图 论-1：图的存储与遍历

引例1.犯罪团伙

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,罪犯数量），

第二行：m（<100000，关系数量）

以下若干行：每行两个数：i 和j，中间一个空格隔开，表示罪犯i和罪犯j相互认识。

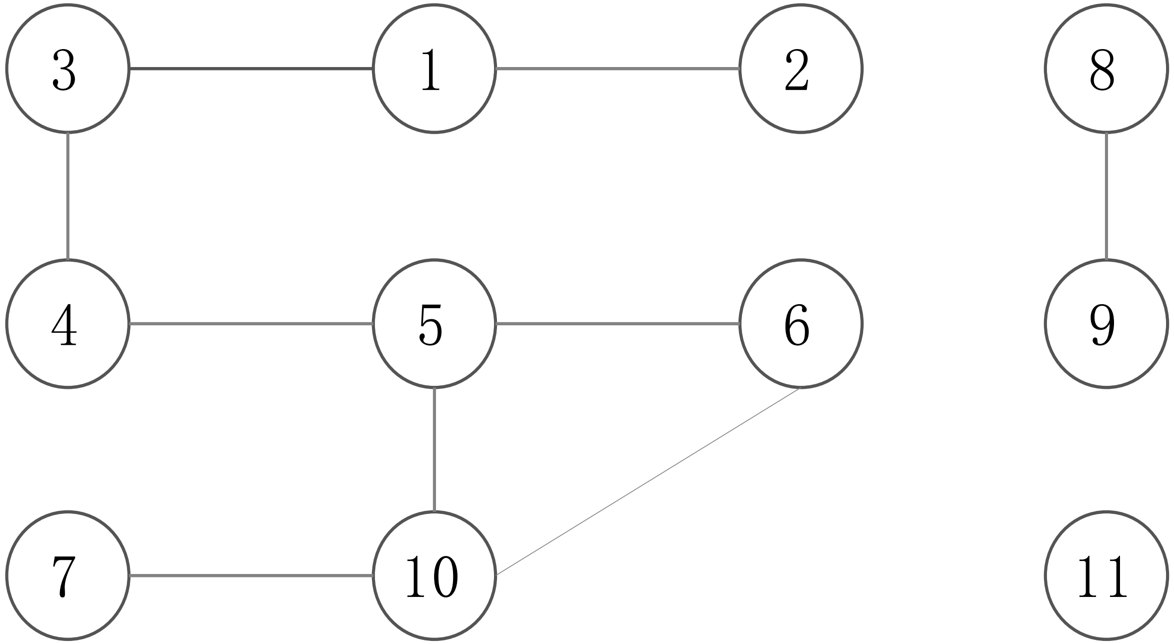
输出：

一个整数，犯罪团伙的数量。

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

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

方法：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。有3部分（图的连通分量）怎么保存？怎么求图的连通分量？



引例2：安排座位

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）。

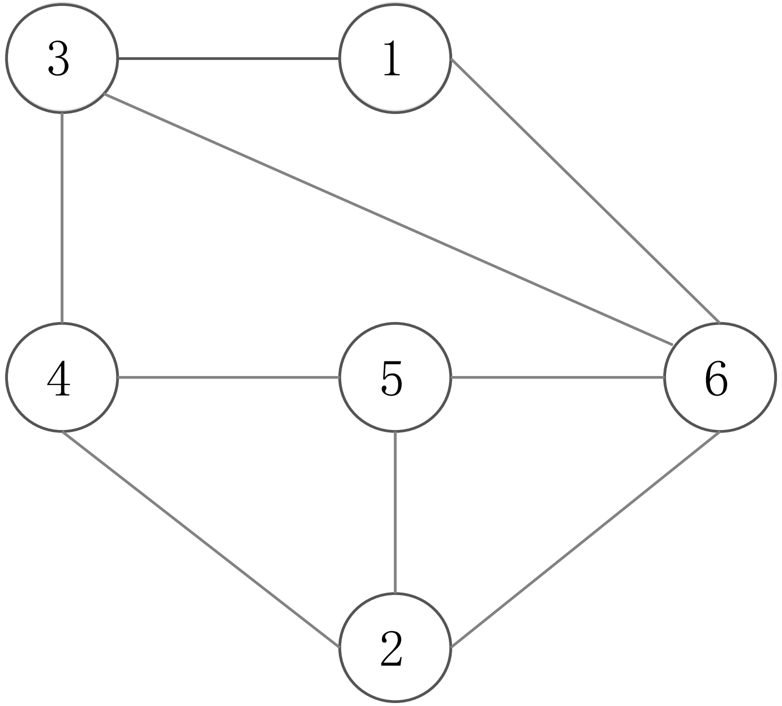
以下n行：第i行的第一个数k表示第i个人认识的人数，后面k个数依次为i认识的人的编号。

【输出】

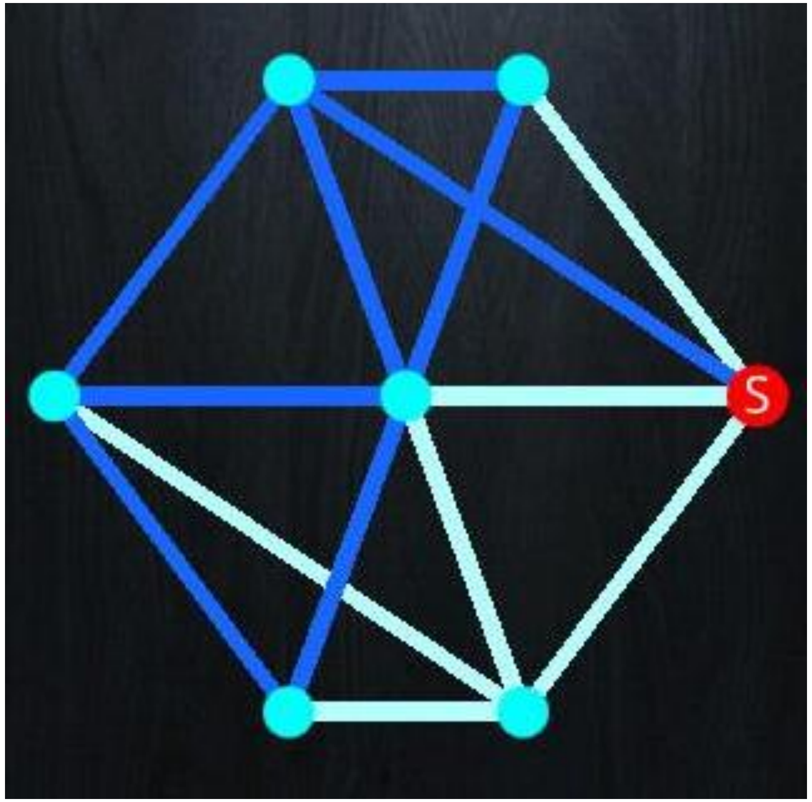
所有座法，要求第一个人为1号作为起点，按顺时针输出其它人的编号。

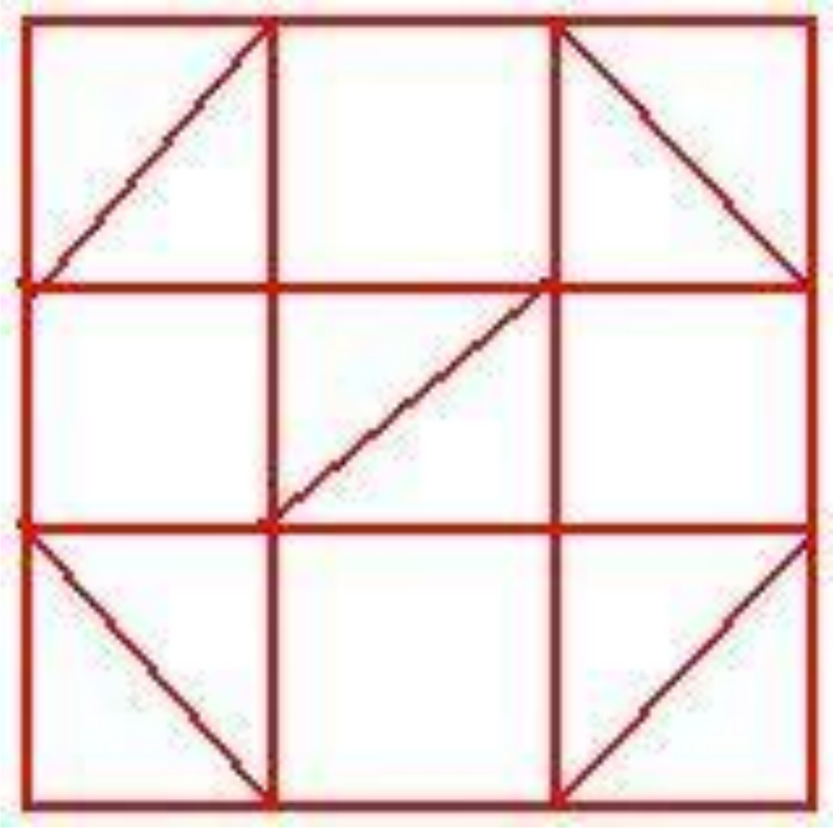
seat.in	seat.out
6	1 3 4 2 5 6
2 3 6	1 3 4 5 2 6
3 4 5 6	1 6 2 5 4 3
3 1 4 6	1 6 5 2 4 3
3 2 3 5	4
3 2 4 6	
4 1 2 3 5	

