;

for(;i<m;i++)

{

x2 += s2[i];

}

shu point = new node();

point->l = NULL;

point->r = NULL;

point->str=s1[x];

if(x1.length() != 0)

{

x++;

point->l = solve(s1,x1);

}

if(x2.length() != 0)

{

x++;

point->r = solve(s1,x2);

}

return point;

}

void dfs(shu point)

{ //后序遍历

if(point->l != NULL) dfs(point->l);

if(point->r != NULL) dfs(point->r);

printf("%c",point->str);//每个叶子节点都可以看做一棵子树的根节点

}

int main()

{

string s1,s2;

cin >> s1 >> s2;

shu root = NULL;

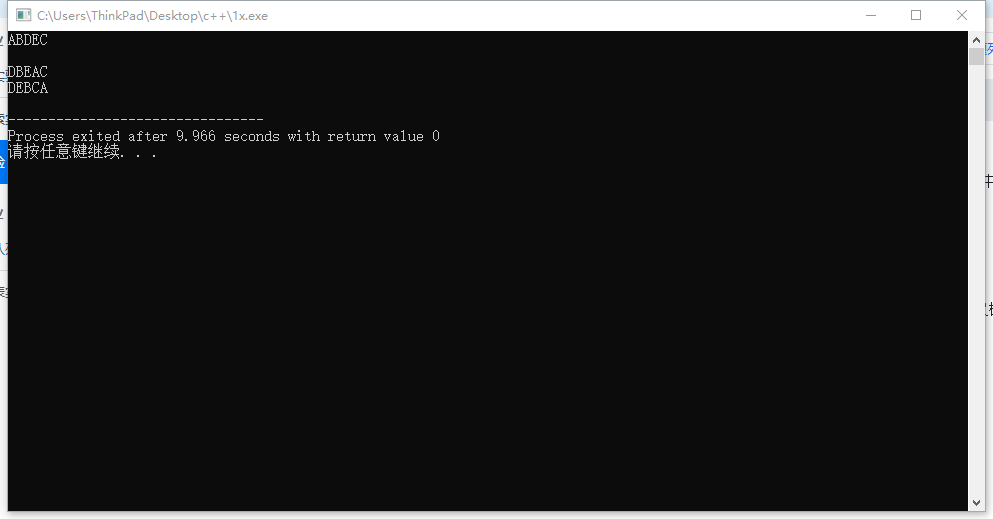
root = solve(s1,s2);

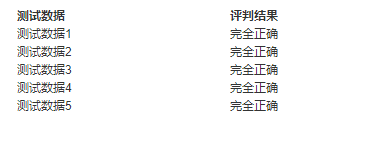
dfs(root);

return 0;

}

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）





**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法）

1.本题的难点主要在于建树而非遍历，熟悉指针的操作，然后

2、哈夫曼树

**问题描述**：

利用哈夫曼编码进行信息通讯可以大大提高信道利用率，缩短信息传输时间，降低传输成本。你的问题是：对于给定的一组字符在文中出现的概率，给出这组字符的哈夫曼编码。

**基本要求：**

采用动态数组存储哈夫曼树。

**Input**

输入包括两行，第一行为一个整数n，表示文中出现的字符个数，第二行为每个字符在文中出现的次数，各数之间由空格分隔。

**Output**

输出包括n行，为按给出出现次数顺序的各字符哈夫曼编码。

**Sample Input**

**8**

**5 29 7 8 14 23 3 11**

**Sample Output**

**0110**

**10**

**1110**

**1111**

**110**

**00**

**0111**

**010**

**需求分析：**（包括对问题的理解，解决问题的策略、方法描述）

分析：根据已知的树创出哈夫曼树，再按照给出的哈夫曼树读出到达这个叶子节点的路径。

解决：每次从从前往后选出最小的两个数，合成一个新数，这个新的数作为俩原来两个数据的双亲结点，原来的两个数据先出现的作为左子树，后出现的作为右子树，数据不可重复使用，新生成的数据放在首面，按照这个规律，创造出这棵树，然后再对创造出来的树进行读取，得到我们需要的哈夫曼编码。

**系统设计：**（包括数据结构定义、抽象出基本操作描述、主程序模块处理过程描述）

typedef struct

{

int weight;

int parent;

int lchild;

int rchild;

int flag;

