# 第 8 章 图

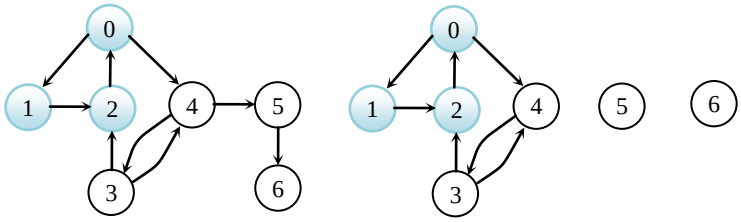
## 教材中练习题及参考答案

1. 图$G$是一个非连通图，共有28条边，则该图至少有多少个顶点？

**答：**由于$G$是一个非连通图，在边数固定时，顶点数最少的情况是该图由两个连通分量构成，且其中之一只含一个顶点（没有边），另一个为完全无向图。设该完全无向图的顶点数为$n$，其边数为$n(n-1)/2$，即$n(n-1)/2=28$，得$n=8$。所以，这样的非连通图至少有1+8=9个顶点。

2. 有一个如图 8.2（a）所示的有向图，给出其所有的强连通分量。

**答：**图中顶点0、1、2构成一个环，这个环一定是某个强连通分量的一部分。再考察顶点3、4，它们到这个环中的顶点都有双向路径，所以将顶点3、4加入。考察顶点5、6，它们各自构成一个强连通分量。该有向图的强连通分量有3个，如图8.2（b）所示。




(a) 一个有向图　　(b) 3 个强连通分量

图 8.2 一个有向图及其强连通分量

3. 对于稠密图和稀疏图，采用邻接矩阵和邻接表哪个更好些？

**答：**邻接矩阵适合于稠密图，因为邻接矩阵占用的存储空间与边数无关。邻接表适合于稀疏图，因为邻接表占用的存储空间与边数有关。

4. 对 $n$ 个顶点的无向图和有向图（均为不带权图），采用邻接矩阵和邻接表表示时，如何求解以下问题：

（1）图中有多少条边？

（2）任意两个顶点$i$和$j$是否有边相连？

（3）任意一个顶点的度是多少？

**答：**（1）对于邻接矩阵表示的无向图，图的边数等于邻接矩阵数组中为1的元素个数除以2；对于邻接表表示的无向图，图中的边数等于边结点的个数除以2。

对于邻接矩阵表示的有向图，图中的边数等于邻接矩阵数组中为1的元素个数；对于邻

接表表示的有向图，图中的边数等于边结点的个数。

（2）对于邻接矩阵*g*表示的无向图，邻接矩阵数组元素*g.edges*[*i*][*j*]为1表示它们有边相连，否则为无边相连。对于邻接矩阵*g*表示的有向图，邻接矩阵数组元素*g.edges*[*i*][*j*]为1表示从顶点*i*到顶点*j*有边，*g.edges*[*j*][*i*]为1表示从顶点*j*到顶点*i*有边。

对于邻接表*G*表示的无向图，若从头结点*G*->*adjlist*[*i*]的单链表中找到编号为*j*的边表结点，表示它们有边相连；否则为无边相连。对于邻接表*G*表示的有向图，若从头结点*G*->*adjlist*[*i*]的单链表中找到编号为*j*的边表结点，表示从顶点*i*到顶点*j*有边。若从头结点*G*->*adjlist*[*j*]的单链表中找到编号为*i*的边表结点，表示从顶点*j*到顶点*i*有边。

（3）对于邻接矩阵表示的无向图，顶点*i*的度等于第*i*行中元素为1的个数；对于邻接矩阵表示的有向图，顶点*i*的出度等于第*i*行中元素为1的个数，入度等于第*i*列中元素为1的个数，顶点*i*度等于它们之和。

对于邻接表*G*表示的无向图，顶点*i*的度等于*G*->*adjlist*[*i*]为头结点的单链表中边表结点个数。

对于邻接表*G*表示的有向图，顶点*i*的出度等于*G*->*adjlist*[*i*]为头结点的单链表中边表结点的个数；入度需要遍历所有的边结点，若*G*->*adjlist*[*j*]为头结点的单链表中存在编号为*i*的边结点，则顶点*i*的入度增1，顶点*i*的度等于入度和出度之和。