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。在图中找一个环，包含所有节点（哈密顿回路）。



引例3.一笔画问题





欧拉路问题

内容：

一. 图的基本概念

二. 图的存储结构

三. 图的遍历

四. 图的连通分量

五. 图中两点间的最短路（边数最少）

一、 图的基本概念

1. 图的的定义

图是由一个顶点的集合V和一个顶点间关系的集合E组成：

记　G=（V，E）

V：顶点的有限非空集合。

E：顶点间关系的有限集合（边集）。

存在一个结点v，可能含有多个前驱结点和后继结点（非线性结构）。

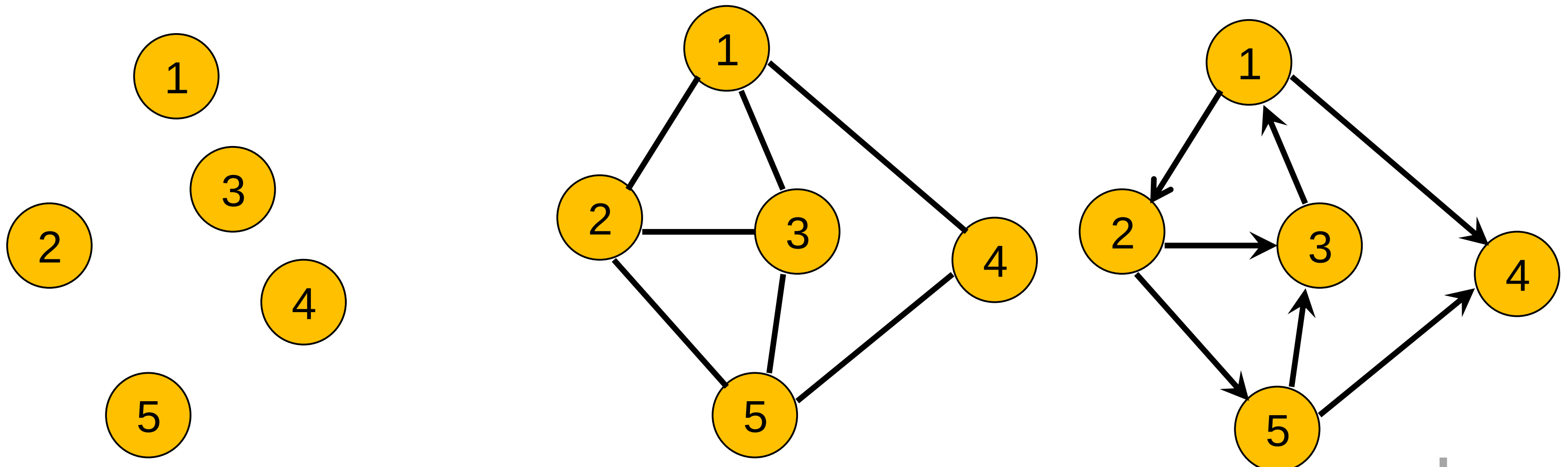


图1　　图2　　图3

2.无向图和有向图

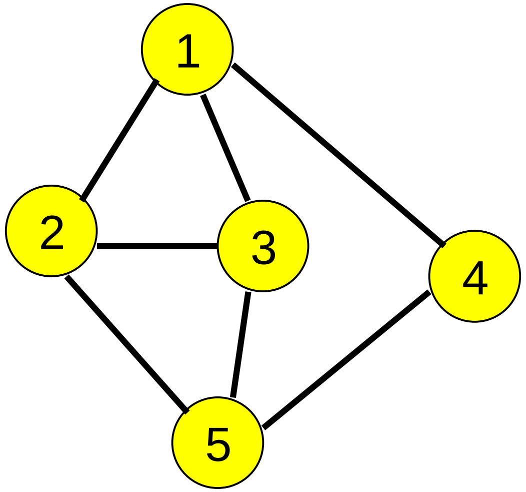
无向图：

在图G=（V，E）中，如果对于任意的顶点a，b∈V，

当(a，b)∈E时，必有（b，a）∈E（即关系R对称），此图称为无向图。

无向图中用不带箭头的边表示顶点的关系

V={1, 2, 3, 4, 5}
E={(1, 2),(1, 3),(1, 4),(2,3),(2,5),(3, 5),(4,5)}



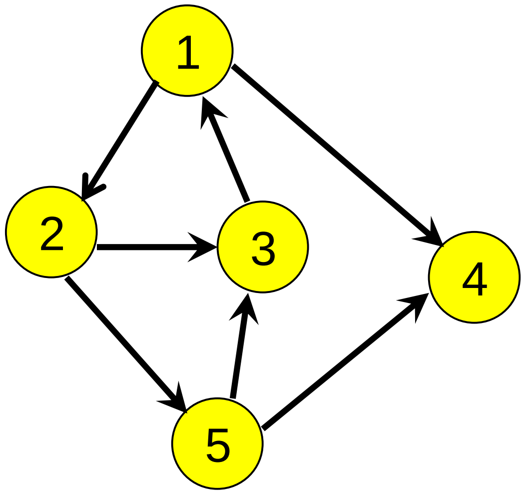
有向图:

如果对于任意的顶点a，b∈V，当(a，b)∈E时，(b，a)∈E未必成立，则称此图为有向图。

在有向图中，通常用带箭头的边连接两个有关联的结点。

$V=\{1, 2, 3, 4,5\}$

$E=\{<1, 2>, <1, 4>, <2, 3>, <2, 5>, <3, 1>, <5, 3>, <5, 4>\}$



3.顶点的度、入度和出度

在无向图中：顶点v的度是指与顶点v相连的边的数目D(v)。D(2)=3

在有向图中：
入度——以该顶点为终点的边的数目和 . ID(3)=2
出度——以该顶点为起点的边的数目和 . OD(3)=1
度数为奇数的顶点叫做奇点，度数为偶数的点叫做偶点。
度：等于该顶点的入度与出度之和。 D(5)=ID(5)+OD(5)=1+2=3

结论：图中所有顶点的度=边数的两倍

$$\sum_{i=1}^{n} D(v_i) = 2 * e$$

无向完全图 $e = \dfrac{n*(n-1)}{2}$

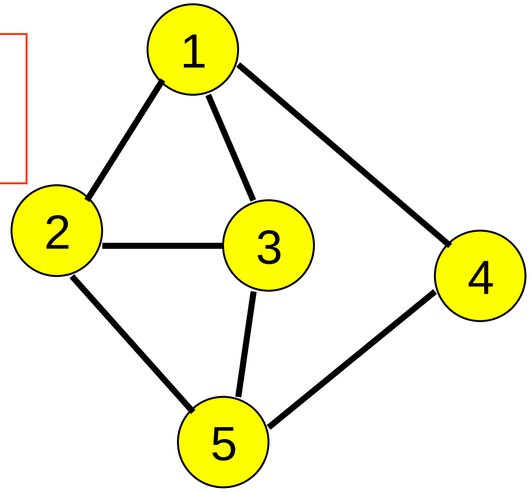


图1:无向图

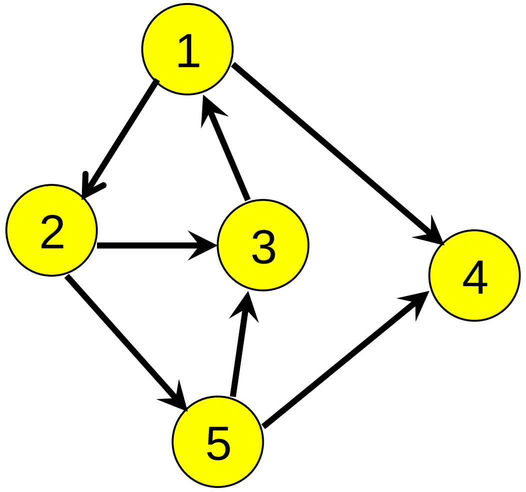


图2：有向图

4. 路径、简单路径、回路

在图G=（V，E）中，如果对于结点a，b，存在满足下述条件的结点序列 $x_1 \cdots\cdots x_k (k>1)$

(1) $x_1=a$，$x_k=b$　(2) $(x_i, x_{i+1}) \in E$　　$i=1 \cdot\cdot k-1$

则称结点序列 $x_1=a$，x_2，…，$x_k=b$ 为结点a到结点b的一条路径，而路径上边的数目（k-1）称为该路径的长度。

图1: 1:(1,2,3,5)　长度=3

2:(1,2,3,5,2)　长度=4

3:(1,2,5,4,1)　长度=4

图2: (1,2,5,4) 长度=3

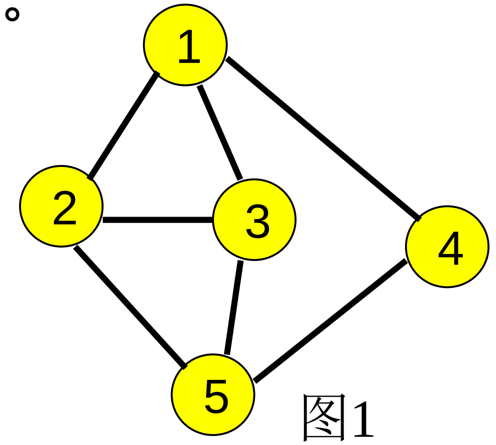


图1

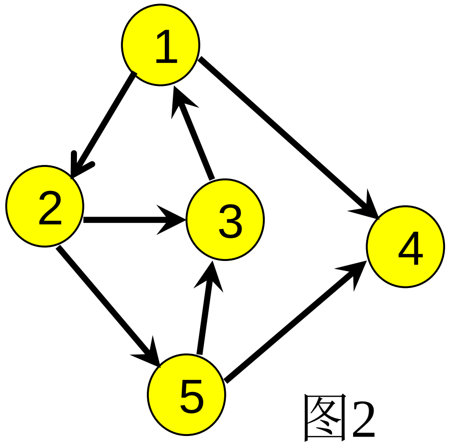


图2

（1）如果一条路径上的结点除起点x1和终点xk可以相同外，其它结点均不相同，则称此路径为一条简单路径。

（2）x1=xk的简单路径称为回路（也称为环）

5.连通、连通图、连通分量（无向图)

