图的基础知识

1、图的定义

图是由顶点的有穷非空集合和顶点之间边的集合组成（图可以没有边，但不能没有顶点），通常表示为：$G = (V, E)$，其中，G 表示一个图，V 是顶点的集合，E 是边的集合。

如上图所示，该图顶点集合 $V = \{v_0, v_1, v_2, v_3, v_4\}$，边的集合
$E = \{(v_0, v_1), (v_0, v_3), (v_1, v_2), (v_1, v_4), (v_2, v_3), (v_2, v_4)\}$。

2、有向图与有向图

在图中，若顶点 v_i 和 v_j 之间的边没有方向，则称这条边为无向边，用无序偶对 (v_i, v_j) 表示；若从顶点 v_i 到 v_j 的边有方向，则称这条边为有向边（也成为弧，以区别于无向边），用有序偶对 $< v_i, v_j >$ 表示，v_i 称为弧尾，v_j 称为弧头。如果图的任意两个顶点之间的边都是无向边，则称该图为无向图，否则称该图为有向图。

如上图为无向图和有向图的例子，其中无向图 G_1 可以表示为：

```
G1 = (V1, E1)
V1 = {0, 1, 2, 3, 4}
E1 = {(0, 1), (0, 3), (1, 2), (1, 4), (2, 3), (2, 4)}
```

有向图 G_2 可以表示为：

```
G2 = (V2, E2)
V2 = {0, 1, 2, 3}
E2 = {<0, 1>, <0, 2>, <2, 3>, <3, 0>}
```

3、简单图

简单图满足以下两条内容：

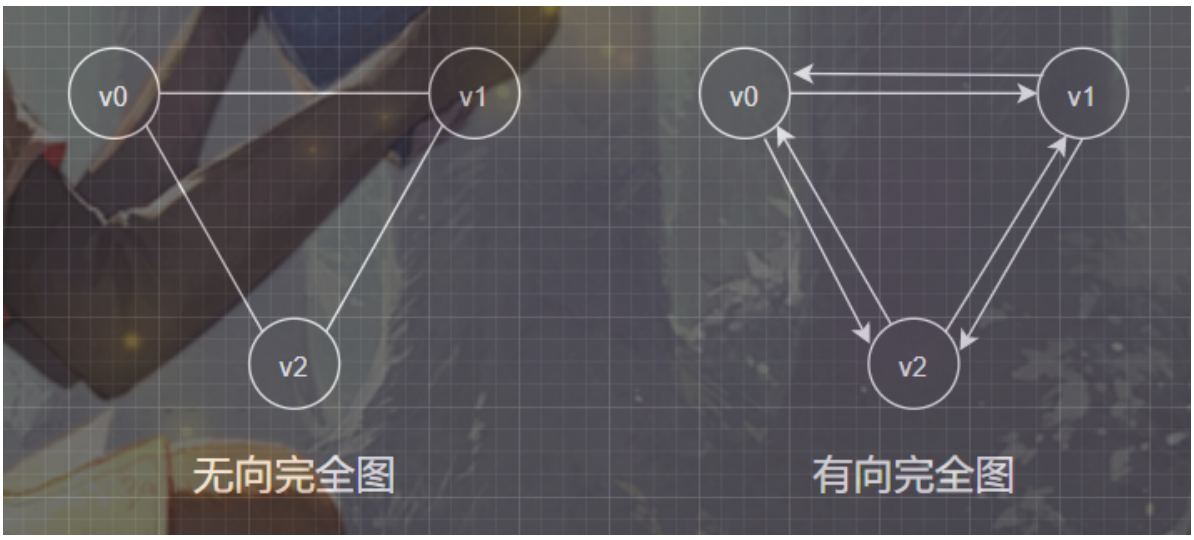
（1）不存在重复的边

（2）不存在顶点到自身的边（即无环无重边）

所以上面所展示的有向图和无向图都是简单图。**与简单图相对的是多重图，即：两个结点直接边数多于一条，又允许顶点通过同一条边与自己关联。**

4、完全图

无向图中任意两点之间都存在边，称为无向完全图；有向图中任意两点之间都存在方向向反的两条弧，称为有向完全图。含有 n 个顶点的无向完全图有 $n * (n - 1)$ 条边；含有 n 个顶点的有向完全图有 $n * (n - 1)$ 条边，如下图一个为无向完全图，一个为有向完全图。