5. 对于如图 8.3 所示的一个无向图 *G*，给出以顶点 0 作为初始点的所有的深度优先遍历序列和广度优先遍历序列。

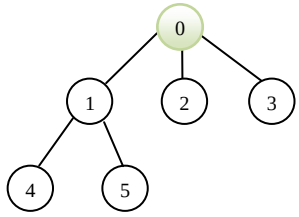



图 8.3 一个无向图 *G*

**答：**无向图 *G* 的所有的深度优先遍历序列如下：

0 1 4 5 2 3
0 1 5 4 2 3
0 1 4 5 3 2
0 1 5 4 3 2
0 2 1 4 5 3
0 2 1 5 4 3
0 2 3 1 4 5
0 2 3 1 5 4
0 3 1 4 5 2
0 3 1 5 4 2
0 3 2 1 4 5
0 3 2 1 5 4

无向图 *G* 所有的广度优先遍历序列如下：

0 1 2 3 4 5

0 1 2 3 5 4
0 1 3 2 4 5
0 1 3 2 5 4
0 2 1 3 4 5
0 2 1 3 5 4
0 2 3 1 4 5
0 2 3 1 5 4
0 3 1 2 4 5
0 3 1 2 5 4
0 3 2 1 4 5
0 3 2 1 5 4

6. 对于如图 8.4 所示的带权无向图，给出利用 Prim 算法（从顶点 0 开始构造）和 Kruskal 算法构造出的最小生成树的结果，要求结果按构造边的顺序列出。

**答：**利用普里姆算法从顶点0出发构造的最小生成树为：{(0，1)，(0，3)，(1，2)，(2，5)，(5，4)}。利用克鲁斯卡尔算法构造出的最小生成树为：{(0，1)，(0，3)，(1，2)，(5，4)，(2，5)}。

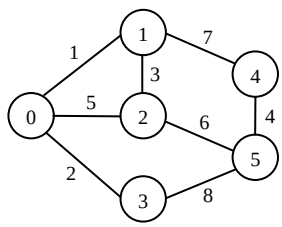



图 8.4　一个带权无向图

7. 对于一个顶点个数大于4的带权无向图，回答以下问题：

（1）该图的最小生成树一定是唯一的吗？如何所有边的权都不相同，那么其最小生成树一定是唯一的吗？

（2）如果该图的最小生成树不是唯一的，那么调用Prim算法和Kruskal算法构造出的最小生成树一定相同吗？

（3）如果图中有且仅有两条权最小的边，它们一定出现在该图的所有的最小生成树中吗？简要说明回答的理由。

（4）如果图中有且仅有3条权最小的边，它们一定出现在该图的所有的最小生成树中吗？简要说明回答的理由。

**答：**（1）该图的最小生成树不一定是唯一的。如何所有边的权都不相同，那么其最小生成树一定是唯一的。

（2）若该图的最小生成树不是唯一的，那么调用Prim算法和Kruskal算法构造出的最小生成树不一定相同。

（3）如果图中有且仅有两条权最小的边，它们一定会出现在该图的所有的最小生成树中。因为在采用Kruskal算法构造最小生成树时，首先选择这两条权最小的边加入，不会出现回路（严格的证明可以采用反证法）。

（4）如果图中有且仅有3条权最小的边，它们不一定出现在该图的所有的最小生成树

中。因为在采用Kruskal算法构造最小生成树时，选择这3条权最小的边加入时，有可能出现回路。例如，如图8.5所示的带权无向图，有3条边的权均为1，它们一定不会同时都出现在其任何最小生成树中。