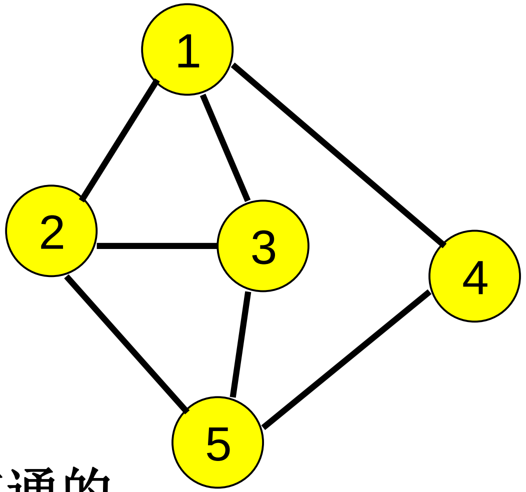
连通：“连成一片”。

连通图：“能连成一片的图”。

定义：

连通：如果存在一条从顶点u到v有路径，则称u和v是连通的。

连通图：图中任意的两个顶点u和v都是连通的，称为连通图。否则称为非连通图。



图一：连通图

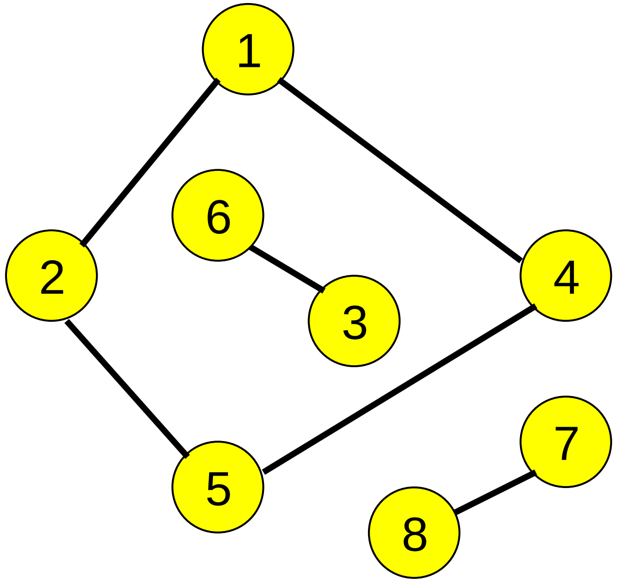
连通分量：无向图中的极大连通子图。

图二中有3个连通分量：

{1 2 4 5} {3 6} {7 8}

求连通分量数，最大连通分量等。

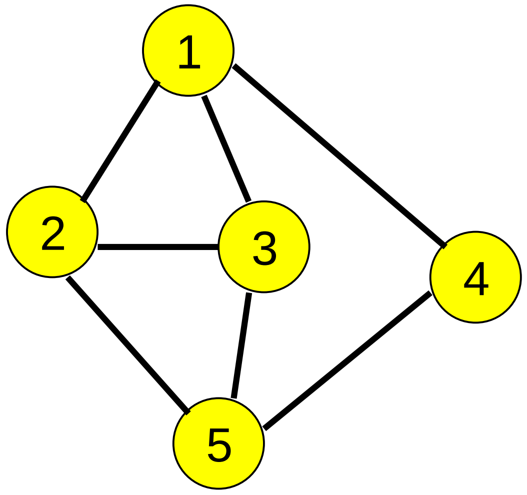
有向图：强连通、强连通图、强连通分量



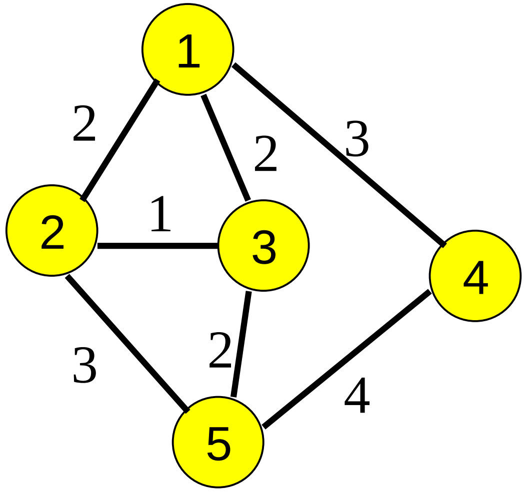
图二：非连通图

6、带权图

一般的图边上没有数字，边仅表示两个顶点的相连接关系。



图中的边可以加上表示某种含义的数值，数值称为边的权，此图称为带权图。也称为网。

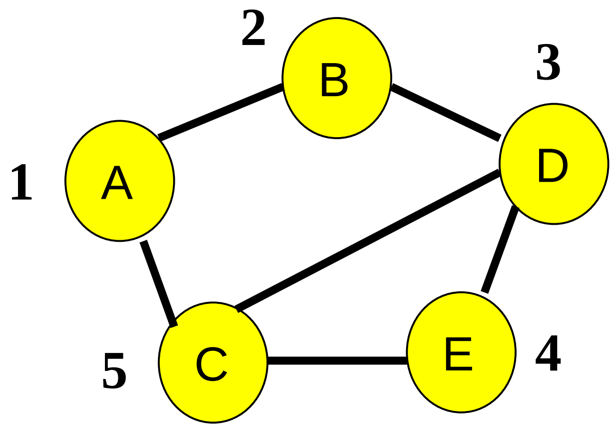


二.图的基本存储结构

G=(V,E).

顶点：数组或记录(大部分直接用编号1到n)

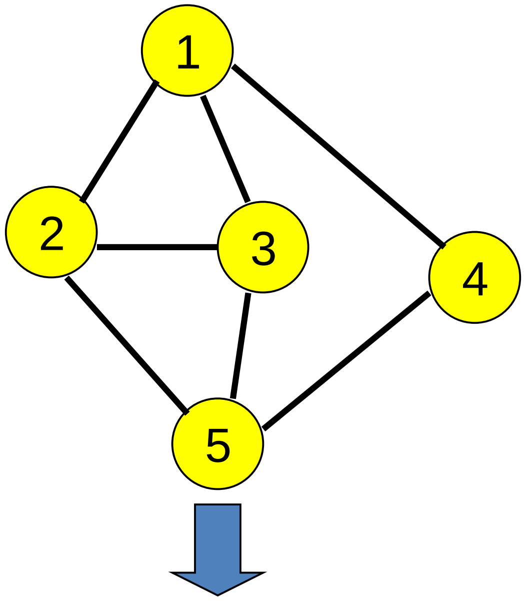
边：邻接矩阵与邻接表

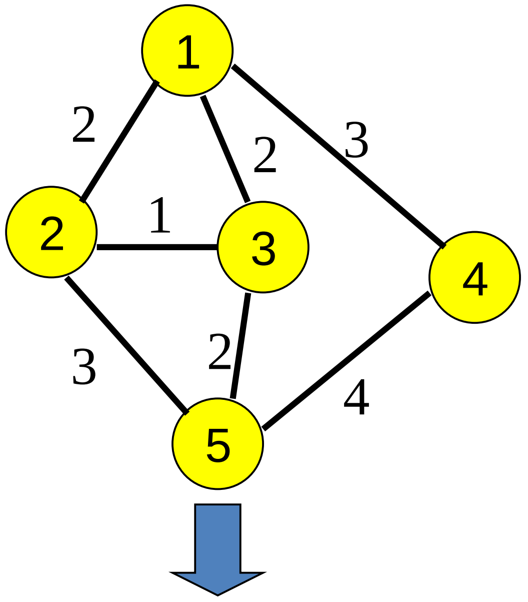


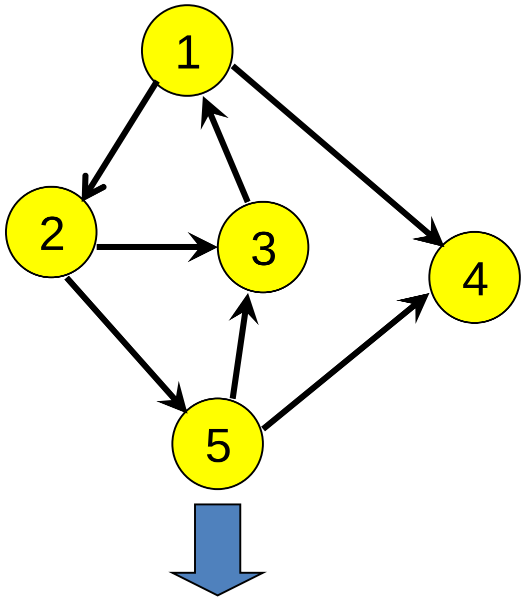
1、图的邻接矩阵表示法（二维数组）

邻接矩阵是表示结点间相邻关系的矩阵。若G=（V，E）是一个具有n个结点的图，则G的邻接矩阵是如下定义的二维数组g[n+1][n+1]。

$$g[i][j]=\begin{cases}1\ \text{(或权值)：无向图：有边}(i,j)\text{和边}(j,i)\\ \qquad\qquad\quad\text{有向图：有边}<i,j>\\ 0\text{或}\infty\text{：}i\text{到}j\text{无边或弧}\end{cases}$$







$$\begin{array}{c} \\ 1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{ccccc}1&2&3&4&5\end{array}\\ \boxed{\begin{array}{ccccc}0&1&1&1&0\\1&0&1&0&1\\1&1&0&0&1\\1&0&0&0&1\\0&1&1&1&0\end{array}}\end{array} \qquad \begin{array}{c} \\ 1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{ccccc}1&2&3&4&5\end{array}\\ \boxed{\begin{array}{ccccc}0&2&2&3&0\\2&0&1&0&3\\2&1&0&0&2\\3&0&0&0&4\\0&3&2&4&0\end{array}}\end{array} \qquad \begin{array}{c} \\ 1\\2\\3\\4\\5 \end{array}\begin{array}{c} \begin{array}{ccccc}1&2&3&4&5\end{array}\\ \boxed{\begin{array}{ccccc}0&1&0&1&0\\0&0&1&0&1\\1&0&0&0&0\\0&0&0&0&0\\0&0&1&1&0\end{array}}\end{array}$$

