☆图

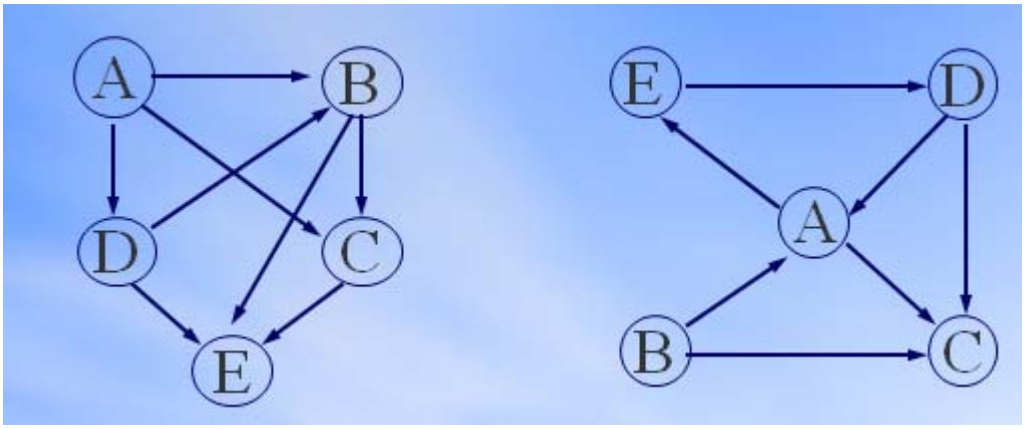
1.对下图求：

(a)邻接矩阵；(b)邻接表；

(c) 列出每一个图中的连通分量；

(d) 那个图示强连通的，哪个是弱连通的，并列出每一

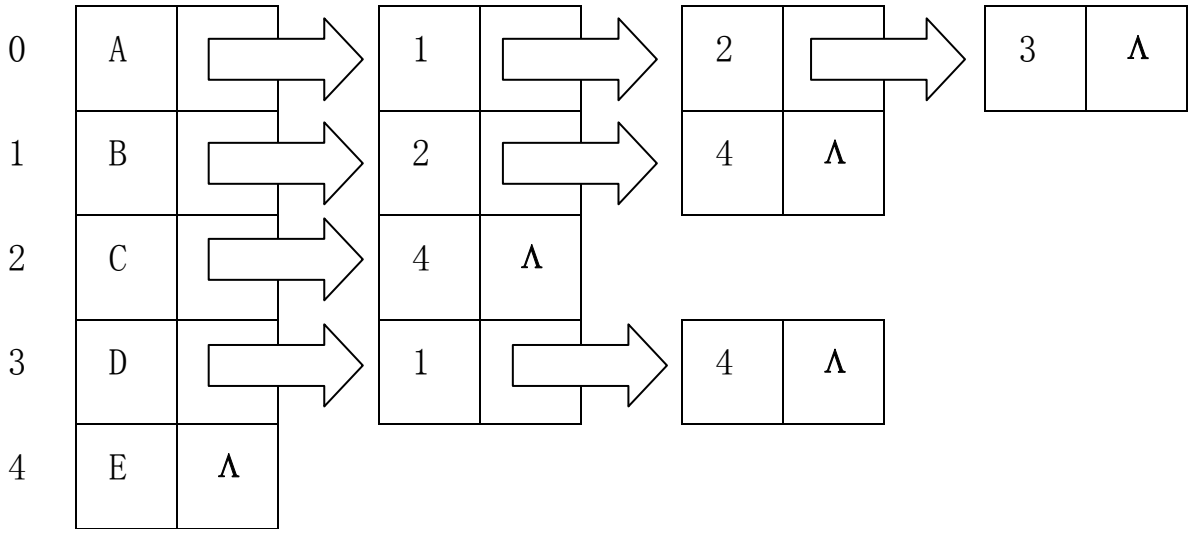
个图的强连通分量。

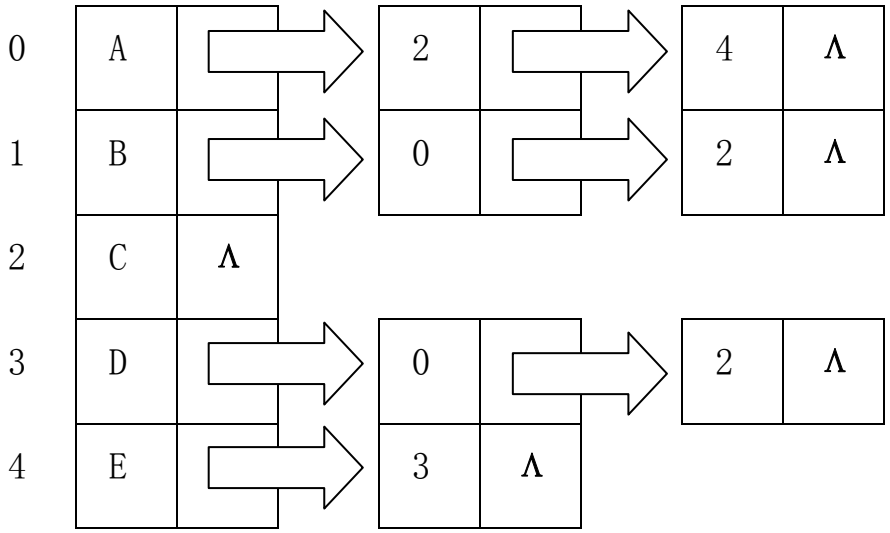


解答：

（a）G1.ARCS=$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ G2.ARCS= $\begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

(b)





上图都不是强连通的，为弱连通图。只有 b 图有强连通分量 A ,E ,D

2.设无向图的存储结构为邻接矩阵，实现如下算法：

(a)添加一个顶点；

(b)删除一个顶点；

(c)添加一条边；

(d)删除一条边。

算法：

```
/*--------------------图的邻接矩阵存储表示----------------------*/
typedef struct ArcCell {
  VRType adj; // VRType 是顶点关系类型。对无权图，用 1 或 0 表示相邻否；
  // 对带权图，则为权值类型。
  InfoType *info; // 该弧相关信息的指针
}ArcCell, AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
typedef struct{
  VertexType vexs[MAX_VERTEX_NUM]; // 顶点向量
  AdjMatrix arcs; // 邻接矩阵
  int vexnum, arcnum; // 图的当前顶点数和弧数