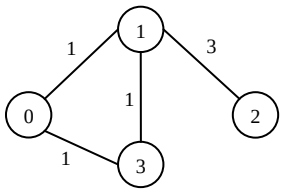



图 8.5　一个带权无向图

8. 对于如图8.6所示的带权有向图，采用Dijkstra算法求出从顶点0到其他各顶点的最短路径及其长度，要求给出求解过程。

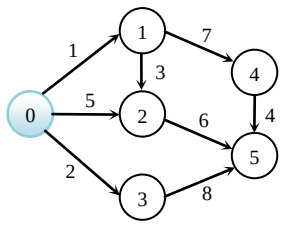



图 8.6　一个带权有向图 G

**答：**采用 Dijkstra 算法求从顶点 0 到其他各顶点的最短路径及其长度的过程如下：

（1）*S*={0}，*dist*[0..5]={0，1，5，2，∞，∞}，*path*[0..5]={0，0，0，0，−1，−1}。选取最短路径长度的顶点1。

（2）*S*={0，1}，调整顶点1到顶点2、4的最短路径长度，*dist*[0..5]={0，1，4，2，8，∞}，*path*[0..5]={0，0，1，0，1，−1}。选取最短路径长度的顶点3。

（3）*S*={0，1，3}，调整顶点3到顶点5的最短路径长度，*dist*[0..5]={0，1，4，2，8，10}，*path*[0..5]={0，0，1，0，1，3}。选取最短路径长度的顶点2。

（4）*S*={0，1，3，2}，调整顶点2到顶点5的最短路径长度，*dist*[0..5]={0，1，4，2，8，10}，*path*[0..5]={0，0，1，0，1，3}。选取最短路径长度的顶点4。

（5）*S*={0，1，3，2，4}，调整顶点4到顶点5的最短路径长度，*dist*[0..5]={0，1，4，2，8，10}，*path*[0..5]={0，0，1，0，1，3}。选取最短路径长度的顶点5。

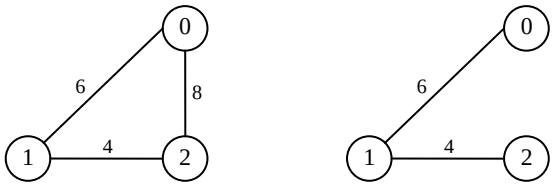
（6）*S*={0，1，3，2，4，5}，顶点5没有出边，*dist*[0..5]={0，1，4，2，8，10}，*path*[0..5]={0，0，1，0，1，3}。

最终结果如下：

从 0 到 1 的最短路径长度为:1，路径为:0, 1
从 0 到 2 的最短路径长度为:4，路径为:0, 1, 2
从 0 到 3 的最短路径长度为:2，路径为:0, 3
从 0 到 4 的最短路径长度为:8，路径为:0, 1, 4
从 0 到 5 的最短路径长度为:10，路径为:0, 3, 5

9. 对于一个带权连通图，可以采用 Prim 算法构造出从某个顶点 *v* 出发的最小生成树，问该最小生成树是否一定包含从顶点 *v* 到其他所有顶点的最短路径。如果回答是，请予以证明；如果回答不是，请给出反例。

**答：**不一定。例如，对于如图8.7（a）所示带权连通图，从顶点0出发的最小生成树如图8.7（b）所示，而从顶点0到顶点2的最短路径为0→2，而不是最小生成树中的0→1→2。




(a) 一个带权连通图　　(b) 最小生成树

图 8.7　一个带权连通图将其最小生成树

10. 若只求带权有向图G中从顶点*i*到顶点*j*的最短路径，如何修改Dijkstra算法来实现这一功能？

**答：**修改Dijkstra算法为：从顶点*i*开始（以顶点*i*为源点），按Dijkstra算法思路不断地扩展顶点集S，当扩展到顶点*j*时，算法结束，通过path回推出从顶点*i*到顶点*j*的最短路径。

11. Dijkstra 算法用于求单源最短路径，为了求一个图中所有顶点对之间的最短路径，可以以每个顶点作为源点调用 Dijkstra 算法，Floyd 算法和这种算法相比，有什么优势？

**答：**对于有*n*个顶点的图，求所有顶点对之间的最短路径，若调用Dijkstra算法*n*次，其时间复杂度为$O(n^3)$。Floyd算法的时间复杂度也是$O(n^3)$。但Floyd算法更快，这是因为前者每次调用Dijkstra算法时都是独立执行的，路径比较中得到的信息没有共享，而Floyd算法中每考虑一个顶点时所得到的路径比较信息保存在*A*数组中，会用于下次路径比较，从而提高整体查找最短路径的效率。

