1.图的存储与遍历

引例 1：犯罪团伙

【问题描述】

警察抓到了 n 个罪犯，警察根据经验知道他们属于不同的犯罪团伙，却不能判断有多少个团伙，但通过警察的审讯，知道其中的一些罪犯之间相互认识，已知同一犯罪团伙的成员之间直接或间接认识。有可能一个犯罪团伙只有一个人。请你根据已知罪犯之间的关系，确定犯罪团伙的数量。已知罪犯的编号从 1 至 n。

【输入】

第一行：n（<=10000,人数），

第二行：m（<=100000，信息）

以下 m 行：每行两个数：i 和 j，中间一个空格隔开，表示 i 和 j 相互认识。

【输出】

公司的数量。

【输入输出样例】

group.in	group.out
11 9 1 2 4 5 3 4 1 3 5 6 7 10 5 10 6 10 8 9	3

【数据规模】

100%个数据：n<=10000，　m<=100000。

分析：

把 n 个人看成 n 个独立的点，如果 i 和 j 相互认识，在点 i 和 j 之间连一条没有方向的边。样例如下：

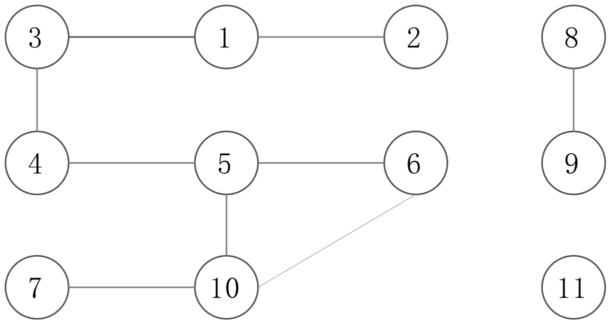


图 1

根据上图，很容易看有 3 个犯罪团伙，对应图 1 中的 3 个独立子图。

问题变为：如何保存这个图？如何求该图由几独的子图构成？

引例 2：安排座位

【问题描述】

已知 n(<20)个人围着一个圆桌吃饭，其中每一个人都至少认识其他的 2 个客人。请设计程序求得 n 个人的一种坐法，使得每个人都认识他左右的客人。

【输入】

第一行:n（吃饭人的个数）。

以下 n 行：第 i 行的第一个数 k 表示第 i 个人认识的人数，后面 k 个数依次为 i 认识的人的编号。

【输出】

所有座法，要求第一个人为 1 号作为起点，按顺时针输出其它人的编号。

【输入输出样例】

seat.in	seat.out
6 2 3 6 3 4 5 6 3 1 4 6 3 2 3 5 3 2 4 6 4 1 2 3 5	1 3 4 2 5 6 1 3 4 5 2 6 1 6 2 5 4 3 1 6 5 2 4 3 4

分析：

类似引例 1，把每个人看作一个顶点，相互认识的两个人用一条无方向的边连接。样例建如下所示的图：

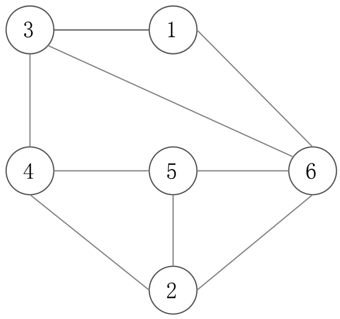


图 2

根据上述图 2，容易找出吃饭的其中一种座法，如：1 3 4 2 5 6。

以上两个引例问题的解决方法：

把每个人看成一个顶点，有关系的两个人连一条边，构成图的结构。

引例 1 的问题变为：求图由几个独立的子图构成。

引例 2 的问题变为：从其中的顶点 1 出发，沿着边经过所有的点走一遍（不能重复走点），最后回到 1。

解决上述问题需要掌握图的两个最基本的问题：图的存储方法与图的遍历。

1.图的概念

无向图与有向图：

图（Graph）是由顶点集合和顶点间的关系集合（边集）组成的数据结构，通常用二元组 G(V,E)表示图，V 表示顶点集，顶点元素经常用 u，v 等符合表示，顶点的个数通常用 n 表示；E 表示边的集合，边的元素经常用 e 等符号表示，边的数量通常用 m 表示。