对角线为0：自身不相连。

无向图：是对称矩阵。有向图一般不是。

第i行非0 的个数是结点i的度

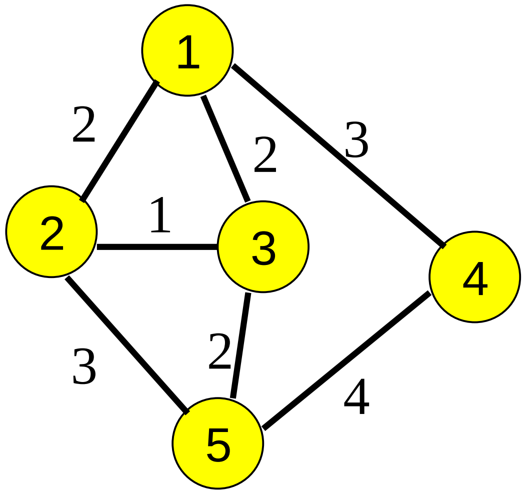
具体到题目时，边的关系的给出格式多种多样：

1）直接给出邻接矩阵，直接读即可。

如：

输入文件内容：

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

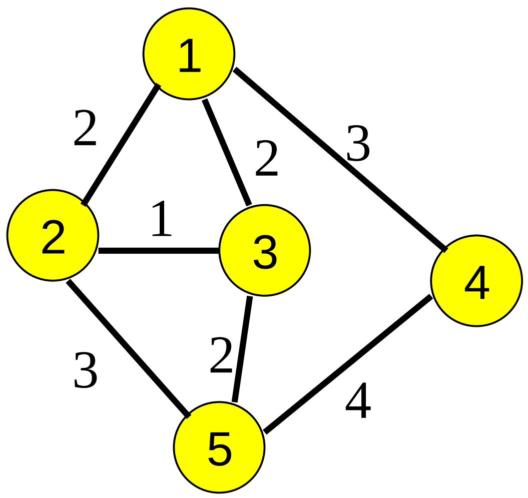



```
cin>>n;
for(int i=1;i<=n;i++)
    for(int j=1;j<=n;j++)
        cin>>a[i][j];
```

2）给出边的顶点。

如输入文件：**u,v,w**两个顶点及其边长

5
7
1 2 2
1 3 2
1 4 3
2 3 1
2 5 3
3 5 2
4 5 4

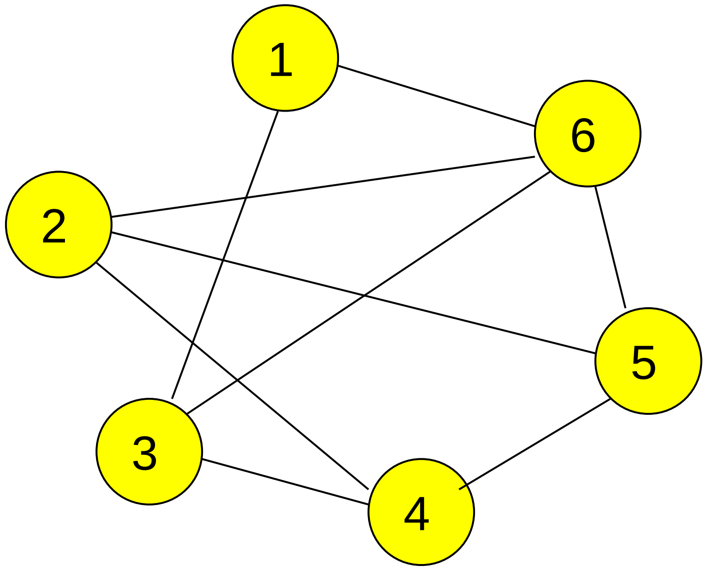



```
cin>>n>>m;
for(int i=0;i<m;i++){
    int u,v,w;
    cin>>u>>v>>w;
    a[u][v]=a[v][u]=w;
}
```

提示：
=赋值可以连续赋值
==判断相等不能连续写

3) 依次给出每个顶点的邻接点的个数及邻接点

```
cin>>n;
for(int i=1;i<=n;i++){
      int k;
      cin>>k;
      for(int j=1;j<=k;j++){
            int v;
            cin>>v;
            a[i][v]=1;   //有向图
            //a[i][v]=a[v][i]=1;   无向图
      }
}
```




```
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
```

初始化数组大可不必使用两重for循环。
#include<cstring>
如果是int数组：
memset(g， 0， sizeof(g))，全部清为0;
memset(g， 0x7f， sizeof(g))可全部初始化为一个很大的数(略小于0x7fffffff);
memset(g， 0x3f， sizeof(g))稍小
memset(g， 0x88， sizeof(g))负值很小
memset(g， 0xaf， sizeof(g))，负值稍大。
memset(g,0xff,sizeof(g)); 全部初始为-1

邻接矩阵的特点：

优点：好理解，容易判断结点i和j是否有边g[i][j]>0

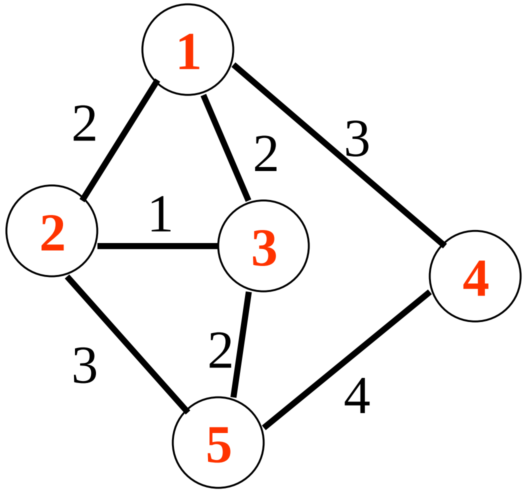
缺点：当结点n很大时，容易超内存；

并且找邻接点时间慢。

邻接矩阵的存储适合顶点n<=1000的图。

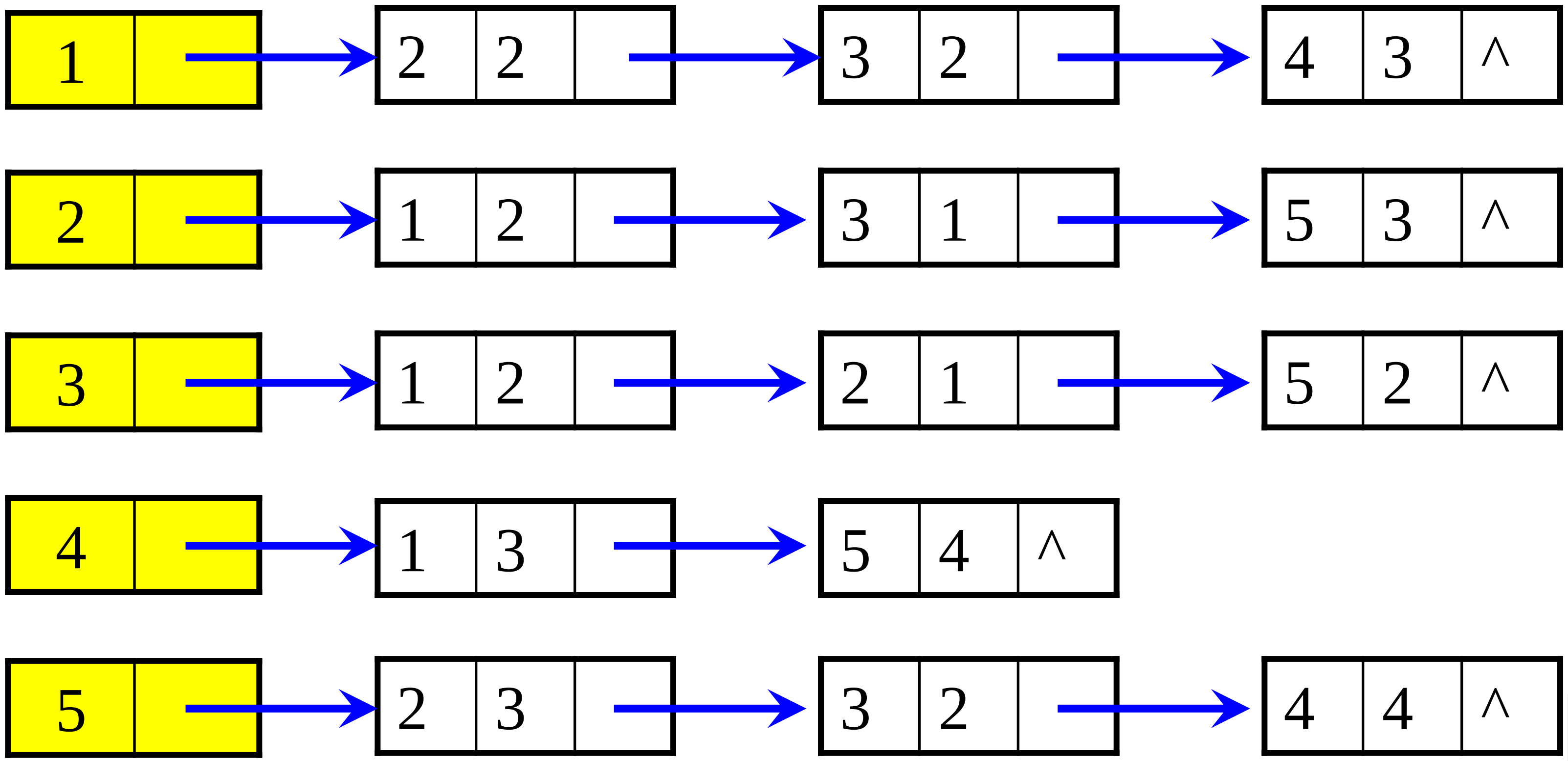
2.图的邻接表表示法

链式存储法



方法1：vector可变数组
方法2：数组模拟链表（后面讲最短路时再讲）

结点 **邻接点**



◆可变数组vector的使用：

- 常用可变数组vector：
- **Vectors 包含着一系列连续存储的元素,其行为和数组类似。**
- Vector<int>a; 定义了一个可变的一维数组a[?].容器a
- a.push_back(x); 再后面添加元素x；
- a.size();返回a的元素个数。
- 访问a的每一个元素a[i]：
- for(int i=0;i<a.size();i++)