}HTnode;

typedef struct

{

char cd[20];

int start;

}Hcode;

void creatHT(HTnode ht[],int n0)

//操作结果：创造出一颗哈夫曼树

void creatHcode(HTnode ht[],Hcode hcd[],int n0)

//初始条件：存在一颗哈夫曼树

//操作结果：读出哈夫曼编码

**基本操作的实现：**（对各基本操作实现的描述）

void creatHT(HTnode ht[],int n0)

{

int i,j,k,lnode,rnode,m,flag1,flag2;

m=2\*n0-1;

for(i=0;i<m;i++)

{

ht[i].parent=-1;

ht[i].lchild=-1;

ht[i].rchild=-1;

ht[i].flag=-1;

}

int min1,min2;

for(i=n0;i<m;i++)

{

min1=min2=999;

lnode=rnode=-1;

for(k=0;k<=i-1;k++)

{

if(ht[k].parent==-1&&ht[k].flag==-1)

{

if(ht[k].weight<min1)

{

min1=ht[k].weight;

flag1=k;

}

}

}

ht[flag1].flag=0;

for(k=0;k<=i-1;k++)

{

if(ht[k].parent==-1&&ht[k].flag==-1)

{

if(ht[k].weight<min2)

{

min2=ht[k].weight;

flag2=k;

}

}

}

ht[flag2].flag=0;

if(flag2<flag1)

{

lnode=flag2;

rnode=flag1;

}

else

{

lnode=flag1;

rnode=flag2;

}

ht[lnode].parent=i;

ht[rnode].parent=i;

ht[i].weight=ht[lnode].weight+ht[rnode].weight;

ht[i].lchild=lnode;

ht[i].rchild=rnode;

}

}

void creatHcode(HTnode ht[],Hcode hcd[],int n0)

{

int i,f,c;

Hcode hc;

for(i=0;i<n0;i++)

{

hc.start=n0;

c=i;

f=ht[i].parent;

while(f!=-1)

{

if(ht[f].lchild==c)

{

hc.cd[hc.start--]='0';

}

else

hc.cd[hc.start--]='1';

c=f;

f=ht[f].parent;

}

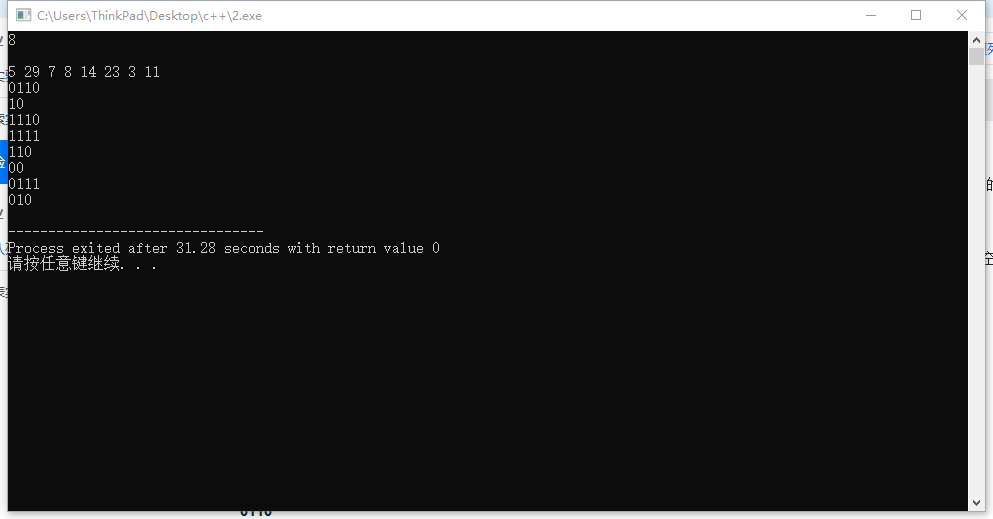
hc.start++;

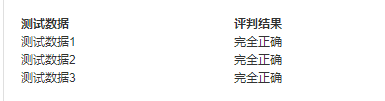
hcd[i]=hc;

}

}

**测试结果：**（输入的测试数据及运行结果、正确性、在线测试情况）





**调试分析：**（包括调试过程中对原设计的修改，以及遇到的问题和解决的方法）

1.建树操作需要熟练，没有怎么写过哈夫曼树