如图 2：顶点集 V={1,2,3,4,5,6}，边集 E={(1,3),(1,6),(2,4),(2,5),(2,6),(3,4),(3,6),(4,5),(5,6)}。

在边集 E 中，每个元素(u,v)是两个顶点组成的无序对（用圆括号括起来），表示顶点 u 和 v 相关联的一条无向边。（u,v）和（v,u）表示同一条边。如果图中所有的边都没有方向，这种图称为**无向图**。

下列图 3 的表示：V={1,2,3},E={<1,2>,<1,3>,<2,3>}。其中 E 的每个元素<u,v>是一对顶点的有序对（用尖括号括起来），表示从顶点 u 到顶点 v 的一条有向边，u 是起点，v 是终点，所以<u,v>和<v,u>不是同一条边。如果图中所有的边都有方向的，这种图称为**有向图**。

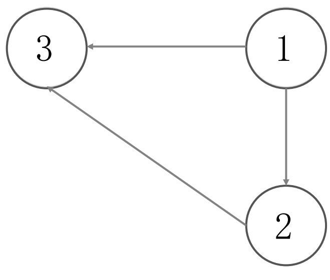


图 3

2.图的存储方法

一个图需要保存两个信息，一个是顶点信息，一个是顶点间的信息（边）。

顶点信息：一维数组即可。如果是 1 到 n，无需保存，直接用 1 到 n 表示就行了。

顶点间关系的存储常用的方法有两种：邻接矩阵与邻接表，这里只介绍简单的邻接矩阵。

顶点间的关系，用矩阵表示，称为**邻接矩阵**。

设 $G(V, E)$是一个具有 n 个顶点的图，则图的邻接矩阵是一个 $n\times n$ 的二维数组，定义为：

$$a[i][j]=\begin{cases}1 & 如果<i,j>\in E，或(i,j)\in E\\ & 0\ 否则\end{cases}$$

图 2 的邻接矩阵：

$$a[6][6]=\begin{bmatrix}0&0&1&0&0&1\\0&0&0&1&1&1\\1&0&0&1&0&1\\0&1&1&0&1&0\\0&1&0&1&0&1\\1&1&1&0&1&0\end{bmatrix}$$

图 3 的邻接矩阵：

$$a[3][3]=\begin{bmatrix}0&1&1\\0&0&1\\0&0&0\end{bmatrix}$$

3.图的遍历

图的遍历是图论算法的基础，图的遍历（graph traversal），也称图的搜索（ search），就是从图中某个顶点出发，沿着一些边访遍图中所有的顶点，且使每个顶点仅被访问一次。

图的遍历可以采取两种方法进行：深度优先搜索（ DFS： depth first search）和广度优先搜索（ BFS：breadth first search）。

（1）DFS 遍历

深度优先搜索是一个递归过程，有回退过程。

DFS 算法思想：

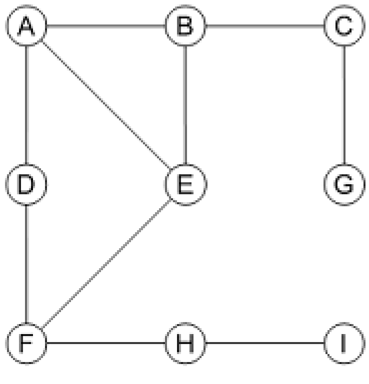
对于一个连通图，在访问图中某一起始顶点 u 后，由 u 出发，访问它的某一邻接顶点 v1；再从 v1 出发，访问与 v1 邻接但还没有访问过的顶点 v2；然后再从 v2 出发，进行类似的访问；…；如此进行下去，直至到达所有邻接顶点都被访问过为止；接着，回退一步，回退到前一次刚访问过的顶点 x，看是否还有 x 的其它没有被访问过的邻接顶点 y，如果有，则访问此顶点 y，之后再从此顶点 y 出发，进行与前述类似的访问；如果没有，就再回退一步进行类似的访问。重复上述过程，直到该连通图中所有顶点都被访问过为止。

