

**算法与数据结构课程设计报 告**

**系 （院）：**  计算机科学学院

**专业班级：** 软工11601班

**姓 名：** 洪胜兵

**学 号：** 201603346

**序 号：** 11

**指导教师：** 周云才

目录

1. **课程设计目的..........................................................3**
2. **课程设计方案..........................................................4**

**2.1单链表基本操作的设计及应用设计....................4**

**2.2 栈基本操作的设计............................................5**

**2.3矩阵基本操作的设计及应用设计.......................6**

**2.4树基本操作的设计.............................................6**

**2.5图基本操作的设计.............................................7**

1. **实现过程..................................................................8**

**3.1单链表的实现过程.............................................8**

**3.2栈的实现过程....................................................9**

**3.3矩阵的实现过程................................................9**

**3.4树的实现过程....................................................9**

**3.5图的实现过程及拓扑排序................................10**

1. **实现代码................................................................10**

4.1**单链表的应用代码..........................................10**

4.2**矩阵的应用代码..............................................13**

4.3**图基本操作及拓扑排序的代码........................15**

**五．测试.........................................................................18**

**六．结论.........................................................................23**

**七．收获与难点.............................................................23**

1. **课程设计目的**

1．提高数据抽象能力。根据实际问题，能利用数据结构理论课中所学到的知识选择合适的逻辑结构以及存储结构，并设计出有效解决问题的算法。

2．提高程序设计和调试能力。学生通过上机练习，验证自己设计的算法的正确性。学会有效利用基本调试方法，迅速找出程序代码中的错误并且修改。

3．初步了解开发过程中问题分析、整体设计、程序编码、测试等基本方法和技能。

1. **课程设计方案**

**流程图：**

数据结构(主菜单)

单链表

栈

数组

二叉树

图

创建

显示

查找

删除

插入

进栈

出栈

取栈顶

创建

显示

矩阵乘法

创建

遍历

叶子结点

深度

结点双亲

结点兄弟

……

无向图

有向图

无向网

有向网

遍历

拓扑排序

### 单链表的基本操作及应用

表一：单链表的基本操作涉及的主要函数列表

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 函数名(包含参数) |
| 1 | 创建链表 | CreateList\_L(L,n) |
| 2 | 插入结点 | ListInsert\_L(L,i,n) |
| 3 | 删除结点 | ListDelete\_L( L, i ) |
| 4 | 查找结点 | ListSearch\_L（L,i,e); |
| 5 | 显示链表 | PrintList( L ); |
| 6 | 输入联系人的信息 | AddrListInsert\_L(L,i,e) |
| 7 | 输出联系人的信息 | PrintAddrList(L) |

首先实现单链表的基本操作1.创建链表2.插入结点3.创建4.删除5.查找6.显示。最后利用一个单链表实现通讯录设计，其中可以输入姓名，电话号码，插入名单，显示名单，同时还可以通过好友姓名查找到好友的电话号码。

### 2.2.栈的基本操作

表二：栈的基本操作涉及的主要函数列表

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 函数名 |
| 1 | 初始化栈 | InitStack(Stack &S) |
| 2 | 进栈 | Push(Stack &S, ElemType e) |
| 3 | 出栈 | Pop(Stack &S, ElemType &e) |
| 4 | 取栈顶元素 | GetTop(Stack S, ElemType &e) |

实现栈的基本操作 ①进栈 ②出栈 ③取栈顶元素

### 2.3.数组的基本操作及应用

表三：数组的基本操作涉及的主要函数列表

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 函数名 |
| 1 | 以三元组方式创建矩阵 | CreateSMatrix(TSMatrix &M) |
| 2 | 显示矩阵 | PrintSMatrix(TSMatrix M) |
| 3 | 矩阵乘法 | MultSMatrix(M, N, &Q) |