5、稀疏图和稠密图

称边数很少的图为稀疏图，反之，称为稠密图。

5、子图

若有两个图 $G=(V,\ E)$，$G_1=(V_1,\ E_1)$，若 V_1 是 V 的子集且 E_1 是 E 的子集，称 G_1 是 G 的子图。

6、邻接、依附

在无向图中，对于任意两个顶点 v_1 和 v_j，若存在边 $(v_i,\ v_j)$，则称顶点 v_i 和 v_j 互为邻接点，同时称边 $(v_i,\ v_j)$ 依附于顶点 v_i 和 v_j。

在有向图中，对于任意两个顶点 v_i 和 v_j，若存在弧 $<v_i,\ v_j>$，则称顶点 v_i 邻接到 v_j，顶点 v_j 邻接自 v_i，同时称弧 $<v_i,\ v_j>$ 依附于顶点 v_i 和 v_j。在不致混淆的情况下，通常称 v_j 是 v_i 的邻接点。

7、顶点的度、入度、出度

顶点的度为以该顶点为一个端点的边的数目。**对于无向图，顶点的边数为度，度数之和是顶点边数的两倍；对于有向图，入度是以顶点为终点，出度相反。有向图的全部顶点入度之和等于出度之和且等于边数。顶点的度等于入度与出度之和。注意：入度与出度是针对有向图来说的。**

8、路径、路径长度、回路

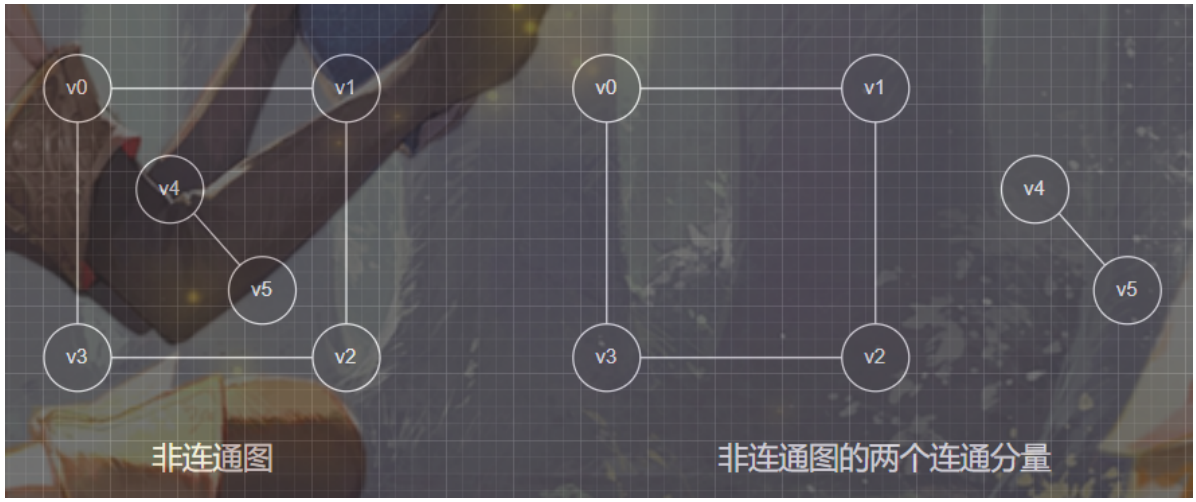
在无向图 $G=(V,\ E)$ 中顶点 v_p 到 v_q 之间的路径是一个顶点序列 $v_p=v_{i0}v_{i1}\ldots v_{im}=v_q$，其中，$(v_{ij-1},\ v_{ij})\in E\ (1\leq j\leq m)$；如果 G 是有向图，则 $<v_{ij-1},\ v_{ij}>\in E\ (1\leq j\leq m)$。路径上边的数目称为路径长度、第一个顶点和最后一个顶点相同的路径成为回路。显然，在图中路径可能不唯一，回路也可能不唯一。（简单来说，两顶点之间的路径指顶点之间经过的顶点序列，经过路径上边的数目称为路径长度。若有 n 个顶点，且边数大于 $n-1$，那么此图一定有环）。