12. 回答以下有关拓扑排序的问题：

（1）给出如图8.8所示有向图的所有不同的拓扑序列。

（2）什么样的有向图的拓扑序列是唯一的？

（3）现要对一个有向图的所有顶点重新编号，使所有表示边的非0元素集中到邻接矩阵数组的上三角部分。根据什么顺序对顶点进行编号可以实现这个功能？

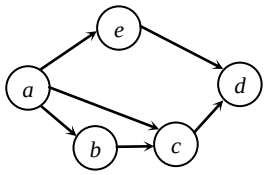



图 8.8　一个有向图

**答：**（1）该有向图的所有不同的拓扑序列有：*aebcd*、*abced*、*abecd*。

（2）这样的有向图的拓扑序列是唯一的：图中只有一个入度为0的顶点，在拓扑排序中每次输出一个顶点后都只有一个入度为0的顶点。

（3）首先该对有向图进行拓扑排序，把所有顶点排在一个拓扑序列中。然后按该序列对所有顶点重新编号，使得每条有向边的起点编号小于终点编号，就可以把所有边集中到邻接矩阵数组的上三角部分。

13. 已知有 6 个顶点（顶点编号为 0～5）的带权有向图 *G*，其邻接矩阵数组 *A* 为上三角矩阵，按行为主序（行优先）保存在如下的一维数组中：

| 4 | 6 | ∞ | ∞ | ∞ | 5 | ∞ | ∞ | ∞ | 4 | 3 | ∞ | ∞ | 3 | 3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

要求：

（1）写出图*G*的邻接矩阵数组*A*。

（2）画出带权有向图*G*。

（3）求图*G*的关键路径，并计算该关键路径的长度。

答：（1）图*G*的邻接矩阵数组*A*如图8.9所示。

（2）有向带权图*G*如图8.10所示。

（3）图8.11中粗线所标识的4个活动组成图*G*的关键路径。

$$A=\begin{bmatrix} 0 & 4 & 6 & \infty & \infty & \infty \\ \infty & 0 & 5 & \infty & \infty & \infty \\ \infty & \infty & 0 & 4 & 3 & \infty \\ \infty & \infty & \infty & 0 & \infty & 3 \\ \infty & \infty & \infty & \infty & 0 & 3 \\ \infty & \infty & \infty & \infty & \infty & 0 \end{bmatrix}$$

图 8.9 邻接矩阵 *A*

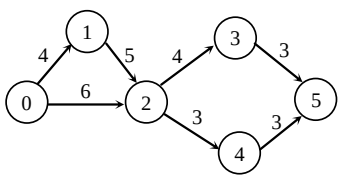



图 8.10 图 *G*

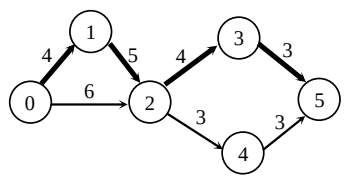



图 8.11 图 *G* 中的关键路径

14. 假设不带权有向图采用邻接矩阵 *g* 存储，设计实现以下功能的算法：

（1）求出图中每个顶点的入度。

（2）求出图中每个顶点的出度。

（3）求出图中出度为0的顶点数。

**解**：利用邻接矩阵的特点和相关概念得到如下算法：

```
void InDs1(MatGraph g)              //求出图 G 中每个顶点的入度
{   int i,j,n;
    printf("各顶点入度:\n");
    for (j=0;j<g.n;j++)
    {   n=0;
        for (i=0;i<g.n;i++)
            if (g.edges[i][j]!=0)
```

```
                n++;          //n 累计入度数
        printf("  顶点%d:%d\n",j,n);
    }
}
void OutDs1(MatGraph g)          //求出图 G 中每个顶点的出度
{   int i,j,n;
    printf("各顶点出度:\n");
    for (i=0;i<g.n;i++)
    {   n=0;
        for (j=0;j<g.n;j++)
            if (g.edges[i][j]!=0)
                n++;          //n 累计出度数
        printf("  顶点%d:%d\n",i,n);
    }
}
void ZeroOutDs1(MatGraph g)      //求出图 G 中出度为 0 的顶点个数
{   int i,j,n;
    printf("出度为 0 的顶点:");
    for (i=0;i<g.n;i++)
    {   n=0;
        for (j=0;j<g.n;j++)
            if (g.edges[i][j]!=0)  //存在一条出边
                n++;
        if (n==0)
            printf("%2d\n",i);
    }
    printf("\n");
}
```