首先完成数组的基本操作①创建矩阵②显示矩阵，再实现矩阵乘法(应用)。

### 2.4.二叉树的基本操作

表四：二叉树的基本操作涉及的主要函数列表

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 函数名 |
| 1 | 创建二叉树 | CreateBiTree(BiTree &T) |
| 2 | 先序遍历二叉树 | PreOrderTraverse(BiTree &T,int (\*Visit)(char e)) |
| 3 | 中序遍历二叉树 | InOrderTraverse(BiTree &T,int (\*Visit)(char e)) |
| 4 | 后序遍历二叉树 | PostOrderTraverse(BiTree &T,int (\*Visit)(char e)) |
| 5 | 计算叶子结点数 | LeafCount(BiTree T) |
| 6 | 树的深度 | BiTreeDepth(BiTree T) |
| 7 | 求左兄弟 | LeftSibling(BiTree T, BiTNode \*e |
| 8 | 求右兄弟 | RightSibling(BiTree T, BiTNode \*e) |
| 9 | 求双亲 | SearchParent(BiTree T, BiTNode \*e) |

完成二叉树的基本操作：1.创建二叉树2.遍历二叉树（三种顺序）3.计算叶子节点数4.求树的深度5.求兄弟6.求双亲；

### 2.5.图的基本操作

表五：图的基本操作涉及的主要函数列表

|  |  |  |
| --- | --- | --- |
| 序号 | 名称 | 函数名 |
| 1 | 有向图的创建 | CreateDG(ALGraph &G) |
| 2 | 无向图的创建 | CreateUDG(ALraph &G) |
| 3 | 有向网的创建 | CreateDN(ALGraph &G) |
| 4 | 无向网的创建 | CreateUDN(ALGraph &G) |
| 5 | 选择图的类型 | CreateGraph(ALGraph &G) |
| 6 | 初始化队列 | InitQueue(LinkQueue &Q) |
| 7 | 增加队列里的元素 | EnQueue(LinkQueue &Q, int e) |
| 8 | 删除队列里的元素 | DeQueue(LinkQueue &Q, int &e) |
| 9 | 判断队列是否为空 | QueueEmpty(LinkQueue Q) |
| 10 | 找第一个邻结点 | FirstAdjVex(ALGraph G, int v) |
| 11 | 找下一个邻结点 | NextAdjVex(ALGraph G, int v, int w) |
| 12 | 深度遍历 | DFSTraverse(ALGraph &G) |
| 13 | 广度遍历 | BFSTraverse(ALGraph &G) |
| 14 | 拓扑排序 | TopoSort(ALGraph &g) |