  – a[i]

多叉树的存储：

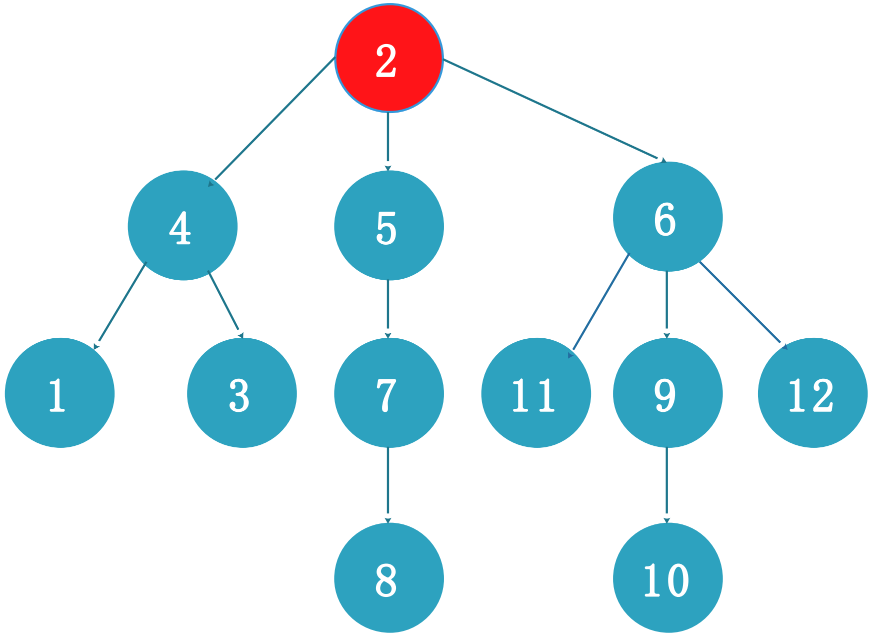
Vector<int>son[201];

定义了201个容器：a[0],a[1],..,a[200]

每个容器都是可变数组：相当于二维数组

a[i].push_back(x);

a[i][0],a[i][1]....



vector可变数组（同树中结点的儿子的存储方式一样）

```
vector<int>g[maxn];
cin>>u>>v;
g[u].push_back(v);
g[v].push_back(u);
u的邻接点个数m=g[u].size();
for(int i=0;i<m;i++){
        int v=g[u][i];
         ....
}
```

优点：当n很大时，节省内存空间，找一个结点的邻接点快速

有边权的图：

```
struct Edge{int v,w;};
vector<Edge>g[10001];

        int u,v,w;
        cin>>u>>v>>w;
        g[u].push_back((Edge){v,w});
        g[v].push_back((Edge){u,w});
```

三.图的遍历(图论算法的基础)

给出一个图G，从某一个初始点出发，按照一定的搜索方法对图中的每一个结点访问仅且访问一次的过程。

访问结点：处理结点的过程。如输出、查找结点的信息。

按照搜索方法的不同，通常有两种遍历方法：

1. 深度优先搜索dfs
2. 广度优先搜索bfs

1、深度优先搜索DFS

遍历算法（递归过程）：

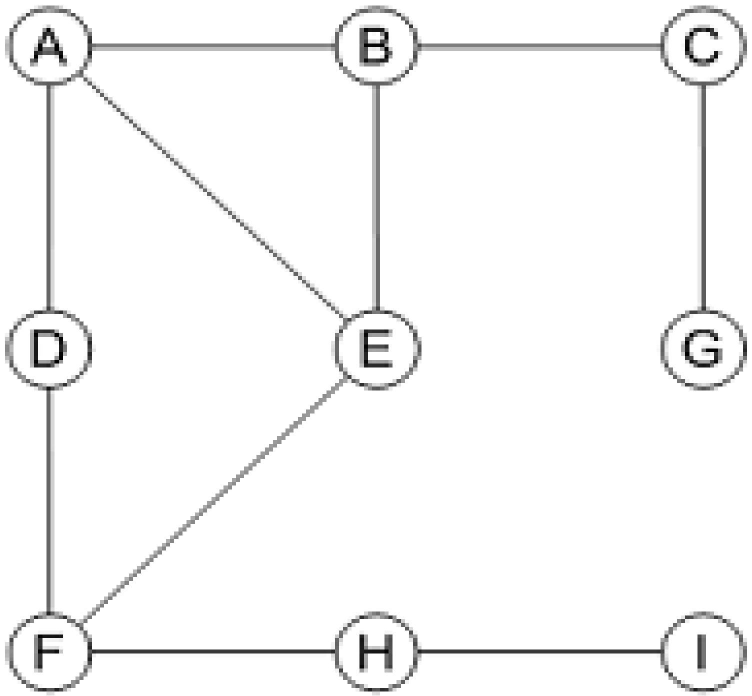
1)从某一初始出发点u开始访问：输出该点编号；并对该点作被访问标志（以免被重复访问）。