9、简单路径、简单回路

在路径序列中，顶点不重复出现的路径称为简单路径。除了第一个顶点和最后一个顶点之外，其余顶点不重复出现的回路称为简单回路。

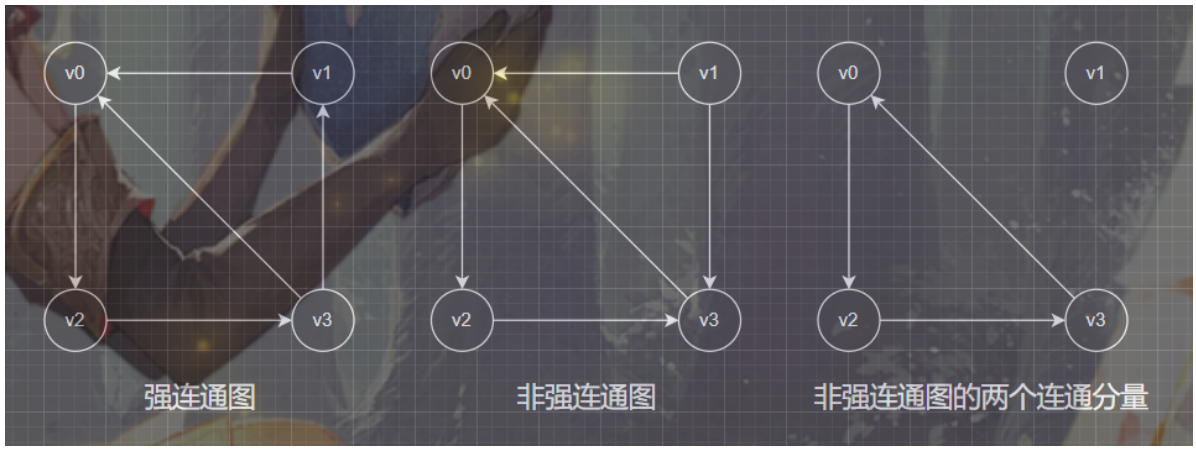
10、连通图、连通分量

在无向图中，若顶点 v_i 和 v_j $(i \neq j)$ 之间存在路径，则称 v_i 和 v_j 是连通的。若任意顶点的 v_i 和 v_j $(i \neq j)$ 之间均有路径，则称该图是连通图。非连通图的极大连通子图称为连通分量，极大的含义是指子图在满足连通的条件下，包括所有连通的的顶点以及与这些顶点相关联的所有边。下图展示了 一个非连通图以及它的两个连通分量。



11、强连通图、强连通分量

在有向图中，对任意顶点 v_i 和 v_j $(i \neq j)$，若从顶点 v_i 到 v_j 均有路径，则称该有向图是强连通图。强连通图的极大连通子图称为强连通分量。下图分别展示了一个强连通图和非强连通图，以及非强连通图的强连通分量。 （连通图和连通分量是相对于无向图来说的，而强连通图和强连通分量是相对于有向图来说的）



12、生成树和生成森林

连通图的生成树是包含图中全部顶点的一个极小连通子图，若图中有 n 个顶点，则生成树有 $n-1$ 条边。所以对于生成树而言，若砍去一条边，就会变成非连通图。在非连通图中，连通分量的生成树构成了非连通图的生成森林。如下图为无向图的一棵生成树。

13、边的权和网图

在图中，权通常是对边赋予的有意义的数值量，在实际应用中，权可以有具体的含义。例如，对于城市交通线路图，边上的权可以表示该条线路的长度或者等级。边上带权的图称为带权图或网图，如下图为无向网图和有向网图的示例。

14、图的存储结构

（1）邻接矩阵

图的邻接矩阵用一个二维数组存储图中的边（即个顶点之间的邻接关系）。存储顶点之间的邻接关系的二维数组称为邻接矩阵。设图 $G=(V,\ E)$ 有 n 个顶点，则邻接矩阵是一个 $n*n$ 的方阵，定义为：$edge[i][j]=\begin{cases}1\text{ , 若}(v_i,\ v_j\in E\text{ 或 }<v_i,\ v_j>\in E)\\0\text{ , 否则}\end{cases}$，若 G 是网图，则邻接矩阵的定义为：$edge[i][j]=\begin{cases}w_{ij},\text{ 若}(v_i,\ v_j)\in E\text{或 }<v_i,\ v_j>\in E\\0\quad\text{，若 }i=j\\\infty\text{ ，否则}\end{cases}$，其中，$w_{ij}$ 表示边 $(v_i,\ v_j)$ 或弧 $<v_i,\ v_j>$ 上的权值；∞ 表示不可达。