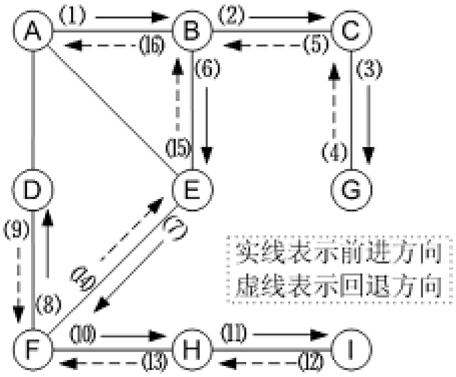
如下列(a)无向图 G：

顶点数组：

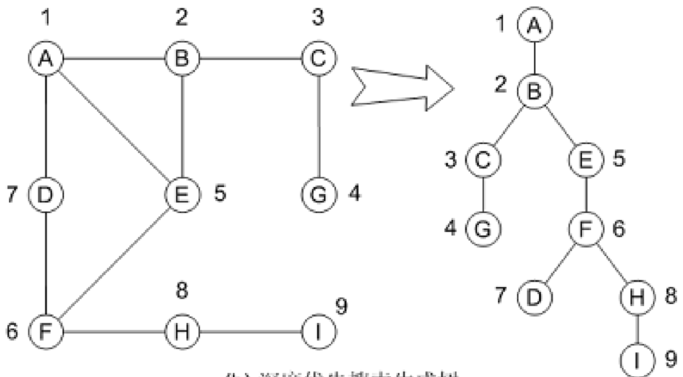
1	2	3	4	5	6	7	8	9
A	B	C	D	E	F	G	H	I

假设在多个未被访问的邻接顶点中进行选择时，按顶点序号从小到大的顺序进行选择。选择从编号最小的 A 开始：


(a) 无向图G



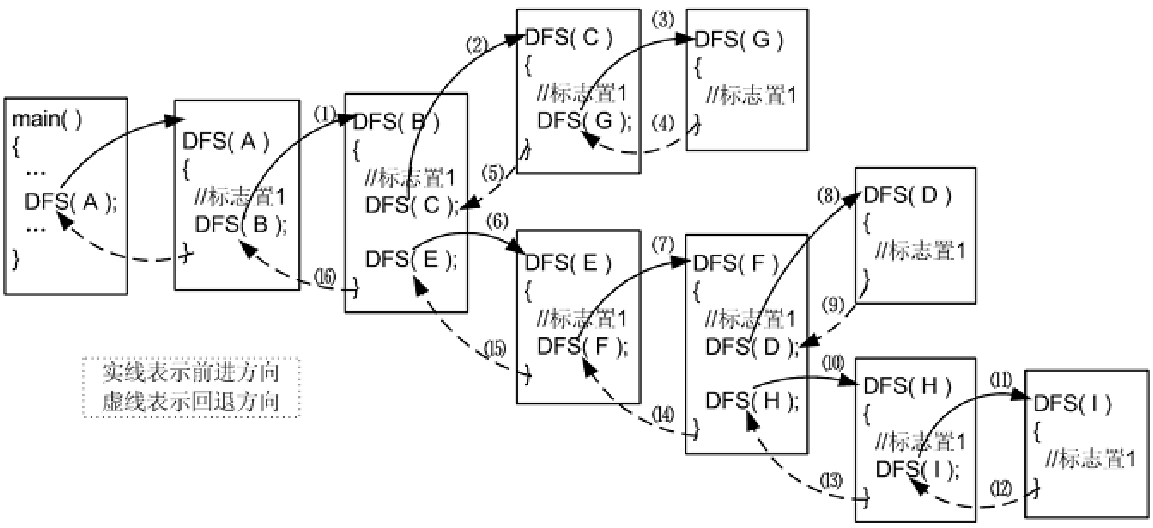
(a) 深度优先过程


(b) 深度优先搜索生成树

每个顶点外侧的数字标明了进行深度优先搜索时各顶点访问的次序，称为顶点的深度优先数。图(b)给出了访问 n 个顶点时经过的 n-1 条边，这 n-1 条边将 n 个顶点连接成一棵树，称此图为原图的深度优先生成树，该树的根结点就是深度优先搜索的起始顶点。

上图 DFS 遍历的过程（递归实现）:

为避免重复访问，需要一个状态数组 visited[n]，用来存储各顶点的访问状态。如果 visited[i] = 1，则表示顶点 i 已经访问过；如果 visited[i] = 0，则表示顶点 i 还未访问过。初始时，各顶点的访问状态均为 0。