2)再从u的其中一个未被访问的邻接点v作为初始点出发继续深搜。

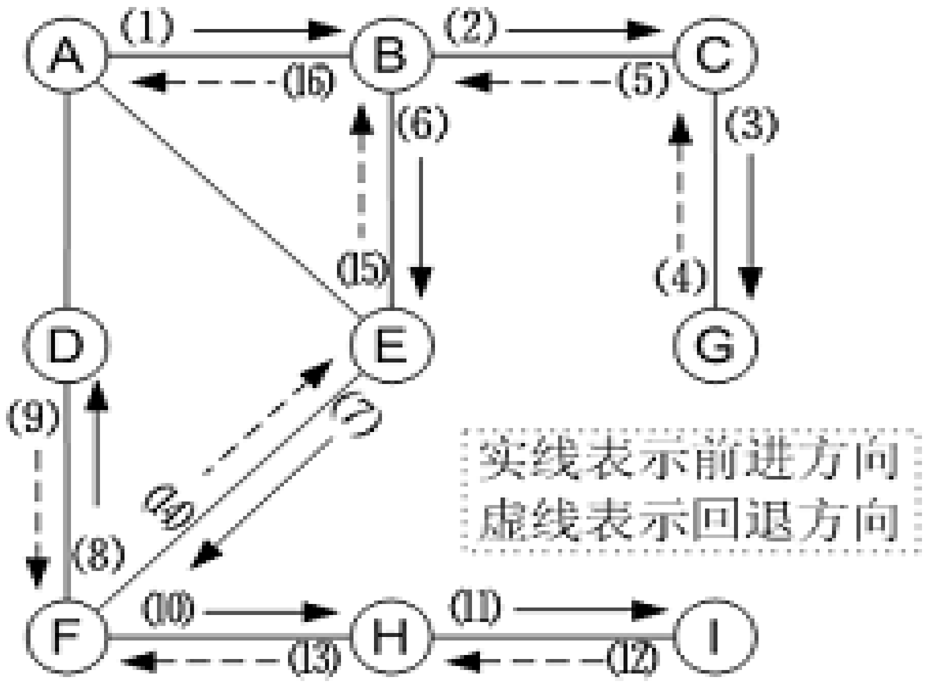
当i的所有邻接点都被访问完，则退回到u的父结点的另一个邻接点再继续深搜。

直到全部结点访问完毕

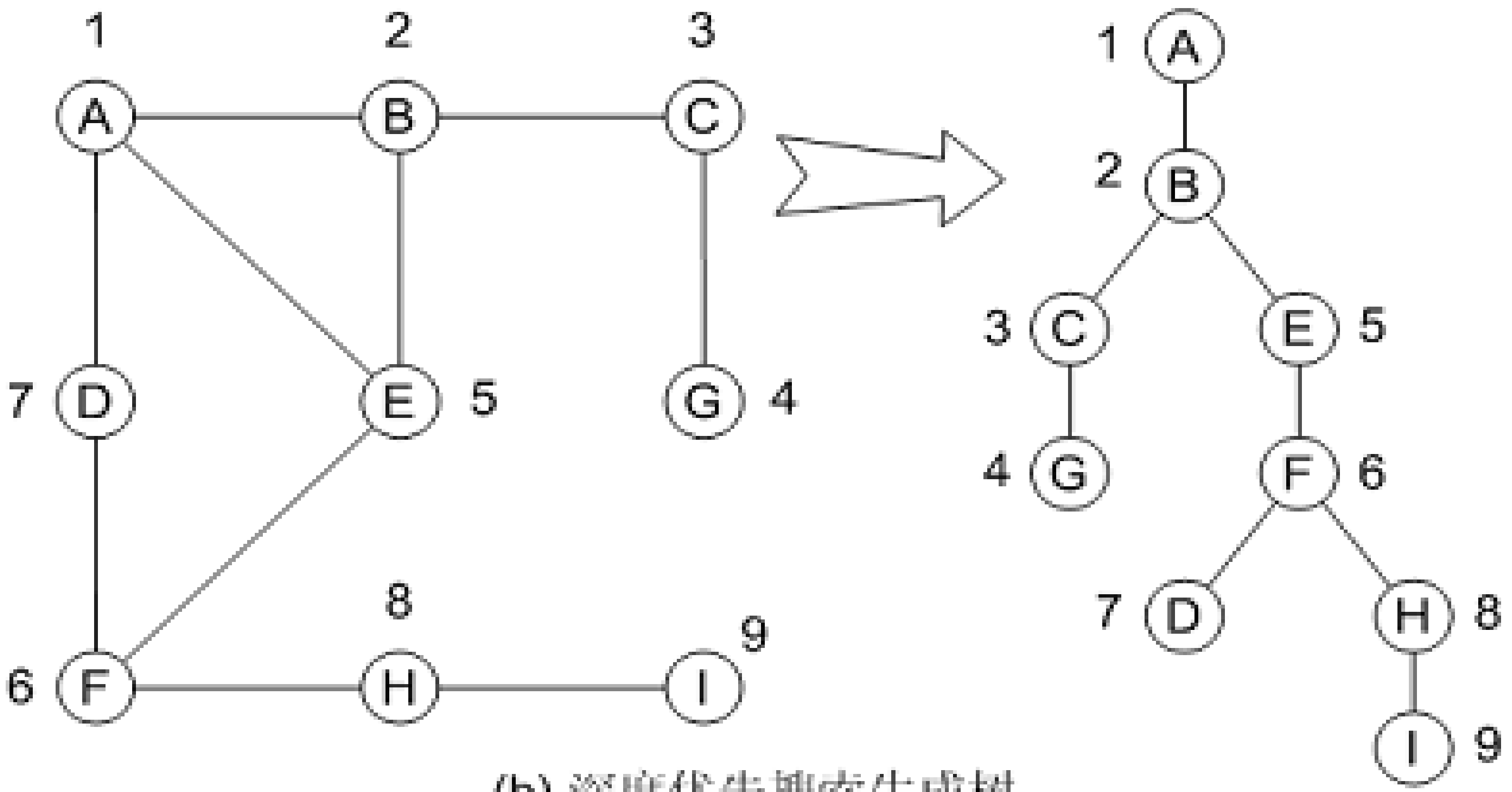
图的dfs遍历可以用递归实现，有回退过程



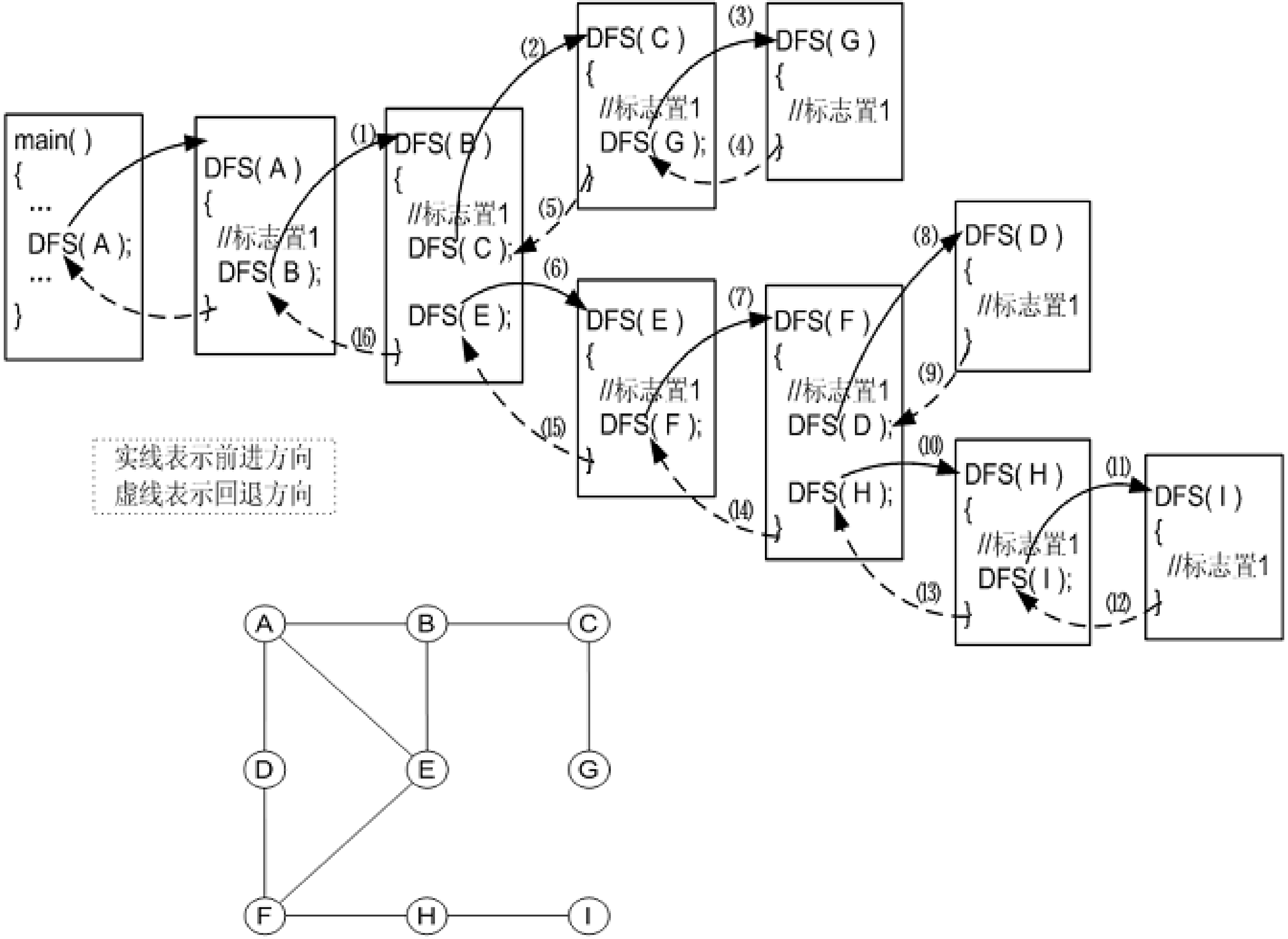
(a) 无向图G



(a) 深度优先过程



(b) 深度优先搜索生成树


main( )
{
...
DFS(A);
...
}
DFS(A)
{
//标志置1
DFS(B);
}
DFS(B)
{
//标志置1
DFS(C);
DFS(E);
}
DFS(C)
{
//标志置1
DFS(G);
}
DFS(G)
{
//标志置1
}
DFS(E)
{
//标志置1
DFS(F);
}
DFS(F)
{
//标志置1
DFS(D);
DFS(H);
}
DFS(D)
{
//标志置1
}
DFS(H)
{
//标志置1
DFS(I);
}
DFS(I)
{
//标志置1
}
(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16)
实线表示前进方向
虚线表示回退方向
A B C D E G F H I

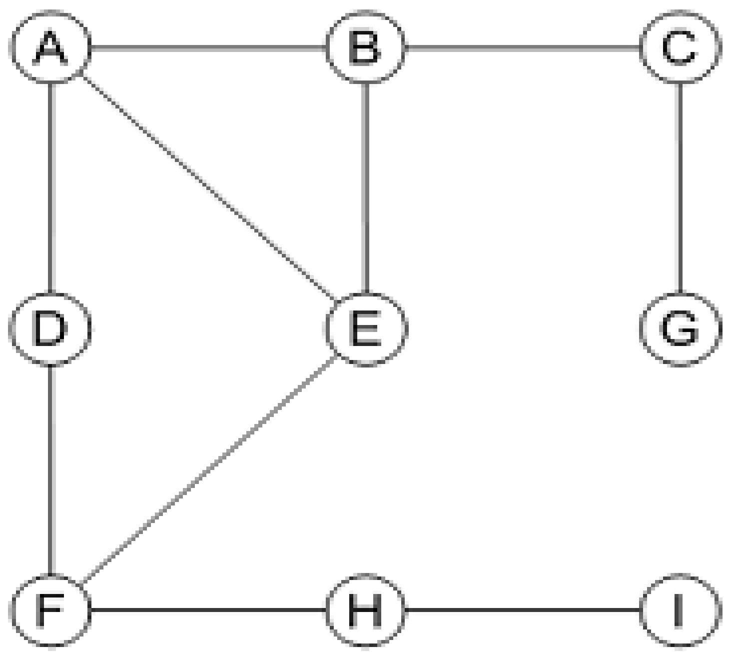
(a) 无向图*G*

实现:

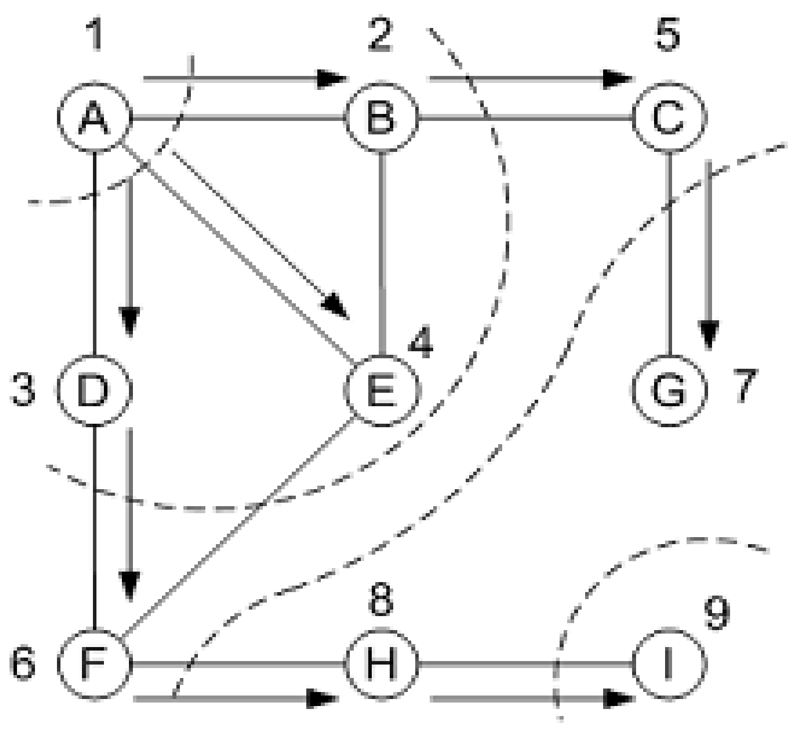
DFS 算法框架:

```
DFS( 顶点 u ) //从顶点 i 进行深度优先搜索{
    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
        }
    }
}
```

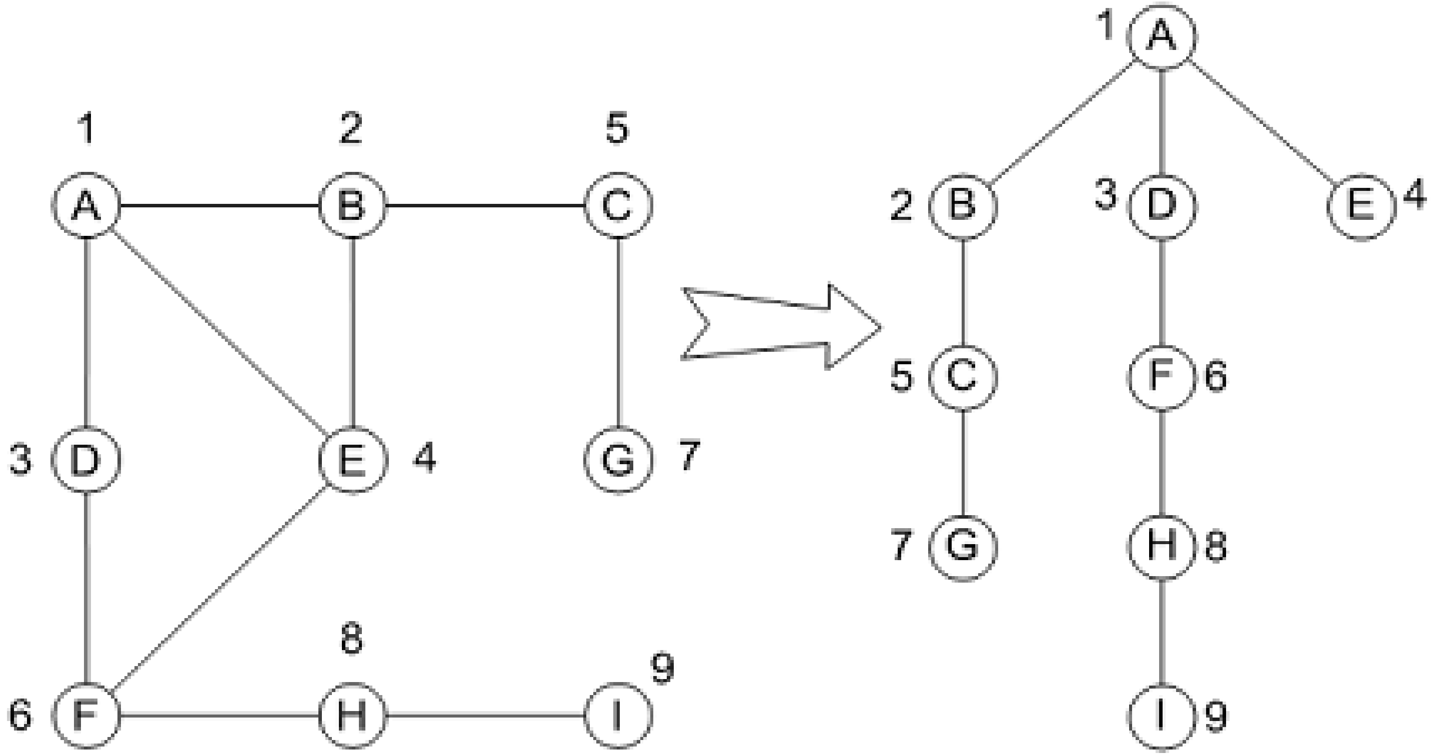
- **2.BFS 遍历**
- 广度优先搜索（ BFS， Breadth First Search）是一个分层的搜索过程，没有回退过程，是非递归的。
- BFS 算法思想:
- 对一个连通图，在访问图中某一起始顶点 u 后，由 u 出发，依次访问 u 的所有未访问过的邻接顶点 v1, v2, v3, ...vt；然后再顺序访问 v1, v2, v3, ...vt 的所有还未访问过的邻接顶点；再从这些访问过的顶点出发，再访问它们的所有还未访问过的邻接顶点，……，如此直到图中所有顶点都被访问到为止。



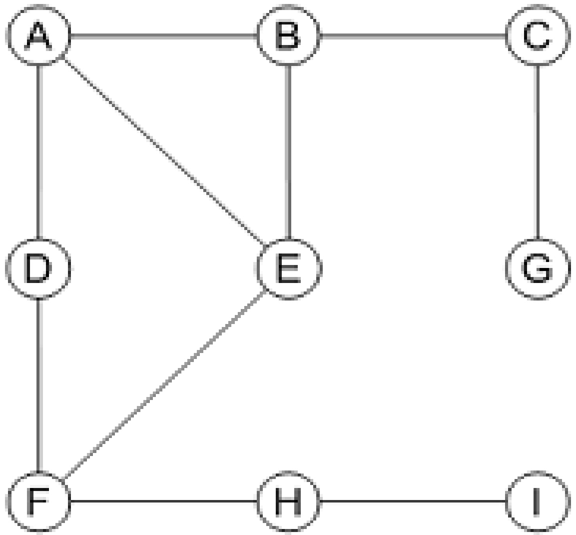
(a) 无向图G



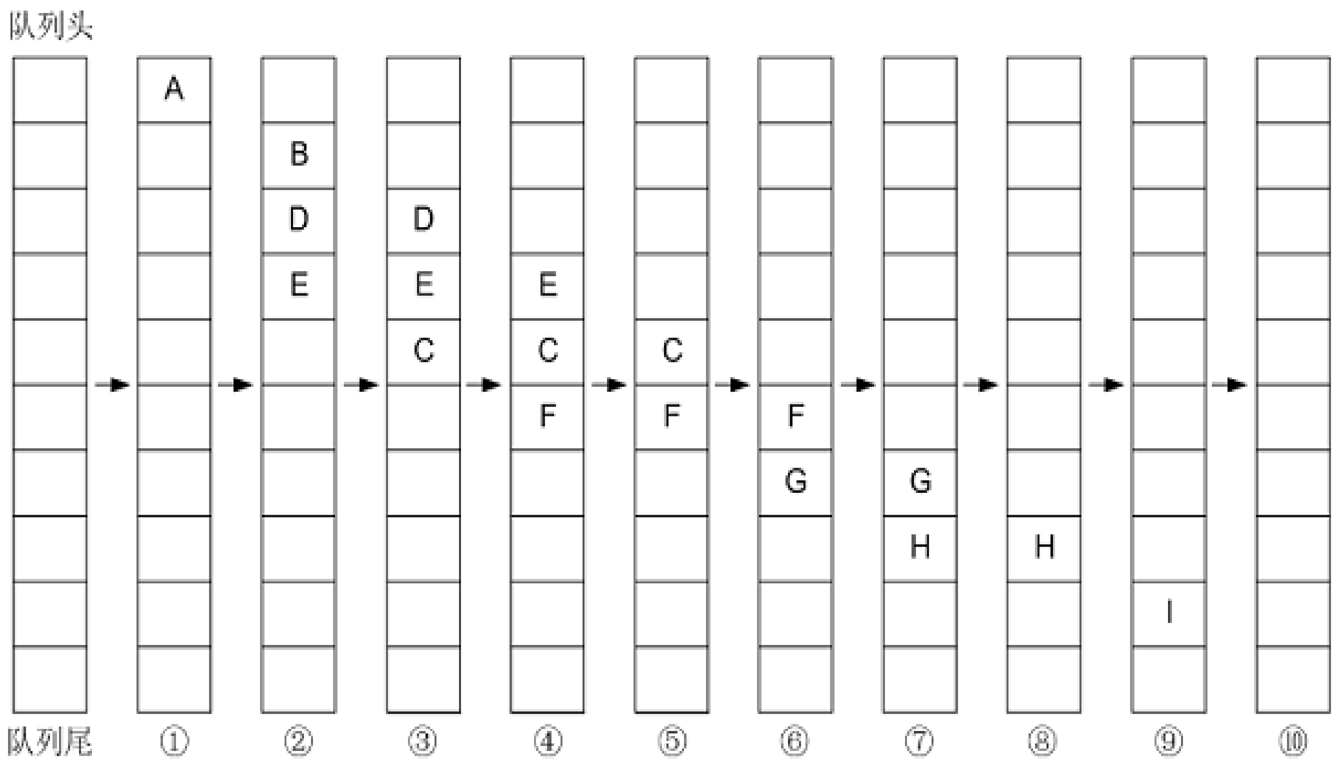
(a) 广度优先过程



(b) 广度优先搜索生成树



(a) 无向图*G*



BFS 算法：

```
BFS( 顶点 s ) //从顶点 i 进行广度优先搜索{
    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
```