  GraphKind kind; // 图的种类标志
}MGraph;
Status Insert_Vex(MGraph &G, char v)//在邻接矩阵表示的图 G 上插入顶点 v
{
   if(G.vexnum+1)>MAX_VERTEX_NUM return INFEASIBLE;
   G.vexs[++G.vexnum]=v;
   return OK;
}//Insert_Vex
Status Insert_Arc(MGraph &G,char v,char w)//在邻接矩阵表示的图 G 上插入边(v,w)
{
   if((i=LocateVex(G,v))<0) return ERROR;
   if((j=LocateVex(G,w))<0) return ERROR;
   if(i==j) return ERROR;
```

```
    if(!G.arcs[i][j].adj)
    {
        G.arcs[i][j].adj=1;
        G.arcnum++;
    }
    return OK;
}//Insert_Arc
Status Delete_Vex(MGraph &G,char v)//在邻接矩阵表示的图 G 上删除顶点 v
{
    n=G.vexnum;
    if((m=LocateVex(G,v))<0) return ERROR;
    G.vexs[m]<->G.vexs[n]; //将待删除顶点交换到最后一个顶点
    for(i=0;i<n;i++)
    {
        G.arcs[i][m]=G.arcs[i][n];
        G.arcs[m][i]=G.arcs[n][i]; //将边的关系随之交换
    }
    G.arcs[m][m].adj=0;
    G.vexnum--;
    return OK;
}//Delete_Vex
```