DFS 算法框架：

DFS（ 顶点 u ） //从顶点 i 进行深度优先搜索{

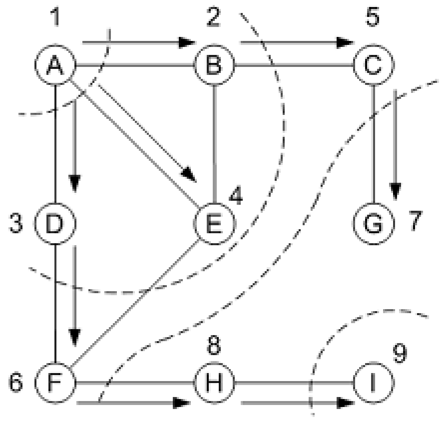
```
        //输出结点 u
        vis[ u ] = 1; //将顶点 i 的访问标志置为 1
        for( v=1; v<=n; v++ ) //对其他所有顶点 j{
            //v 是 u 的邻接顶点，且顶点 v 没有访问过
            if( a[u][v]==1 && !vis[v] ){
                //递归搜索前的准备工作需要在这里写代码
                DFS( v ) //从顶点 v 出发进行 DFS 搜索
                //以下是 DFS 的回退位置，在很多应用中需要在这里写代码
            }
        }
    }
```

（1）BFS 遍历

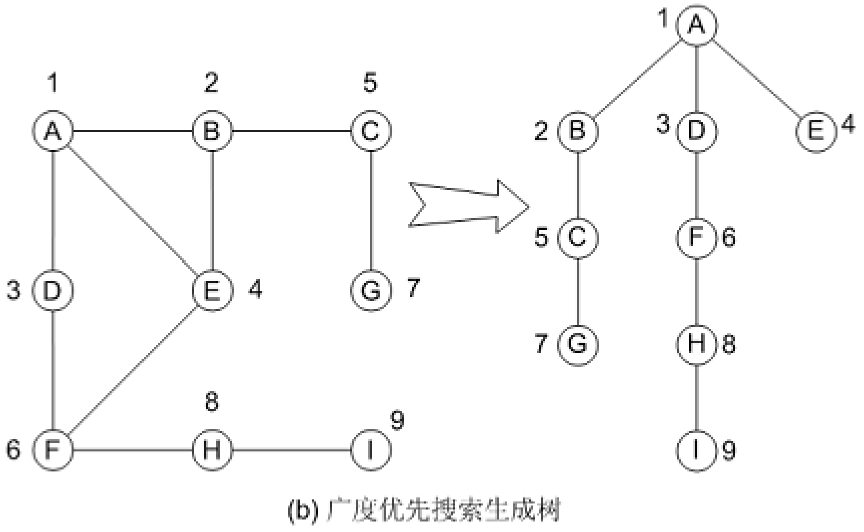
广度优先搜索（ BFS， Breadth First Search）是一个分层的搜索过程，没有回退过程，是非递归的。

BFS 算法思想:

对一个连通图，在访问图中某一起始顶点 u 后，由 u 出发，依次访问 u 的所有未访问过的邻接顶点 v1, v2, v3, …vt；然后再顺序访问 v1, v2, v3, …vt 的所有还未访问过的邻接顶点；再从这些访问过的顶点出发，再访问它们的所有还未访问过的邻接顶点，……，如此直到图中所有顶点都被访问到为止。