图的基本操作：1.创建图（UDG，UDN，DG，DN）2.遍历图（深度优先，广度优先）3.拓扑排序；

**三．实现过程**

1.单链表的基本操作及应用

①定义一个结点的结构体

结构体中包括：好友姓名，定义一个string型数组来存储好友姓名。

好友电话号，定义一个int型数组来存储好友电话号。

定义一个指针，指针指向该结构体。

②定义一个创建链表函数CreatList()。

CreatList函数中，首先创建一个头结点，然后每来一个数据元素，就申请一个空间。用一个指针指向该空间挂在前一个结点的尾部。

③定义一个插入结点的函数ListInsert()。

ListInsert函数中，当位置参数传入函数体中，首先判断传进的位置参数是否合理，判断合理之后，去要找该位置，断开链子，将结点插入其中。

④定义一个删除结点的函数ListDelete()。

ListDelete函数中，与插入函数类似，首先判断传入的位置参数是否合理，找到该结点，删除结点之后，将前一个结点的指针域指向下一个结点的空间。

⑤定义一个查找好友的函数AddrListSearch\_L()。

AddrListSearch\_L函数中，首先输入好友的姓名，通过strcmp()函数找到好友名单，可以输出好友的电话号。

⑥显示函数PrintAddrList()。

PrintAddrList函数需要传入一个指向链表的头指针，判断头指针不为空时，输出链表里所有的数据元素。

⑦显示链表菜单函数LinkListMenu()。

LinkListMenu函数的菜单项是通过switch()语句实现选择功能。

2.栈的基本操作

①定义一个栈的结构体

结构体中包括：栈底指针\*base，栈顶指针\*top。

②定义一个进栈函数

用Push()函数，将元素压入栈低，S.top++；，栈顶指针加一。

③定义一个出栈函数

将后进入的元素弹出，e = \*(S.top-1); S.top--;栈顶指针减一。

④定义一个取栈顶元素

保存栈顶元素，并把值带回主函数，top指针不变。

3.矩阵的基本操作及应用

1. 创建矩阵

创建的矩阵包括非零元行、列下标及其数据。通过循环输入函数存储于三元组Triple data[maxsize+1];数组中。

1. 显示矩阵

循环输出矩阵的非零元行、列下标及其数据。

1. 矩阵乘法（应用）

为了简化矩阵乘法运算，首先找到矩阵中的非零元，存储于三元组数组rpos[MAXRL+1]中，然后再将数组中对应行、列的非零元相乘、再相加，将和及对应的行、列下标存储于三元组W中，通过打印三元组函数PrintSMatrix（），将矩阵相乘结果输出。得到非零元所在的行、列下标及其数据。

4.二叉树的基本操作

* 1. 创建二叉树

构造一个链式存储结构体，其中包括数据域、左、右孩子指针。设计一个以先序方式创建二叉树，通过递归调用CreateBiTree()，T->data=ch;生成根结点CreateBiTree(T->lchild);构造左子树CreateBiTree(T->rchild); 构造右子树，完成构造二叉树。

* 1. 遍历二叉树（先序、中序、后序）

先序遍历二叉树PreOrderTraverse()，通过设计Visit()显示函数，递归InOrderTraverse(T->lchild,Visit)遍历左子树，递归InOrderTraverse(T->rchild,Visit)遍历右子树。访问顺序：根、左、右。

中序遍历二叉树InOrderTraverse()，递归PostOrderTraverse(T->lchild,Visit）遍历左子树，递归PostOrderTraverse(T->rchild,Visit）遍历右子树。访问顺序：左、根、右。

后序遍历二叉树PostOrderTraverse ()，递归PostOrderTraverse (T->lchild,Visit）遍历左子树，递归PostOrderTraverse (T->rchild,Visit）遍历右子树。访问顺序：左、右、根。

* 1. 计算叶子结点个数及树的深度

叶子节点个数：递归调用CountLeaf()，当结点的左右孩子都无，加一，累计于count。

树的深度：递归调用Depth()，输出左子树、右子树中最大值。

* 1. 查找指定结点的双亲

递归调用SearchParent()函数，遍历根结点的左右子树，找到指定结点，输出双亲。

* 1. 查找指定结点的兄弟

递归调用找左兄弟LeftSibling()函数，遍历结点的左右子树，看双亲，输出结果。

递归调用找右兄弟RightSibling ()函数，遍历结点的左右子树，看双亲，输出结果。

5.图的基本操作

①创建无向图

先构造邻接矩阵结构体AdjMatrix[]，与图结构体，包括邻接矩阵、存储被访问的顶点向量。

再设计定位函数LocateVex()，构造AdjMatrix arcs[i][j];邻接矩阵存储图的顶点之间的关系, 通过i=LocateVex(G,v1); j=LocateVex(G,v2); G.arcs[i][j].adj=w; G.arcs[j][i] = G.arcs[i][j];构造邻接矩阵，创建无向网。

②创建有向图

通过i=LocateVex(G,v1); j=LocateVex(G,v2); G.arcs[i][j].adj=1;构造邻接矩阵，创建有向图。

③创建无向网

通过i=LocateVex(G,v1); j=LocateVex(G,v2); G.arcs[i][j].adj=w; G.arcs[j][i] = G.arcs[i][j];构造邻接矩阵，创建无向网。

④创建有向网

通过i=LocateVex(G,v1); j=LocateVex(G,v2); G.arcs[i][j].adj=w;构造邻接矩阵，创建 有向网。

⑤DFS和BFS遍历

深度优先遍历图：构造FirstAdjVex()、NextAdjVex()两个函数，构造访问标记数组visited[]，递归调用DFS()，直至图中所有结点被访问来实现深度遍历图。