分析:如果不把待删除顶点交换到最后一个顶点的话,算法将会比较复杂,而伴随着大量元素的移动,时间复杂度也会大大增加.

```
Status Delete_Arc(MGraph &G,char v,char w)//在邻接矩阵表示的图 G 上删除边(v,w)
{
    if((i=LocateVex(G,v))<0) return ERROR;
    if((j=LocateVex(G,w))<0) return ERROR;
    if(G.arcs[i][j].adj)
    {
```

```
        G.arcs[i][j].adj=0;
        G.arcnum--;
    }
    return OK;
}//Delete_Arc
```

3. 设无向图的存储结构为邻接表，实现第 2 题中的各算法。

算法：

```
//图的邻接表存储表示
typedef struct ArcNode {
  int weight;
  int adjvex; //  该弧所指向的顶点的位置
  struct ArcNode *nextarc; //  指向下一条弧的指针
  InfoType *info; //  该弧相关信息的指针
}ArcNode;
typedef struct VNode {
  VertexType data; //  顶点信息
  ArcNode *firstarc; //  指向第一条依附该顶点的弧
}VNode, AdjList[MAX_VERTEX_NUM];
typedef struct {
  AdjList vertices;
  int vexnum, arcnum; //  图的当前顶点数和弧数
  GraphKind kind; //  图的种类标志
}ALGraph;
/*--------------------插入弧-----------------------*/
int GraphAdd(ALGraph &G){
  int n,k,i,j,w;
  ArcNode *p;
        VertexType v1,v2;
```

```
k=G.vexnum*(G.vexnum-1)-G.arcnum;
printf("请输入要增加的弧数：");
scanf("%d",&n);
while(n>k){
  printf("\n 输入有误,增加的边数不能超过%d，请重新输入！",k);
  printf("\n 请输入有向图的边数：");
  scanf("%d",&n);
}
for(k=0;k<n;k++){
  getchar();
  printf("请输入要增加的弧的起点与终点(用逗号分隔)：");
  scanf("%c,%c",&v1,&v2);
  i=locateALG(G,v1);
  j=locateALG(G,v2);
  if(i<0||j<0||i==j||GraphExist(G,i,j)){
    printf("输入有误,请重新输入\n");
    k--;
    continue;
  }
  printf("请输入第%d 条弧的权值：",k+1);
  scanf("%d",&w);
  p=new ArcNode;
  p->adjvex=j;
  p->weight=w;
  p->nextarc=G.vertices[i].firstarc;
  G.vertices[i].firstarc=p;
  G.arcnum++;
  printf("插入弧成功\n");
}
return 1;
```

```
}

/*--------------------插入顶点-----------------------*/
int NodeAdd(ALGraph &G){
  int i,l,n;
  char c;
  printf("请输入要增加的顶点个数：");
  scanf("%d",&n);
  //增加的定点个数判断
  if(G.vexnum+n>MAX_VERTEX_NUM){
    printf("输入错误，最多有 50 个顶点\n");
    return ERROR;
  }
  for(i=0;i<n;i++){//输入顶点信息
    getchar();
    printf("请输入要增加的顶点:");
    scanf("%c",&c);
    l=locateALG(G,c);
    if(l>=0){
      printf("输入的顶点重复，请重新输入\n");
      i--;
      continue;
    }
    G.vertices[G.vexnum].data=c;
    G.vertices[G.vexnum].firstarc=NULL;
    G.vexnum++;
    printf("增加顶点成功\n");
  }
  return OK;
}
```

```
/*--------------------删除弧--------------------*/
int delArc(ALGraph &G,int i,int j){
  ArcNode *p,*q;
  p=G.vertices[i].firstarc;
  if(p->adjvex==j){
    G.vertices[i].firstarc=p->nextarc;
    free(p);
  }
  else{
    while(p->nextarc&&p->nextarc->adjvex!=j)
      p=p->nextarc;
    if(p){
      q=p->nextarc;
      p->nextarc=q->nextarc;
      free(q);
    }
  }
  G.arcnum--;
  return OK;
}
int GraphDel(ALGraph &G){
  int n,k,i,j;
       VertexType v1,v2;
  printf("请输入要删除的弧数：");
  scanf("%d",&n);
  while(n>G.arcnum){
    printf("删除的弧数不能超过%d，请重新输入\n",G.arcnum);
    printf("请输入要删除的弧数：");
         scanf("%d",&n);
```

```
  }
  for(k=0;k<n;k++){
    getchar();
    printf("请输入要删除的弧的起点与终点(用逗号分隔)：");
    scanf("%c,%c",&v1,&v2);
    i=locateALG(G,v1);
    j=locateALG(G,v2);
    if(i<0||j<0||i==j||(!GraphExist(G,i,j))){
      printf("输入有误,请重新输入\n");
      k--;
      continue;
    }
    delArc(G,i,j);
  }
  return 1;
}

/*---------------------------删除顶点--------------------*/
int NodeDel(ALGraph &G){
  int i,l,n,k;
  char c;
  ArcNode *p;
  printf("请输入要删除的顶点个数：");
  scanf("%d",&n);
       if(n>G.vexnum){
    printf("输入错误\n");
              return ERROR;
  }
  for(k=0;k<n;k++){//输入要删除顶点信息
    getchar();
```

```
printf("请输入要删除的顶点:");
scanf("%c",&c);
l=locateALG(G,c);
if(l<0){
  printf("输入的顶点不存在，请重新输入\n");
  k--;
  continue;
}
for(i=0;i<G.vexnum;i++){
  //删除与此顶点相关的弧
  if(GraphExist(G,i,l)){
    delArc(G,i,l);
  }
  if(GraphExist(G,l,i)){
    delArc(G,l,i);
  }
  //修改必要表结点的顶点的位置值
  p=G.vertices[i].firstarc;
                while(p){
    if(p->adjvex>l){    p->adjvex--;
    }
    p=p->nextarc;
  }
}
//释放空间，顶点 c 后的顶点前移
for(i=l;i<G.vexnum-1;i++){
  G.vertices[i]=G.vertices[i+1];
}
G.vexnum--;
printf("删除顶点成功\n");
```

```
 }
 return OK;
}
```