(a) 广度优先过程

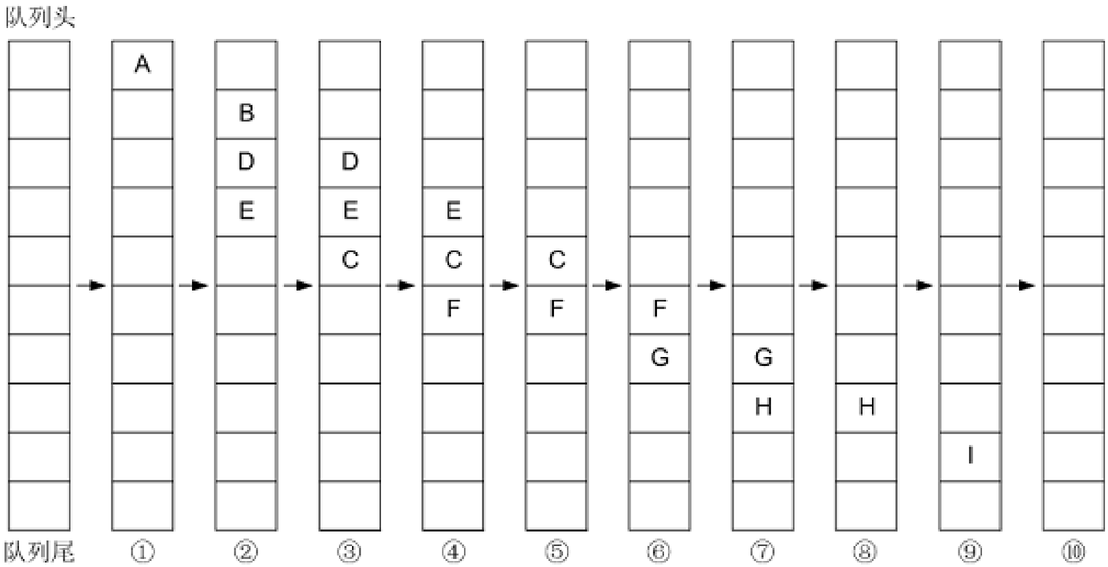


(b) 广度优先搜索生成树

BFS 算法的实现：

与深度优先搜索过程一样，为避免重复访问，也需要一个状态数组 visited[n]，用来存储各顶点的访问状态。如果 visited[i] = 1，则表示顶点 i 已经访问过；如果 visited[i] = 0，则表示顶点 i 还未访问过。初始时，各顶点的访问状态均为 0。

为了实现逐层访问， BFS 算法在实现时需要使用一个队列，来记忆正在访问的这一层和上一层的顶点，以便于向下一层访问（约定在队列中，取出元素的一端为队列头，插入元素的一端为队列尾,初始时，队列为空）：



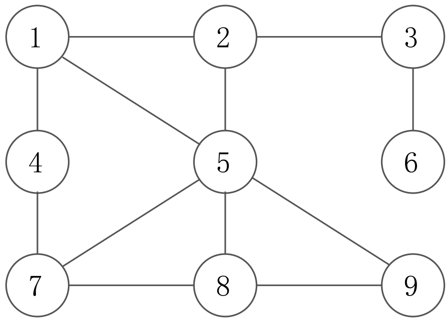
BFS 算法：

```
BFS( 顶点 s ) //从顶点 i 进行广度优先搜索{
    //输出结点 s
    vis[ s ] = 1; //将顶点 s 的访问标志置为 1
    将顶点 s 入队列;
    while( 队列不为空 ){
        取出队列头的顶点，设为 u
        for( v=1; v<=n; v++ ) //对其他所有顶点 v{
            //v 是 u 的邻接顶点，且顶点 v 没有访问过
            if( a[u][v]==1 && !vis[v] ){
                将顶点 v 的访问标志置为 1
                将顶点 v 入队列
            }
        } //end of for
    } //end of while
} //end of BFS
```

【上机实践】

例 1.图的遍历

对下图进行存储（邻接矩阵）和遍历（用 DFS 和 BFS 分别实现）。



输入：
第一行：顶点数 n。
第二行：边数 m。
以下 m 行，每行代表一条边的两个顶点编号 u，v。
输入输出样例：
9
12
1 2
1 4
1 5
2 3
2 5
3 6
4 7
5 7
5 8
5 9
7 8
8 9
参考代码：
DFS 算法：
```
#include<cstdio>
#include<iostream>
const int maxn=1001;
int a[maxn][maxn]={0};
int vis[maxn]={0};
int n,m;
using namespace std;
void dfs(int u){
    vis[u]=1;
    cout<<u<<" ";
    for(int v=1;v<=n;v++)
        if(a[u][v]==1&&vis[v]==0)dfs(v);
}
int main(){
    cin>>n>>m;
```

```
    for(int i=0;i<m;i++){
        int u,v;
        cin>>u>>v;
        a[u][v]=a[v][u]=1;
    }
    dfs(1);
    return 0;
}
```