15. 假设不带权有向图采用邻接表 $G$ 存储，设计实现以下功能的算法：

（1）求出图中每个顶点的入度。

（2）求出图中每个顶点的出度。

（3）求出图中出度为0的顶点数。

**解：**利用邻接表的特点和相关概念得到如下算法：

```
void InDs2(AdjGraph *G)              //求出图 G 中每个顶点的入度
{   ArcNode *p;
    int A[MAXV],i;                   //A 存放各顶点的入度
    for (i=0;i<G->n;i++)             //A 中元素置初值 0
        A[i]=0;
    for (i=0;i<G->n;i++)             //扫描所有头结点
    {   p=G->adjlist[i].firstarc;
        while (p!=NULL)              //扫描边结点
        {   A[p->adjvex]++;          //表示 i 到 p->adjvex 顶点有一条边
            p=p->nextarc;
        }
    }
    printf("各顶点入度:\n");         //输出各顶点的入度
    for (i=0;i<G->n;i++)
        printf("  顶点%d:%d\n",i,A[i]);
```

```
}
void OutDs2(AdjGraph *G)                    //求出图 G 中每个顶点的出度
{   int i,n;
    ArcNode *p;
    printf("各顶点出度:\n");
    for (i=0;i<G->n;i++)                    //扫描所有头结点
    {   n=0;
        p=G->adjlist[i].firstarc;
        while (p!=NULL)                     //扫描边结点
        {   n++;                            //累计出边的数
            p=p->nextarc;
        }
        printf("  顶点%d:%d\n",i,n);
    }
}
void ZeroOutDs2(AdjGraph *G)                //求出图 G 中出度为 0 的顶点数
{   int i,n;
    ArcNode *p;
    printf("出度为 0 的顶点:");
    for (i=0;i<G->n;i++)                    //扫描所有头结点
    {   p=G->adjlist[i].firstarc;
        n=0;
        while (p!=NULL)                     //扫描边结点
        {   n++;                            //累计出边的数
            p=p->nextarc;
        }
        if (n==0)                           //输出出边数为 0 的顶点编号
            printf("%2d",i);
    }
    printf("\n");
}
```

16. 假设一个连通图采用邻接表作为存储结构，试设计一个算法，判断其中是否存在经过顶点 *v* 的回路。

**解：**从顶点 *v* 出发进行深度优先遍历，用 *d* 记录走过的路径长度，对每个访问的顶点设置标记为 1。若当前访问顶点 *u*，表示 *v*⇨*u* 存在一条路径，如果顶点 *u* 的邻接点 *w* 等于 *v* 并且 *d*>1，表示顶点 *u* 到 *v* 有一条边，即构成经过顶点 *v* 的回路，如图 8.12 所示。Cycle 算法中 *has* 是布尔值，初始调用时置为 false，执行后若为 true 表示存在经过顶点 *v* 的回路，否则表示没有相应的回路。

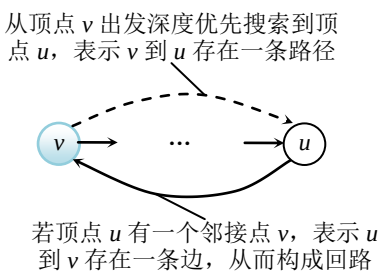


图 8.12 图中存在回路的示意图

对应的算法如下：

```
int visited[MAXV];                    //全局变量数组
void Cycle(AdjGraph *G,int u,int v,int d,bool &has)
{   //调用时 has 置初值 false,d 为-1
    ArcNode *p;   int w;
    visited[u]=1; d++;                //置已访问标记
    p=G->adjlist[u].firstarc;         //p 指向顶点 u 的第一个邻接点
    while (p!=NULL)
    {   w=p->adjvex;
        if (visited[w]==0)            //若顶点 w 未访问,递归访问它
            Cycle(G,w,v,d,has);       //从顶点 w 出发搜索
        else if (w==v && d>1)         //u 到 v 存在一条边且回路长度大于 1
        {   has=true;
            return;
        }
        p=p->nextarc;                 //找下一个邻接点
    }
}
bool hasCycle(AdjGraph *G,int v)      //判断连通图 G 中是否有经过顶点 v 的回路
{   bool has=false;
    Cycle(G,v,v,-1,has);              //从顶点 v 出发搜索
    return has;
}
```