广度优先遍历图：先要实现队列，构造结点结构体，其中包括数据、指向下一个结点的指针。还构造了一个包括有队头指针\*front、队尾指针\*rear的结构体。同时实现进队列，出队列等功能。调用FirstAdjVex()、NextAdjVex()两个函数，找到未被访问的结点，再将访问过的元素进队列，按照先进先出原则，出队列，广度优先遍历图。

⑥拓扑排序

先把每个顶点的入度求出来，用数组保存，把入度为零的节点压栈，弹出顶点时，删除相对应的边。

**四．实现代码**

4.1单链表的基本操作及应用

**（1）创建联系人；**

**Status CreateAddrList\_L(AddrLinkList &L, int n)**

**{**

**AddrLinkList p, q;**

**L = new ALNode;**

**L->next = NULL;**

**q = L;**

**cout << "请输入联系人\n";**

**cout << "姓名" << "\t" << "号码" << endl;**

**for (int i = 0; i < n; i++)**

**{**

**p = new ALNode;**

**cin >> p->data.name>>p->data.numble;**

**p->next = NULL; //p是单独的一个个空间，q将其链接成链表。**

**q->next = p;**

**q = p;**

**}**

**return OK;**

**}**

**（2）显示联系人；**

**void PrintAddrList(AddrLinkList L)**

**{**

**if (L == 0)**

**cout << "通讯录为空";**

**AddrLinkList p = L->next;**

**cout << "姓名\t号码\n";**

**while (p)**

**{**

**cout << p->data.name << "\t " << p->data.numble << endl;**

**p = p->next;**

**}**

**}**

**（3）插入联系人；**

**Status AddrListInsert\_L(AddrLinkList & L, int i, LinkMan e)**

**{**

**AddrLinkList p = L;**

**for (int j = 0; j < i - 1; j++)**

**{**

**if (p)**

**p = p->next;**

**}**

**if (p)**

**{**

**AddrLinkList q = new ALNode;**

**q->data = e;**

**q->next = p->next;**

**p->next = q;**

**return OK;**

**}**

**return ERROR;**

**}**

**（4）删除联系人；**

**Status AddrListDelete\_L(AddrLinkList & L, int i)**

**{**

**AddrLinkList p = L;**

**int j = 0;**

**while (p->next&&j < i - 1)**

**{**

**p = p->next;**

**j++;**

**}**

**if (!(p->next) || j > i - 1)**

**return ERROR;**

**AddrLinkList q = p->next;**

**p->next = q->next;**

**free(q);**

**return OK;**

**}**

**（5）查找联系人；**

**LinkMan AddrListSearch\_L(AddrLinkList & L, int i, LinkMan &e)**

**{**

**AddrLinkList p = L->next;**

**int j = 1;**

**while (p&&j<i)**

**{**

**p = p->next;**

**j++;**

**}**

**if (!p || j > i)**

**exit(OVERFLOW);**

**e = p->data;**

**return e;**

**}**

2.矩阵的应用

TSMatrix MultSMatrix(TSMatrix M, TSMatrix N, TSMatrix &Q)

{

TSMatrix m,n,w;

m = CreateSMatrix();

n = CreateSMatrix();

int row;

//num[]数组用来求出矩阵中每一行的非零元个数

int num[maxsize];

for (row = 1; row <= m.mu; row++)

{

//初始化 各行非零元个数都为零

num[row] = 0;

}

for (int t = 0; t <= m.tu; ++t)

{

//计算各行非零元个数

if(t==0) continue;

++num[m.data[t].i];

}

m.rpos[1] = 1;

//为了确定矩阵中每一行的第一个非零元在三元组表的位置

//此位置用数组rpos[MAXRL+1]里面的元素进行存储 MAXRL全称 MAXROWLOCATE

for (row = 2; row <= m.mu; row++)

{

m.rpos[row] = m.rpos[row - 1] + num[row - 1];