BFS 算法：

```
#include<cstdio>
#include<iostream>
using namespace std;
const int maxn=1001;
int a[maxn][maxn]={0};
int vis[maxn]={0};
int q[maxn];
int n,m;
void bfs(int s){
    vis[s]=1;
    q[0]=s;
    int head=0,tail=1;
    while(head<tail){
        int u=q[head++];
        cout<<u<<" ";
        for(int v=1;v<=n;v++)
            if(a[u][v]==1&&vis[v]==0){
                vis[v]=1;
                q[tail++]=v;
            };
    }
}
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v;
        cin>>u>>v;
        a[u][v]=a[v][u]=1;
    }
    bfs(1);
    return 0;
}
```

说明：上述是对一个连通图（图中任意两点都有路线可达）的遍历方法，可以从任意一个顶点作为起点开始 dfs 或 bfs 都能完成整个连通图的遍历。

4.无向图的连通分量

如果一个图是非连通图，那么一次 dfs 或 bfs 只能遍历一个连通分量。非连通图中极大连通子图的数量称为无向图的连通分量。

求图的连通分量的方法很简单：每次找一个没有遍历的顶点 i 作为起点 dfs(i)即可，调用的次数就是连通分量。

求无向图的连通分量算法：

```
for(int i=1;i<=n;i++)
    if(!vis[i]){
        dfs(i);//或 bfs(i)
        cnt++;//连通分量
    }
```

图的遍历是图的算法的基础，掌握了图的遍历后，在此基础上稍加变化，能解决很实际问题。

例 2.犯罪团伙

以人为图的顶点，相互认识的建立无向边，求无向图的连通分量。

参考代码：

```
#include<cstdio>
#include<iostream>
using namespace std;
int a[10001][10001];
int vis[10001];
int n,m,cnt=0;
void dfs(int u){
    vis[u]=1;
    for(int v=1;v<=n;v++)
        if(a[u][v]&&!vis[v])dfs(v);
}
int main(){
    scanf("%d%d",&n,&m);
    for(int i=1;i<=m;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        a[u][v]=a[v][u]=1;
    }
    cnt=0;
    for(int i=1;i<=n;i++)
        if(!vis[i]){
            cnt++;
            dfs(i);
        }
    cout<<cnt<<endl;
    return 0;
```

}

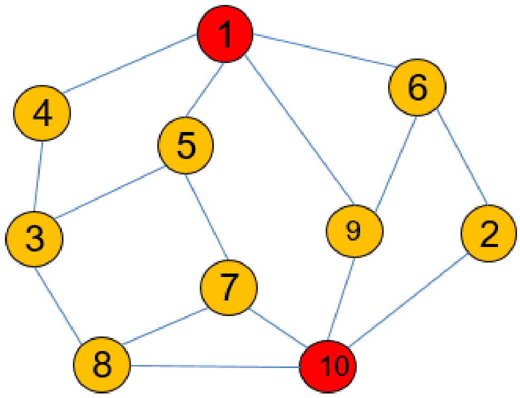
5.图中两点间的最少边数（边长为 1 的最短距离）

例 3.最少换乘次数

给定 n(<=100)个城市及城市间的交通路线（双向），每列火车只能在固定的两个城市间运行，也就是说从城市 A 到城市 B，如果中间经过城市 C，则从 A 到 C 后，必须在 C 处换乘另一辆火车才能到达 B。

求从 1 号城市到 n 号城市的最少的换乘次数。

从 1 到 10，最少换乘 1 次。



输入：

```
10
15
1 4
1 5
1 9
1 6
4 3
5 3
5 7
9 10
6 9
6 2
3 8
7 8
7 10
2 10
8 10
```

输出：

```
1
```

分析：

求一条从 1 到 n 的路线，要求中间经过的点最少或者是最少边数-1。

如果用 dfs，需要找出所有路线比较，找最短的。

如果采用 bfs 找到即可，无需比较。

```
#include<cstdio>
#include<iostream>
using namespace std;
struct node{
    int x;   //存顶点
    int dep; //存深度：x 点所在的层数
};//结构体
node q[101];
int vis[101]={0};
int a[101][101];
int n,m;
int bfs(){
    int head=0,tail=1;
    q[0].x=1;
    q[0].dep=0;
    vis[1]=1;
    while(head<tail){
        node p=q[head++];
        int u=p.x;
        if(u==n) return p.dep-1;
        for(int v=1;v<=n;v++)
            if(a[u][v]==1&&!vis[v]){
                vis[v]=1;
                q[tail].x=v;
                q[tail].dep=p.dep+1;
                tail++;
            }
    }
}
int main(){
    cin>>n>>m;
    for(int i=0;i<m;i++){
        int u,v;
        cin>>u>>v;
        a[u][v]=a[u][v]=1;
    }
    cout<<bfs()<<endl;
    return 0;
}
```