4.对第一题中的各图，从顶点 A 出发，按深度优先搜索和广度优先搜索算法列出其顶点序列。

答： a)深度优先搜索顶点序列为 A B C E D

广度优先搜索顶点序列为 A B C D E

b)深度优先搜索顶点序列为 A C E D B

广度优先搜索顶点序列为 A C E D B

5.设图 G 为 n 个顶点的无向连通图。证明：

(a)G 至少有 n-1 条边；

(b)所有具有 n-1 条边且 n 个顶点的无向连通图是树。

证明：（a）设 G 中结点为 v1、v2、…、vn 。由连通性，必存在与 v1 相邻的结点，不妨设它为 v2（否则可重新编号），连接 v1 和 v2 ，得边 e1，还是由连通性，在 v3、v4、…vn 中必存在与 v1 或 v2 相邻的结点，不妨设为 v3，将其连接得边 e2，续行此法，vn 必与 v1、v2…、vn-1 中的某个结点相邻，得新边 en-1，由此可见 G 中至少有 n－1 条边。

(b) 由所要证明的问题可知，要证明具有 n 个顶点的无向连通图 G 是树图的充分必要条件是 G 有 n-1 条边。

必要性：由无向连通图的定义可知，具有 n 个顶点的无向连通图至少有 n-1 条边，因为这里是树图，所有它只有 n-1 条边，多于 n-1 条边就会形成环，少于 n-1 条边则不连通。