17．假设图 $G$ 采用邻接表存储，试设计一个算法，判断无向图 $G$ 是否是一棵树。若是树，返回真；否则返回假。

**解**：一个无向图 $G$ 是一棵树的条件是：$G$ 必须是无回路的连通图或者是有 $n$-1 条边的连通图。这里采用后者作为判断条件，通过深度优先遍历图 $G$，并求出遍历过的顶点数 $vn$ 和边数 $en$，若 $vn$==$G$->$n$ 成立（表示为连通图）且 $en$==2($G$->$n$-1)（遍历边数为 2($G$->$n$-1)）成立，则 $G$ 为一棵树。对应的算法如下：

```
void DFS2(AdjGraph *G,int v,int &vn,int &en)
{   //深度优先遍历图 G,并求出遍历过的顶点数 vn 和边数 en
    ArcNode *p;
    visited[v]=1;vn++;                //遍历过的顶点数增 1
    p=G->adjlist[v].firstarc;
    while (p!=NULL)
    {   en++;                         //遍历过的边数增 1
        if (visited[p->adjvex]==0)
            DFS2(G,p->adjvex,vn,en);
        p=p->nextarc;
    }
}
int IsTree(AdjGraph *G)               //判断无向图 G 是否是一棵树
{   int vn=0,en=0,i;
    for (i=0;i<G->n;i++)
        visited[i]=0;
    DFS2(G,1,vn,en);
```

```
    if (vn==G->n && en==2*(G->n-1))
        return 1;                    //遍历顶点为G->n个,遍历边数为2(G->n-1),则为树
    else
        return 0;
}
```

18. 设有 5 地（0～4）之间架设有 6 座桥（*A*～*F*），如图 8.13 所示，设计一个算法，从某一地出发，经过每座桥恰巧一次，最后仍回到原地。

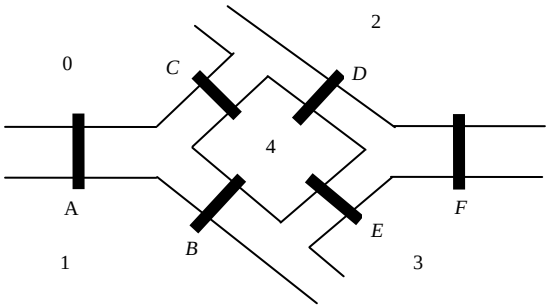



图 8.13　实地图

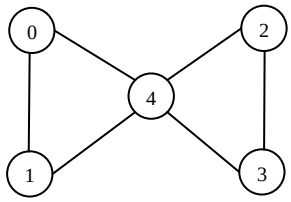



图 8.14　一个无向图 *G*

**解：**该实地图对应的一个无向图 *G* 如图 8.14 所示，本题变为从指定点 *k* 出发找经过所有 6 条边回到 *k* 顶点的路径，由于所有顶点的度均为偶数，可以找到这样的路径。对应的算法如下：

```
int vedge[MAXV][MAXV];       //边访问数组,vedge[i][j]表示(i,j)边是否访问过
void Traversal(AdjGraph *G,int u,int v,int k,int path[],int d)
//d是到当前为止已走过的路径长度，调用时初值为-1
{   int w,i;
    ArcNode *p;
    d++; path[d]=v;                  //(u,v)加入到path中
    vedge[u][v]=vedge[v][u]=1;       //(u,v)边已访问
    p=G->adjlist[v].firstarc;        //p指向顶点v的第一条边
    while (p!=NULL)
    {   w=p->adjvex;                 //(v,w)有一条边
        if (w==k && d==G->e-1)       //找到一个回路，输出之
        {   printf("  %d->",k);
            for (i=0;i<=d;i++)
                printf("%d->",path[i]);
            printf("%d\n",w);
        }
        if (vedge[v][w]==0)          //(v,w)未访问过,则递归访问之
```

```
            Traversal(G,v,w,k,path,d);
        p=p->nextarc;                   //找 v 的下一条边
    }
    vedge[u][v]=vedge[v][u]=0;          //恢复环境：使该边点可重新使用
}
void FindCPath(AdjGraph *G,int k)     //输出经过顶点 k 和所有边的全部回路
{   int path[MAXV];
    int i,j,v;
    ArcNode *p;
    for (i=0;i<G->n;i++)                //vedge 数组置初值
        for (j=0;j<G->n;j++)
            if (i==j) vedge[i][j]=1;
            else vedge[i][j]=0;
    printf("经过顶点%d 的走过所有边的回路:\n",k);
    p=G->adjlist[k].firstarc;
    while (p!=NULL)
    {   v=p->adjvex;
        Traversal(G,k,v,k,path,-1);
        p=p->nextarc;
    }
}
```