}

int a[20];

for (row = 1; row <= n.mu; row++)

{

//初始化 各行非零元个数都为零

a[row] = 0;

}

for (int z = 0; z <= n.tu; ++z)

{

//计算各行非零元个数

if(z==0) continue;

++a[n.data[z].i];

}

n.rpos[1] = 1;

for (row = 2; row <= n.mu; row++)

{

n.rpos[row] = n.rpos[row - 1] + a[row - 1];

}

////////////////////////////////////以上完成建表//////////////////下面进行矩阵乘法//////////////////

int arow;

int ccol;

w.mu = m.mu;

w.nu = n.nu;

w.tu = 0;

//w是非零矩阵

if ((m.tu\*n.tu) != 0)

{

for (arow = 1; arow <= m.mu; arow++)

{

//累加器ctemp 用来放矩阵中行和列各元素对应相乘 在相加的结果

//累加器清零

int ctemp[maxsize] = { 0 };

int tp;

if (arow < m.mu)

tp = m.rpos[arow + 1];

else

{

tp = m.tu + 1;

}

//当前行中每一个非零元的位置

for (int p = m.rpos[arow]; p < tp; p++)

{

//找到对应元在n中的的行号

int brow = m.data[p].j;

int t;

if (brow < n.mu)

t = n.rpos[brow + 1];

else

{

t = n.tu + 1;

}

for (int q = n.rpos[brow]; q < t; q++)

{

//乘积元素在q中的列号

ccol = n.data[q].j;

//实现非零元素对应相乘 在相加

ctemp[ccol] += m.data[p].val\*n.data[q].val;

}

//求得w中第crow(=arow)行的非零元

}

for (ccol = 1; ccol <= w.nu; ++ccol)

if (ctemp[ccol] != 0)

{

++w.tu;

w.data[w.tu].i = arow;

w.data[w.tu].j = ccol;

w.data[w.tu].val = ctemp[ccol];

}

}

}

return w;

}

3.二叉树的基本操作

①求二叉树深度

int BiTreeDepth(BiTree T)

{

if (T == 0)

return 0;

if (BiTreeDepth(T->lchild) >= BiTreeDepth(T->rchild)) //判断左右子树深度的大小；

return BiTreeDepth(T->lchild) + 1;

return BiTreeDepth(T->rchild) + 1;

}

②叶子节点总数

int LeafCount(BiTree T)

{

if (T == 0)

return 0;

else if (LeafCount(T->lchild) == 0 && LeafCount(T->rchild) == 0) //当二叉树的左右子树都为空时就只有一个节点。

return 1;

else

return LeafCount(T->lchild) + LeafCount(T->rchild);

}

③查找双亲

void SearchParent(BiTree T, BiTNode \*e)

{

if (T)

{

if (T->data == e->data)

cout<<"无双亲!\n";

//判断节点e为T->lchild;

else if (T->lchild&&T->lchild->data == e->data)

{

cout << "双亲为： " << T->data << endl;

}

else

{ //判断节点e为T->rchild;

if (T->rchild&&T->rchild->data == e->data)

{

cout << "双亲为： " << T->data << endl;

}

else

{

SearchParent(T->lchild, e);

SearchParent(T->rchild, e);

}

}

}

else

return;

}

④查找双亲

void RightSibling(BiTree T, BiTNode \* e)

{

if (T->lchild) //左孩子存在才有可能存在左兄弟；

{

if (T->lchild->data == e->data)

{

if (T->rchild)

cout << "右兄弟为" << T->rchild->data << endl;

else

cout<<"无右兄弟\n";

}

else

RightSibling(T->lchild, e);

}

else

cout<<"无右兄弟！\n";

}

void LeftSibling(BiTree T, BiTNode \* e)

{

if (T->rchild) //右孩子存在才有可能存在左兄弟；

{

if (T->rchild->data == e->data)

{

if (T->lchild)

cout << "左兄弟为" << T->rchild->data << endl;

else

cout << "无左兄弟\n";