充分性：若 G 是连通图，且只有 n-1 条边，则 G 必是树图。设 G 是连通的，但不是树图，则 G 中必然有回路存在。我们至少可以从某个回路中去掉一条边，且仍然保证 G 是连通的。则 G 将变成含有 n 个顶点且最多只有 n-2 条边的连通图，这显然是错误的，与无向连通图的定义相违背，故 G 必然是树图。

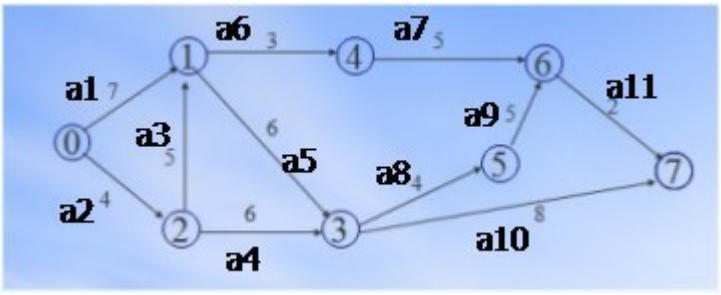
7.N 个顶点的强连通图至少有多少条边？它是什么形状的有向图？

答： 强连通图必须从任何一点出发都可以回到原处,每个节点至少要一条出路(单节点除外)至少有 n 条边,正好可以组成一个环，即形状是环状。

8.证明：对于一个无向图 G=（V，E），若 G 中各顶点的度均大于等于 2，则 G 中必存在回路。

反证法： 对于一个无向图 G=（V，E），若 G 中各顶点的度均大于或等于 2，则 G 中没有回路。此时从某一个顶点出发，应能按拓扑有序的顺序遍历图中所有顶点。但当遍历到该顶点的另一邻接顶点时，又可能回到该顶点，没有回路的假设不成立。

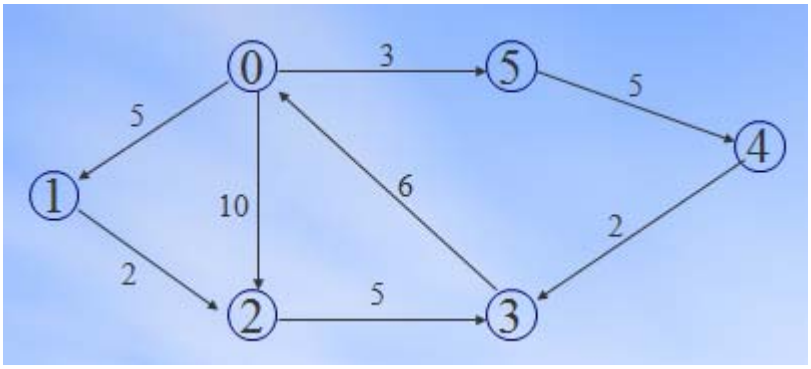
9.对下图所示的 AOE 网，计算各事件（顶点）的 ve 和 vl 值，以及各活动（弧）的 e 和 l 值。



	V0	V1	V2	V3	V4	V5	V6	V7
ve	0	9	4	15	10	19	24	26
vl	0	9	9	15	19	19	24	26
vl-ve	0	0	5	0	9	0	0	0

	a1	a2	a3	a4	a5	a6	a7	a8	a9	a10	a11
E(ai)	0	0	4	4	9	9	10	15	19	15	24
L(ai)	2	5	4	9	9	16	19	15	19	18	24
	2	5	0	5	0	7	9	0	0	3	0

10.利用 Dijkstra 算法，求下图中从顶点 V0 到其余各顶点的最短路径，写出算法过程的每一步状态。



S	D						W
	0	1	2	3	4	5	
{V0}	0	5	10	15	8	3	V5
{V0, V5}	0	5	10	15	8	-	V1
{V0, V1, V5}	0	-	7	12	8	-	V2
{V0, V1, V2, V5}	0	-	-	12	8	-	V4
{V0, V1, V2, V4, V5}	0	-	-	12	-	-	V3