设计如下主函数：

```
int main()
{   int v=4;
    AdjGraph *G;
    int n=5,e=6;
    int A[MAXV][MAXV]={{0,1,0,0,1},  {1,0,0,0,1},
                       {0,0,0,1,1},{0,0,1,0,1},{1,1,1,1,0}};
    CreateAdj(G,A,n,e);
    printf("图 G 的邻接表:\n");DispAdj(G);//输出邻接表
    FindCPath(G,v);
    printf("\n");
    DestroyAdj(G);
    return 1;
}
```

程序执行结果如下：

```
图 G 的邻接表:
  0:   1[1]→  4[1]→∧
  1:   0[1]→  4[1]→∧
  2:   3[1]→  4[1]→∧
  3:   2[1]→  4[1]→∧
  4:   0[1]→  1[1]→  2[1]→  3[1]→∧
经过顶点 4 的走过所有边的回路:
  4->0->1->4->2->3->4
  4->0->1->4->3->2->4
  4->1->0->4->2->3->4
  4->1->0->4->3->2->4
  4->2->3->4->0->1->4
```

```
4->2->3->4->1->0->4
4->3->2->4->0->1->4
4->3->2->4->1->0->4
```

19. 设不带权无向图 *G* 采用邻接表表示，设计一个算法求源点 *i* 到其余各顶点的最短路径。

**解:**利用广度优先遍历的思想,求 *i* 和 *j* 两顶点间的最短路径转化为求从 *i* 到 *j* 的层数，为此设计一个 *level*[]数组记录每个顶点的层次。对应的算法如下：

```
void ShortPath(AdjGraph *G,int i)
{   int qu[MAXV],level[MAXV];
    int front=0,rear=0,k,lev;              //lev 保存从 i 到访问顶点的层数
    ArcNode *p;
    visited[i]=1;
    rear++;qu[rear]=i;level[rear]=0;       //顶点 i 已访问,将其进队
    while (front!=rear)                    //队非空则执行
    {   front=(front+1)% MAXV;
        k=qu[front];                       //出队
        lev=level[front];
        if (k!=i)
            printf("  顶点%d 到顶点%d 的最短距离是:%d\n",i,k,lev);
        p=G->adjlist[k].firstarc;          //取 k 的边表头指针
        while (p!=NULL)                    //依次搜索邻接点
        {   if (visited[p->adjvex]==0)     //若未访问过
            {   visited[p->adjvex]=1;
                rear=(rear+1)% MAXV;
                qu[rear]=p->adjvex;        //访问过的邻接点进队
                level[rear]=lev+1;
            }
            p=p->nextarc;                  //找顶点 i 的下一邻接点
        }
    }
}
```

设计如下主函数：

```
int main()
{   AdjGraph *G;
    int n=5,e=8;
    int A[MAXV][MAXV]={{0,1,0,1,1},{1,0,1,1,0},
        {0,1,0,1,1},{1,1,1,0,1},{1,0,1,1,0}};
    CreateAdj(G,A,n,e);                    //创建《教程》图 8.1(a)的邻接表
    printf("图 G 的邻接表:\n");DispAdj(G);//输出邻接表
    for (int i=0;i<n;i++)
        visited[i]=0;
    printf("顶点 1 到其他各顶点的最短距离如下:\n");
    ShortPath(G,1);
    return 1;
}
```