}

else

LeftSibling(T->rchild, e);

}

else

cout << "无左兄弟！\n";

}

4.图的基本操作

①DFS深度优先遍历图G

void DFS(ALGraph &G, int v)//从第v个顶点出发递归的深度优先遍历图G

{

int w;

G.visited[v] = 1;//访问过的结点为1

Visit(G.vers[v]); //访问第V个结点

for(w = FirstAdjVex(G,v) ; w >= 0; w = NextAdjVex(G,v,w))

if(!G.visited[w])

DFS(G,w); //对V的尚未访问的邻结点W递归调用DFS。

}

void DFSTraverse(ALGraph &G)

{

for(int i = 0; i < G.vexnum; i++)

G.visited[i] = 0;

for(int j = 0; j < G.vexnum; j++)

{

if(!G.visited[j])

DFS(G, j);

}

}

②广度遍历图G

void BFSTraverse(ALGraph &G)//广度遍历

{

int u, w;

for(int i = 0; i < G.vexnum; i++)

G.visited[i] = 0;

LinkQueue Q;

InitQueue(Q);

for(int j = 0; j < G.vexnum; j++)

{

if(!G.visited[j])

{

G.visited[j] = 1;

Visit(G.vers[j]);

EnQueue(Q,j);

while(!QueueEmpty(Q))

{

DeQueue(Q,u);//为何要删除u呢

if(FirstAdjVex(G,u)>=0)

{

w = FirstAdjVex(G,u);

for( ; w >= 0; w = NextAdjVex(G,u,w))

if(!G.visited[w])

{

G.visited[w] = 1;

Visit(G.vers[w]);

EnQueue(Q,w);

}

}

}

}

}

}

③拓扑排序

void findInDegree(ALGraph &g)

{

int i, j;

for (i = 0; i < g.vexnum; i++) {

for (j = 0; j < g.vexnum; j++)

if (g.arcs[i][j].adj)//当两顶点之间存在边时，入度自加

indegree[j] ++;

}

void TopoSort(ALGraph &g)

{

findInDegree(g);

SqStack s;

int i, j, count = 0;

InitStack(s);

for (i = 0; i < g.vexnum; i++)

if (!indegree[i])

push(s, i);//把入度为零的节点压栈

printf("拓扑序列如下:\n");

while (!stackEmpty(s)) {

Pop(s, i);

printf( “%d”,vers[i].a );

count++;

for (j = 0; j < vexnum; j++) {

if (arcs[i][j].adj) {

if (!(--indegree[j]))//删除相对应得边

push(s, j);

}

}

}

if (count < vexnum)

printf( "此图有环存在!\n");