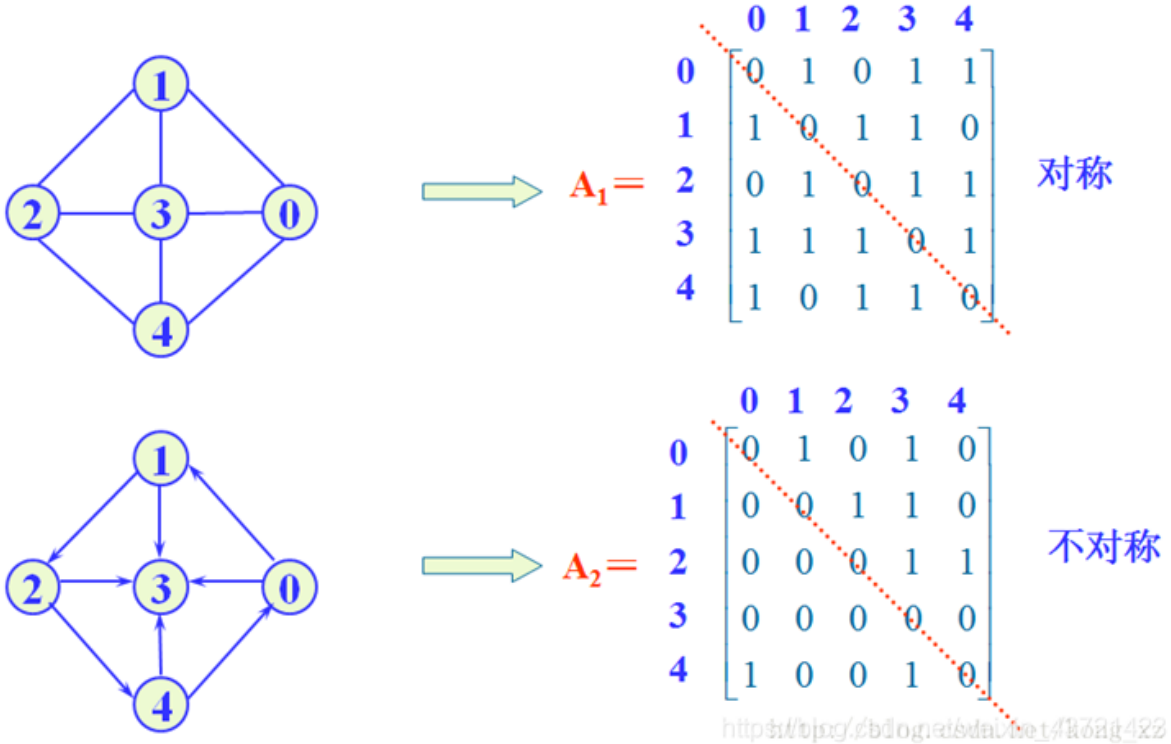
在邻接矩阵中容易实现的基本操作：

（1）对于无向图，顶点 i 的度等于邻接矩阵中第 i 行（或第 i 列）的非零元素的个数。对于有向图，顶点 i 的出度等于邻接矩阵中第 i 行的非零元素个数，顶点 i 的入度等于邻接矩阵中第 i 列非零元素的个数。

（2）判断顶点 i 和 j 之间是否存在边，只需测试邻接矩阵中相应位置的元素 $edge[i][j]$，若其值为 0，则有边；否则，顶点 i 和 j 之间不存在边。