【课后训练】

1.油田(zoj1709/poj1562/uva572)

题目描述：

GeoSurvComp 地质探测公司负责探测地下油田。每次 GeoSurvComp 公司都是在一块长方形的土地上来探测油田。在探测时，他们把这块土地用网格分成若干个小方块，然后逐个分析每块土地，用探测设备探测地下是否有油田。方块土地底下有油田则称为 pocket，如果两个 pocket 相邻，则认为是同一块油田，油田可能覆盖多个 pocket。

你的工作是计算长方形的土地上有多少个不同的油田。

输入描述：

输入文件中包含多个测试数据，每个测试数据描述了一个网格。每个网格数据的第一行为两个整数：m n，分别表示网格的行和列；如果 m = 0，则表示输入结束，否则 1≤m≤100，1 ≤n≤100。接下来有 m 行数据，每行数据有 n 个字符（不包括行结束符）。每个字符代表一个小方块，如果为“*”，则代表没有石油，如果为“@”，则代表有石油，是一个 pocket。

输出描述：

对输入文件中的每个网格，输出网格中不同的油田数目。如果两块不同的 pocket 在水平、垂直、或者对角线方向上相邻，则被认为属于同一块油田。每块油田所包含的 pocket 数目不会超过 100。

样例输入：

```
1 1
*
3 5
*@*@*
**@**
*@*@*
1 8
@@****@*
5 5
****@
*@@*@
*@**@
@@@*@
@@**@
0 0
```

样例输出：

```
0
1
2
2
```

分析：

从网格中某个“@”字符位置开始进行 DFS 搜索，可以搜索到跟该“@”字符位置同属一块油田的所有“@”字符位置。遍历整个图，求图的连通分量。注意是 8 连通。

种子填充：Floodfill

扩展：如何求最大油田的面积？

2.上学路线

【问题描述】

小 A 经过初三一年的努力，终于考上了认为适合自己的山师附中。

由于小 A 的家离学校很远，他又懒得骑自行车上学，于是他决定开学之前先考察一下上学路上的公交车路线，然后做出合适的选择，为新学期做好第一个准备。

经过几天的考察，小 A 记下了若干路公共汽车路线，这些公共汽车都是从某个站出发，沿途依次经过很多的中间站，最后到达终点站，并且都是单向的路线。

令小 A 感到非常不爽的是，他很难找到有一趟公共汽车能从他家直达学校，大部分是他需要先乘某一路汽车坐上几站，下来后再换乘同一站台的另一路汽车，这样换乘几次后才能到达学校。幸运的是他的家门口和学校门口都一个汽车站。

小 A 坐车最烦的就是等车，于是他决定选择一个换乘汽车次数最少的上学路线。

现在用整数 1,2,...,n 给所有的公共汽车站编号，约定小 A 的家门口的汽车站编号为 1，学校门口的汽车站的编号为 n。

你的任务是：帮助小 A 寻找一个最优乘车方案,使他从家乘车到学校的过程中换车的次数最少。

【输入】

第一行 M，表示 M 条单程公共汽车线路。

第二行 N，表示总共有 N 个车站。

以下 M 行依次给出了第 1 条到第 M 条汽车线路的信息。其中第 i 行给出的是第 i 条线路的信息，从左至右按运行顺序依次给出了该线路上的所有站号，相邻两个站号之间用一个空格隔开。

【输出】

一行。如果无法乘从家到学校，则输出"No Roud!"，否则输出你的程序所找到的最少换车次数，换车次数为 0 表示不需换车即可到直达•

【输入输出样例】

roud.in	roud.out
3 7 6 7 4 7 3 6 1 2 3 5	2

【输入输出样例解释】

乘车路线：1 2 3 6 7：换了 2 次车。

【数据范围限制】

30%的数据范围：　1<=M<=20；1<N<=100。

100%的数据范围：　1<=M<=100；1<N<=500。