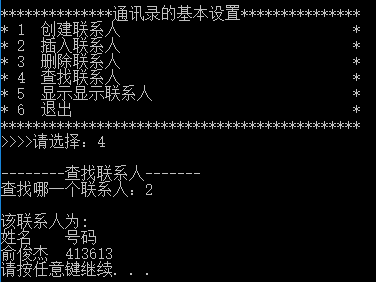
}

1. **测试结果**

**①创建链表**，输入两个联系人名单，把第三个联系人插入第一个位置，显示所有联系人名单，再删除第二个联系人名单，显示所有联系人名单，最后通过输入序号找到联系人信息。显示结果如下：

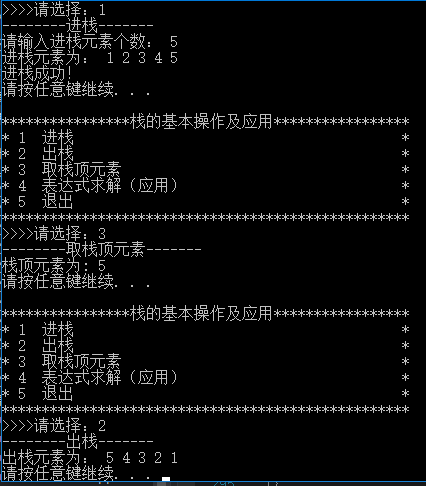
 

显示联系人 插入联系人

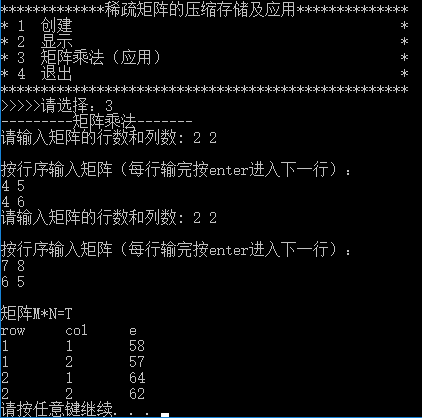
 

删除联系人 查找联系人

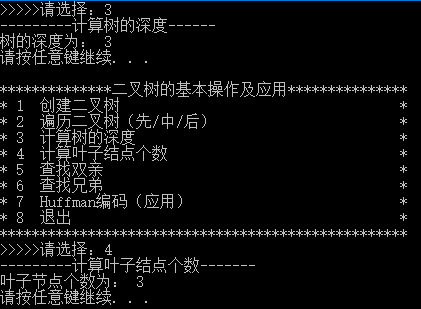
**②栈测试结果，**创建堆栈，输入堆栈个数及数据就进栈，之后可选择出栈和取栈顶元素。



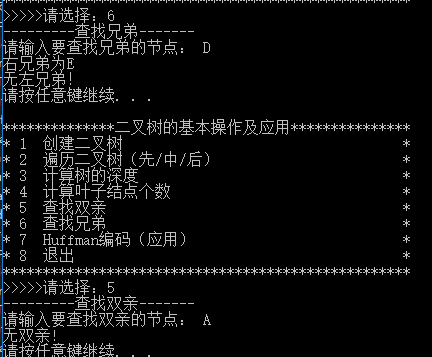
**③矩阵的测试结果，**创建矩阵，输入行数和列数，在行序输入元素就创建成功；显示矩阵它将按行号，列号，数值输出；矩阵相乘要求输入两个矩阵，第一个矩阵的列数要与第二个矩阵的行数相等。



**④树的测试结果，**创建二叉树，按先序顺序创建二叉树，若该节点没有左右子树就输入0，后创建成功；遍历子树有四种方式，先、中、后、层；可以查找叶子节点数和树的深度，查找双亲，查找兄弟包括左兄弟与右兄弟。

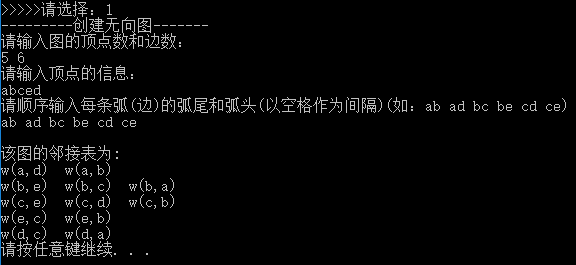


树的创建及遍历求深度及叶子节点个数

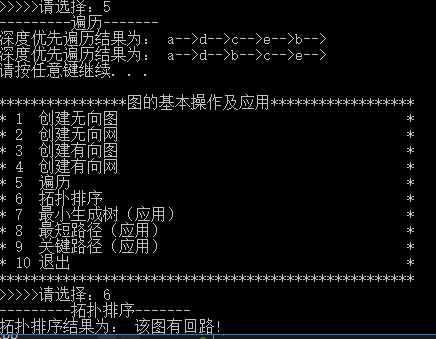


查找兄弟和双亲

**⑤图的测试代码，**创建图，无向图和有向图的创建只需输入顶点数、边数以及顶点信息和边邻接的节点，有向网和无向网还需要输入边的权值，将以邻接表的形示显示图；图的遍历方式有深度优先遍历和广度优先遍历两种方式；图的拓扑排序要求图为邻接表创建的，当为无向图或有向图有回路时它会输出“该图有回路”。