（3）查找顶点 i 的所有邻接点，扫描邻接矩阵的第 i 行，若 $edge[i][j]$ 的值为 1，则顶点 j 是顶点 i 的邻接点。

邻接矩阵示例如下（无向图所对应的邻接矩阵一定是对称的）



（2）邻接表

在邻接表中，对图中每个顶点建立一个单链表，第 i 个单链表中的节点表示依附于顶点 i 的边（对有向图是以顶点 i 为尾的边）。每个单链表上附设一个表头节点。

邻接表的特点如下：

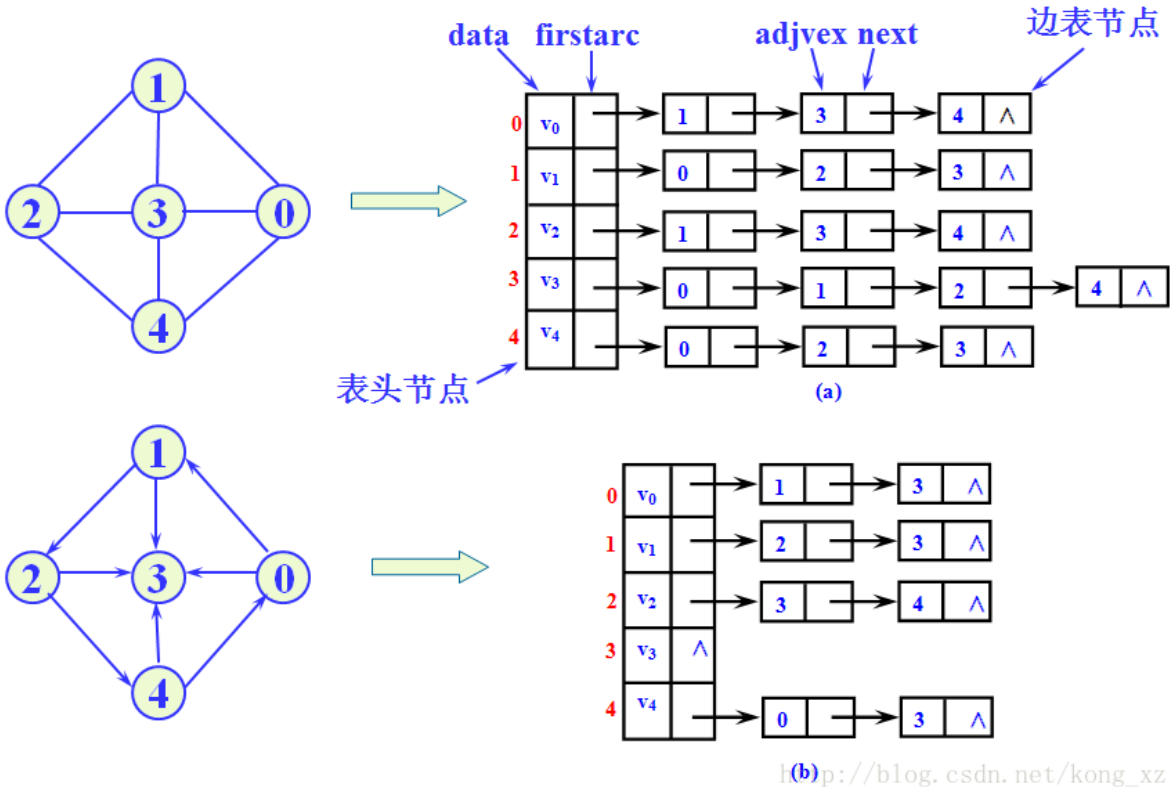
（1）邻接表表示不唯一。这是因为在每个顶点对应的单链表中，各边节点的链接次序可以是任意的，取决于建立邻接表的算法以及边的输入次序。

（2）对于有 n 个顶点和 e 条边的**无向图**，其邻接表有 n 个顶点节点和 $2*e$ 个边节点。

（3）对于**无向图**，邻接表的顶点 i 对应的第 i 个链表的边节点数目正好是顶点 i 的度。

（4）对于**有向图**，邻接表的顶点 i 对应的第 i 个链表的边节点数目仅仅是顶点 i 的出度。

邻接表示例如下



（3）邻接矩阵和邻接表的比较

我们设图 G 有 n 个顶点与 e 条边。

在空间性能方面：图的邻接矩阵是一个 $n*n$ 的矩阵，所以其空间代价是 $O(n^2)$。邻接表的空间代价与图的边数以及顶点数有关，其空间复杂度为 $O(n+e)$。在一般情况下，图越稠密，邻接矩阵的空间效率相应地越高，而对稀疏图使用邻接表存储，则能获得较高的空间效率。（稠密图中 e 约为 n^2 级别）

在时间性能方面：若要在图中访问某个点的邻接点，如果使用的是邻接表，只需要检查该顶点的边表，即只检查与它相关联的边，平均查找的复杂度为 $O(e/n)$；如果使用邻接矩阵，则必须检查所有可能的边，查找复杂度为 $O(n)$，从该操作上来讲，邻接矩阵的时间复杂度高于邻接表。