程序的执行结果如下：

```
图 G 的邻接表:
  0:   1[1]→  3[1]→  4[1]→∧
  1:   0[1]→  2[1]→  3[1]→∧
  2:   1[1]→  3[1]→  4[1]→∧
  3:   0[1]→  1[1]→  2[1]→  4[1]→∧
  4:   0[1]→  2[1]→  3[1]→∧
顶点 1 到其他各顶点的最短距离如下:
  顶点 1 到顶点 0 的最短距离是:1
  顶点 1 到顶点 2 的最短距离是:1
  顶点 1 到顶点 3 的最短距离是:1
  顶点 1 到顶点 4 的最短距离是:2
```

20. 对于一个带权有向图，设计一个算法输出从顶点 *i* 到顶点 *j* 的所有路径及其路径长度。调用该算法求出《教程》图 8.35 中顶点 0 到顶点 3 的所有路径及其长度。

**解：**采用回溯的深度优先遍历方法。增加一个形参 *length* 表示路径长度，其初始值为 0。当从顶点 *u* 出发，设置 *visited*[*u*]=1，当找到一个没有访问过的邻接点 *w*，就从 *w* 出发递归查找，其路径长度 *length* 增加<*u*，*w*>边的权值。当找到终点 *v*，就输出一条路径。通过设置 *visited*[*u*]=0 回溯查找所有的路径。对应的算法如下：

```
int visited[MAXV];
void findpath(AdjGraph *G,int u,int v,int path[],int d,int length)
{   //d 表示 path 中顶点个数，初始为 0；length 表示路径长度，初始为 0
    int w,i;
    ArcNode *p;
    path[d]=u; d++;                  //顶点 u 加入到路径中，d 增 1
    visited[u]=1;                    //置已访问标记
    if (u==v && d>0)                 //找到一条路径则输出
    {   printf("  路径长度:%d, 路径:",length);
        for (i=0;i<d;i++)
            printf("%2d",path[i]);
        printf("\n");
    }
    p=G->adjlist[u].firstarc;        //p 指向顶点 u 的第一个邻接点
    while (p!=NULL)
    {   w=p->adjvex;                 //w 为顶点 u 的邻接点
        if (visited[w]==0)           //若 w 顶点未访问,递归访问它
            findpath(G,w,v,path,d,p->weight+length);
        p=p->nextarc;                //p 指向顶点 u 的下一个邻接点
    }
    visited[u]=0;                    //恢复环境,使该顶点可重新使用
}
```

设计如下主函数求《教程》图 8.35 中顶点 0 到顶点 3 的所有路径及其长度：

```
int main()
{   AdjGraph *G;
    int A[MAXV][MAXV]={
        {0,4,6,6,INF,INF,INF}, {INF,0,1,INF,7,INF,INF},
        {INF,INF,0,INF,6,4,INF},{INF,INF,2,0,INF,5,INF},
        {INF,INF,INF,INF,0,INF,6},{INF,INF,INF,INF,1,0,8},
        {INF,INF,INF,INF,INF,INF,0}};
```

```
    int n=7, e=12;
    CreateAdj(G,A,n,e);                //创建《教程》中图 8.35 的邻接表
    printf("图 G 的邻接表:\n");
    DispAdj(G);                        //输出邻接表
    int u=0,v=5;
    int path[MAXV];
    printf("从%d->%d 的所有路径:\n",u,v);
    findpath(G,u,v,path,0,0);
    DestroyAdj(G);
    return 1;
}
```

上述程序执行结果如下：

```
图 G 的邻接表:
  0:  1[4] → 2[6] → 3[6] → ∧
  1:  2[1] → 4[7] → ∧
  2:  4[6] → 5[4] → ∧
  3:  2[2] → 5[5] → ∧
  4:  6[6] → ∧
  5:  4[1] → 6[8] → ∧
  6:  ∧
从 0->5 的所有路径:
  路径长度:9, 路径: 0 1 2 5
  路径长度:10, 路径:0 2 5
  路径长度:12, 路径:0 3 2 5
  路径长度:11, 路径:0 3 5
```