数组 D 记录从原点到其他个顶点的当前最短距离，初试集合 S 中只包含{V0}，当前 D 中第一行中 v0 到 v5 的距离最短为 3，所以选中为 W，加入到集合 S 中，再将 v0 通过 S 中的点到达其余各点的最短距离更新，以此类推。

11.试编写算法，判断在邻接矩阵存储结构上有向图 G 中两顶点 v1、v2 是否存在路径。

```
int SimplePath(MGraph G, int i, int j, int k);
/* 求有向图G的顶点i到j之间长度为k的简单路径条数*/
```

图的邻接矩阵存储结构的类型定义如下：

```
typedef enum {DG,DN,AG,AN} GraphKind; // 有向图,有向网,无向图,无向网

typedef struct {
    VRType adj; // 顶点关系类型。对无权图，用(是)或(否)表示相邻否；
                // 对带权图，则为权值类型
    InfoType *info; // 该弧相关信息的指针(可无)
}ArcCell,AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];

typedef struct {
    AdjMatrix  arcs; // 邻接矩阵
    VertexType vexs[MAX_VERTEX_NUM]; // 顶点向量
    int vexnum,arcnum; // 图的当前顶点数和弧数
    GraphKind  kind; // 图的种类标志
}MGraph;

int SimplePath(MGraph G, int i, int j, int k)
/* 求有向图G的顶点i到j之间长度为k的简单路径条数*/
{int sum=0,m=0;
 VRType p;
 if(i==j&&!k) return 1;
 else if(k>0)
        {visited[i]=1;
         for(;m<G.vexnum;m++)
          {p=G.arcs[i][m].adj;
           if((!visited[m])&&p)
           sum+=SimplePath(G,m,j,k-1);
          }
         visited[i]=0;
        }
 return sum;
}
```

12.试编写算法，在邻接表存储结构上，求有向图 G 的各顶点的入度、出度。

```
#include<iostream>
using namespace std;
#include<stdlib.h>
#include<stdio.h>
#define type char
#define maxsize 100
typedef struct arcnode
{
 int weizhi;//该边所指的顶点的位置
 struct arcnode *next;
}arcnode;
typedef struct vnode
{
 type data;//顶点信息
 arcnode *firstarc;//指向第一条依附该顶点的边的指针
}vnode,adjlist[maxsize];
typedef struct
{
 int vexnum,arcnum;
 adjlist a;
}graph;
void create(graph &G)
{
 cout<<"请输入图的顶点个数：";
 cin>>G.vexnum;
 cout<<"请输入顶点的信息(字符表示)："<<endl;
 for(int i=1;i<=G.vexnum;i++)
 {
  cin>>G.a[i].data;
  G.a[i].firstarc=NULL;
 }
 for(i=1;i<=G.vexnum;i++)
 {
  int k=0;
  int n;
  cout<<"请输入与顶点"<<G.a[i].data<<"相联通的顶点号(以大于顶点的数结束此次输入)：";
  while(cin>>n&&G.vexnum>=n>=1)   //以小于大于顶点的数结束输入
  {
   k++;
   arcnode *p;
      if(k==1)   //第一个边表节点
   {
```

```
    p=new arcnode;
    p->next=NULL;
    p->weizhi=n;
    G.a[i].firstarc=p;
   }
   else
   {
    arcnode *s;
    s=new arcnode;
    s->weizhi=n;
    s->next=NULL;
    p->next=s;
    p=s;
   }
   //cout<<"请输入与顶点"<<G.a[i].data<<"相联通的顶点号(以小于 1 的数或者大于顶点的数结束此次输入)：
";
  }

 }

}
//求有向图的第 n 个顶点出度
int getchudu (graph G,int n)
{
 int count=0;
 arcnode *p;
     p=G.a[n].firstarc;
 if(p==NULL)
  return 0;
 while(p)
 {
  count++;
  p=p->next;
 }
 return count;
}
//求有向图的第 n 个顶点入度
void getrudu(graph G,int n)
{
 int count=0;
 for(int i=1;i<=G.vexnum;i++)
 {
  arcnode *p;
  p=G.a[i].firstarc;
  if(p==NULL)
```

```
   continue;    //跳过以下操作
  while(p)
  {
   if(p->weizhi==n)
   {
    count++;
    p=p->next;
   }
   else p=p->next;
  }
 }
 if(count!=0)
  cout<<"顶点"<<G.a[n].data<<"的入度为:"<<count<<endl;
 else cout<<"顶点"<<G.a[n].data<<"没有入度"<<endl;
}
//打印有向图
void print(graph G)
{
 for(int i=1;i<=G.vexnum;i++)
 {
  cout<<"顶点"<<G.a[i].data<<"指向的顶点有：";
   arcnode *p;
      p=G.a[i].firstarc;
   while(p)
   {
    cout<<G.a[p->weizhi].data<<" ";
    p=p->next;
   }
   cout<<endl;
 }
}
void main()
{
 int n;
 graph G;
 create(G);
 print(G);
 cout<<"请输入要求出度和入度的顶点号(以大于顶点的数目结束输入)：";
 while(cin>>n&&G.vexnum>=n>=1)
 {
  cout<<"顶点"<<G.a[n].data<<"的出度为：";
  cout<<getchudu(G,n)<<endl;
  getrudu(G,n);
  cout<<"请输入要求出度和入度的顶点号(以大于顶点的数目结束输入)：";
```

```
 }
}
```