15、图的遍历方式

（1）图的DFS遍历（CHDOJ 2248）

AC代码：

```
#include <bits/stdc++.h>

using namespace std;

int g[15][15]; //邻接矩阵
bool st[15]; //存储被没被遍历过
int n, e; //n个点，e条边

void dfs (int x)
{
    printf ("%d ", x);
    st[x] = true;
    for (int i = 1; i <= n; i ++) //因为要优先访问最小编号的结点，所以从小到大循环
        if (g[x][i] == 1 && !st[i]) //依次遍历点x能到达的点
            dfs (i);
}

int main ()
{
    scanf ("%d%d", &n, &e);
    int x, y;
    for (int i = 1 ; i <= e; i ++) //输入数据
    {
        scanf ("%d%d", &x, &y);
        g[x][y] = 1, g[y][x] = 1; //注意题目说的是无向图，每次要存两条边
    }
    dfs (1); //从1号结点开始遍历
    return 0;
}
```

（2）图的BFS遍历（CHDOJ 2249）

AC代码：

```
#include <bits/stdc++.h>

using namespace std;

int g[15][15]; //邻接矩阵
bool st[15]; //判断是否被遍历过
int n, e; //n个点，e条边

void bfs (int x)
{
    queue <int> q; //宽搜要用的队列
    st[x] = true;
    q.push (x); //将1号点加入
    while (!q.empty ())
    {
        int a = q.front ();
        printf ("%d ", a);
```

```
        q.pop ();
        for (int i = 1; i <= n; i ++) //从小到大遍历
        {
            if (!st[i] && g[a][i] == 1)
            {
                q.push (i);
                st[i] = true;
            }
        }
    }
}

int main ()
{
    scanf ("%d%d", &n, &e);
    int x, y;
    for (int i = 1 ; i <= e; i ++) //输入数据
    {
        scanf ("%d%d", &x, &y);
        g[x][y] = 1, g[y][x] = 1; //注意题目说的是无向图，每次要存两条边
    }
    bfs (1); //从1号结点开始遍历
    return 0;
}
```

拓扑排序

定义：在图论中，**拓扑排序（Topological Sorting）**是一个**有向无环图（DAG）**的所有顶点的线性序列。且该序列必须满足以下两个条件：

（1）每个顶点出现且只出现一次

（2）若存在一条从顶点 A 到顶点 B 的路径，那么在拓扑序列中顶点 A 排在顶点 B 的前面。

注意：有向无环图才有拓扑序列，非有向无环图没有拓扑排序这一说。

求法：求一个有向无环图的拓扑序列主要分为以下三步：

（1）从有向无环图中选择一个没有前驱（即入度为 0）的顶点并输出

（2）从图中删除该顶点和所有以它为起点的有向边

（3）重复步骤（1）和（2）直到图为空或者当前图中不存在无前驱的顶点。后一种情况表明该图中必然存在环。

如下图就为求拓扑排序的一个例子。

于是，得到拓扑排序后的结果是 $\{1,\ 2,\ 4,\ 3,\ 5\}$

通常，一个有向无环图可以有一个或者多个拓扑序列（**拓扑序列可能不唯一**）。

模板题：CHDOJ 4048

AC代码：

```
//拓扑排序
#include <bits/stdc++.h>

using namespace std;

const int N = 55;

int e[N * N], ne[N * N], idx, h[N]; //邻接表存图，边数一定不会超过点数的平方
int n;
int d[N]; //存点的入度

vector <int> an; //存拓扑排序

void add (int a, int b) //邻接表的插入操作
{
    e[idx] = b;
    ne[idx] = h[a];
    h[a] = idx ++;
}

void topsort ()
{
    stack <int> q; //我们这里用栈来维护入度为零的点
    for (int i = 0; i < n; i ++) //将第一批入度为零的点送进去
        if (!d[i])
            q.push (i);
    while (!q.empty ())
    {
        int t = q.top ();
        an.push_back (t);
        q.pop ();
        for (int i = h[t]; i != -1; i = ne[i]) //删除邻边
```

```
        {
            int j = e[i];
            d[j] --;
            if (d[j] == 0) //如果删除后当前点的入度变为0，则将其入队
                q.push (j);
        }
    }
    if (an.size () == n) //如果拓扑序列的顶点数等于图的顶点数，那么就说明无环，否则则有环
        for (int i = 0; i < an.size (); i ++)
            cout << an[i] << ' ';
    else
        cout << "ERROR";
    cout << endl;
}

int main ()
{
    memset (h, -1, sizeof h); //初始化邻接表头数组
    cin >> n;
    for (int i = 0; i < n; i ++)
    {
        for (int j = 0; j < n; j ++)
        {
            int x;
            cin >> x;
            if (x) //如果x为1，说明i到j有一条边
            {
                add (i, j);
                d[j] ++; //点j的入度加1
            }
        }
    }
    topsort ();
    return 0;
}
```