图的创建及邻接表显示



遍历及拓扑排序

**六．结论**

1.菜单中单链表、栈、矩阵、二叉树、图的基本操作的算法实现效果还行，对于创建、插入、删除等功能，可以运用，其算法的正确性可以保证。

2.不足点在于输出结果和显示不太美观，时间复杂度不是最优的，有待进一步完善。

**七．难点与收获**

### 其难点或问题如下：

1. 在链表的插入、删除元素时，要用指针找到其对应的位置，指针的指向有点难得操作；再则在通讯录应用方面，我只是通过查找联系人序号才查到联系人，不能通过查找姓名或者电话号码查找到该联系人，在实现过程中有点困难，但后期我一定可以解决。
2. 在矩阵乘法时，要找到非零元的位置，并存储与数组中，元素指针的位置不太好找；还有最开始我想要用之前创建的数组去实现矩阵的相乘，然后我发现数组的创建需要输入维度dim以及各维的维数bounds，但我很难将用户输入的数据传入到创建数组函数中去，那之后我直接用定义的数组创建矩阵，并实现矩阵的其他操作。
3. 在实现堆栈的操作时，遇到了两个问题，首先在建立堆栈后，在弹出堆栈元素时，只能一个一个的弹出；这个问题我用循环解决了；第二个问题是在弹出堆栈元素之后，再获取栈顶元素就获取不了，之前的栈里的元素全部清空。
4. 在实现二叉树的基本操作时，问题和难点特别多，首先递归的调用特别多，一些地方（比如求二叉树深度，叶子节点个数）的递归调用能够很容易理解并且可以灵活运用。而在求左右兄弟时的递归调用有点不好理解，出错率很高，主要是不知道在何时调用递归函数。再则在实现层序遍历时有用到系统队列，如：queue<BiTree> Q;,不知道里面是什么原理，以至在层序遍历以及后面的图的广度优先遍历时不知如何利用系统的队列，到最后只是运用自己构造的队列。
5. 在图的基本操作里遇到的难点和问题更多。最开始我是用关联矩阵创r建的图，特别的复杂，而且对后面的拓扑排序毫无用处。拓扑排序函数他需要的是用邻接表创建的图，那之后我就用邻接表创建图。在邻接表实现创建的过程中对firstarc和nextarc这两个指针变量不是很理解，不知道怎么运用。后面在做深度优先遍历和广度优先遍历时用到了FirstAdjVex和NextAdjVex函数求邻接顶点的序号，对这两个函数不是很理解。还有个问题，在图的拓扑排序时用到了FindInDegree这个求顶点入度的函数，这个函数的实现部分也很难理解，并且里面再次需要用到辅助的数据结构堆栈。

### **其收获或感触如下**：

* + - 1. 通过自己动手整体的编写老师上课所讲的数据结构的各种基本操作的程序代码，进一步理解了数据结构处理数据的快速与规范，一定程度上了解到了计算机在处理与存储数据的原理。有了一定的数据抽象能力，在解决问题做应用时，有一定的选择合适的数据结构、存储结构及算法的能力。同时，增强了自己的编写程序与调试程序的能力。
      2. 通过这样整体的代码编辑，最大的感触是，自己在编写代码时遇到老师上课所讲的问题，并能自己找到解决方法，将实际操作与平常的理论知识串在了一起，感觉自己明白的更多也更加的透彻，并且能够体验到编程带来的乐趣。
      3. 在编写整个数据结构过程中，让我感觉最好的是在二叉树和图的基本操作中，通过查找资料，不断地调试再调试，最终实现了二叉树里的遍历，查找双亲，兄弟以及图里的遍历拓扑排序等基本操作，并且看到了最后的输出结果样式的展现，让人感觉到了编程的浓厚兴趣。
      4. 最后我想表达的是，这个程序中还一些不尽人意的地方，比如输出的样式，用户需要有更多的输入选择，还有一些未知的bug。总之，我会不断的去优化调整我的程序。我还要感谢本次课设给予我帮助的老师和同学们。

指导老师意见：

成绩: 教师签名：

年 月 日