13.编写求有向图 G 中所有简单回路的算法。

```
#include    <stdio.h>
#define    N    10

int    m[N][N];
int    mark[N];
int    s[N],t[N];
int    e[][2]={{0,3},{1,2},{2,3},{3,1},{2,0},{7,6},{5,6},{4,7},{6,4}};

void    print(int    a,int    l){
int    i;
for    (i=0;i <l;i++)
if    (s[i]==a)    break;
for    (;i <l;i++)
printf( "%d-> ",s[i]);
printf( "%d\n ",a);
}

void    dfs(int    a,int    deep){
int    i;
if    (t[a]){
print(a,deep);
return;
}
if    (mark[a])
return;
mark[a]=1;
s[deep]=a;
t[a]=1;
for    (i=0;i <N;i++)
if    (m[a][i])
dfs(i,deep+1);
t[a]=0;
}

main(){
int    i;
for    (i=0;i <sizeof(e)/sizeof(e[0]);i++)
m[e[i][0]][e[i][1]]=1;
for    (i=0;i <N;i++)
dfs(i,0);
```

}

注意点：全局变量不用初始化，系统自动赋值为 0。
初始化 e[][]这个数组，就可以了。
int e[][2]={{0,3},{1,2},{2,3},{3,1},{2,0},{7,6},{5,6},{4,7},{6,4}};
0，3 就代表编号为 0 和编号为 3 的点有一条边。
1，2 就代表编号为 1 和编号为 2 的点有一条边。

14. 编写求无向图连通分量的算法。

思路：从图的遍历算法可知，每进行一次深度优先遍历算法或广度优先遍历算法，就可以得到图的一个连通分量的所有顶点。因此本题算法思想如下：对图中的每一个顶点，若该顶点未被访问过，则从该顶点出发进行深度优先遍历，且在遍历过程中输出访问的节点信息，即可得到相应的连通分量。算法如下所示：

算法：

```
void dfs ( int v)
{
  visited[v]=1; printf ("%3d",v);   //输出连通分量的顶点
  p=g[v].firstarc;
  while （p!=null）
  {
    if(visited[p->adjvex]==0)
      dfs(p->adjvex);
    p=p->next;
  }
}

void Count(AdjList g)              //求图中连通分量的个数
{
 int k=0 ;
 for (i=1;i<=n;i++ )
 if (visited[i]==0)
  {
    printf ("\n 第%d 个连通分量:\n"，++k);
    dfs(i);
  }
}
```