图的遍历的基本应用：

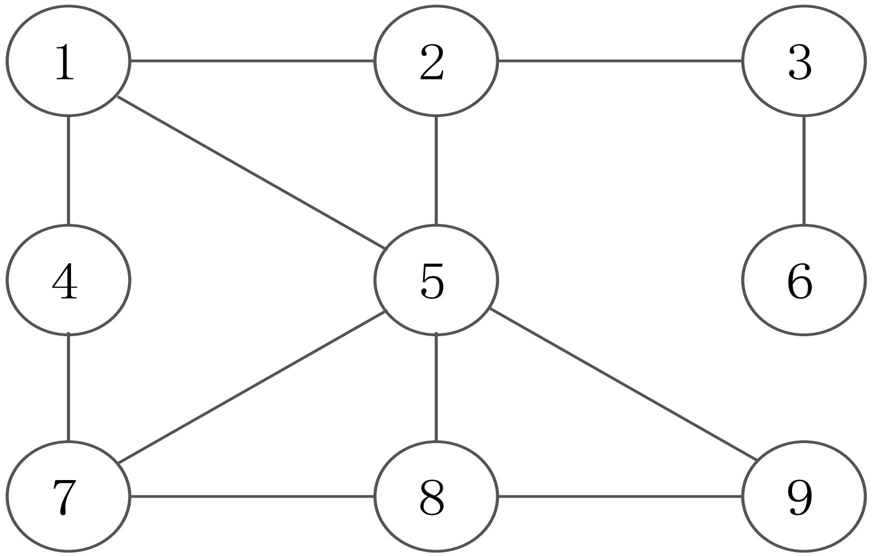
- 1.无向图的连通分量。(DFS||BFS)
- 2.图的最短路程d(s,t):s到t经过的最少边数 。(BFS)

【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。
输出:
第一行：dfs遍历序列
第二行：bfs遍历序列



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

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

方法1：用邻接矩阵存储边

实现:

DFS 算法框架:

```
DFS( 顶点 u ) //从顶点 i 进行深度优先搜索{
    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
        }
    }
}
```

```
void dfs(int u){
    vis[u]=1;
    cout<<u<<" ";
    for(int v=1;v<=n;v++)
        if(g[u][v]==1&&vis[v]==0)dfs(v);
}
```

BFS 算法：

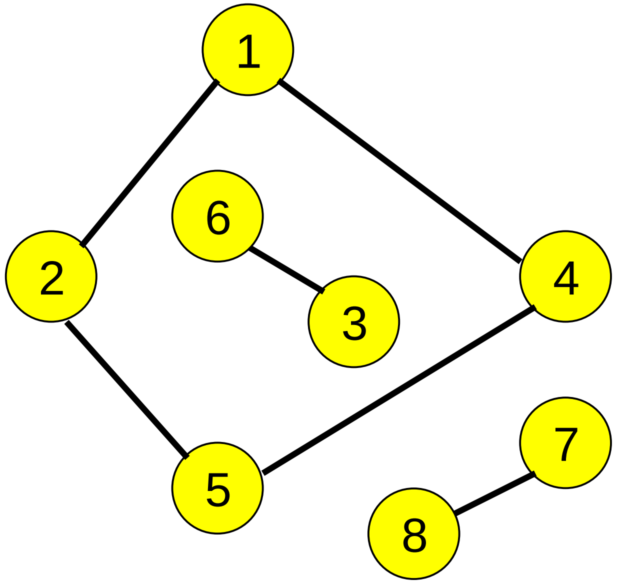
```
BFS( 顶点 s ) //从顶点 i 进行广度优先搜索{
    vis[ s ] = 1; //将顶点 s 的访问标志置为 1
    将顶点 s 入队列;
    while( 队列不为空 ){
        取出队列头的顶点，设为 u
        for( v=1; v<=n; v++ ) //对其他所有顶点 v{
            //v 是 u 的邻接顶点，且顶点 v 没有访问过
            if( a[u][v]==1 && !vis[v] ){
                将顶点 v 的访问标志置为 1
                将顶点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

```
void bfs(int s){
    q[0]=s;
    vis[s]=1;
    int head=0,tail=1;
    while(head<tail){
        int u=q[head++];
        cout<<u<<" ";
        for(int v=1;v<=n;v++)
            if(g[u][v]==1&&vis[v]==0){
                q[tail++]=v;
                vis[v]=1;
            }
    }
}
```

1.求无向的连通分量

dfs(1)?只能找到含1的连通分量

```
cnt=0;
for(int  v=1;v<=n;v++)
      if(!vis[v]){
            cnt++;
            dfs(v);//或者bfs(v)
      }
```



例2.犯罪团伙

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,罪犯数量），

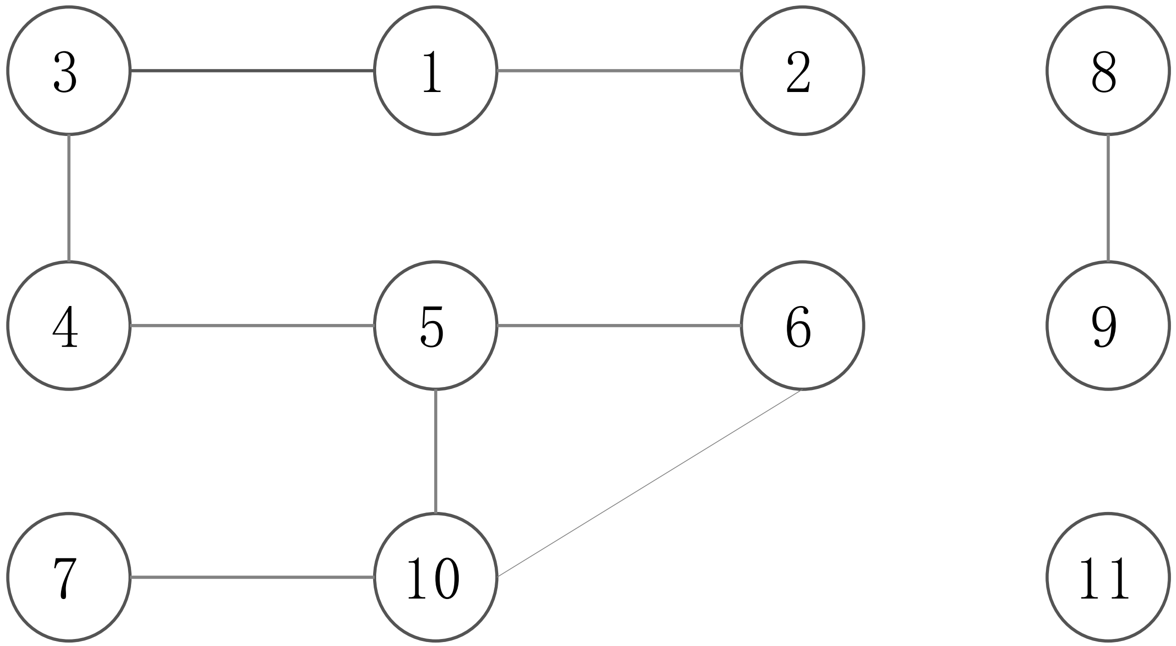
第二行：m（<100000，关系数量）

以下若干行：每行两个数：i 和j，中间一个空格隔开，表示罪犯i和罪犯j相互认识。

输出：

一个整数，犯罪团伙的数量。

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



用邻接表存储（Vector）

- 求无向图的连通分量

```
scanf("%d%d",&n,&m);
for(int i=1;i<=m;i++){
    int u,v;
    scanf("%d%d",&u,&v);
    g[u].push_back(v);
    g[v].push_back(u);
}
cnt=0;
for(int i=1;i<=n;i++)
    if(!vis[i])dfs(i,++cnt);
cout<<cnt<<endl;
```

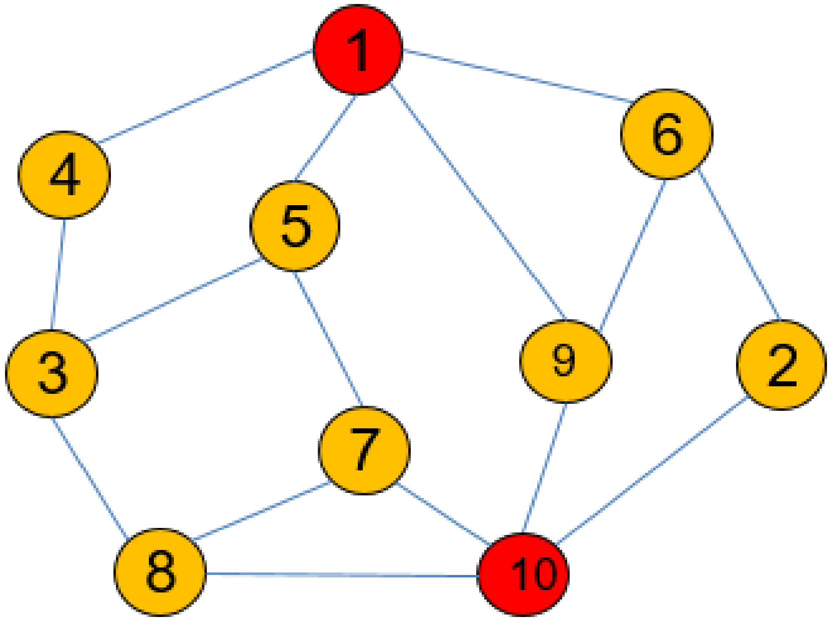
```
void dfs(int u,int id){
	vis[u]=id;
	int k=g[u].size();
	for(int i=0;i<k;i++){
		int v=g[u][i];
		if(vis[v]==0)dfs(v,id);
	}
}
```

2.图的最短路程d(s,t):s到t经过的最少条边数 。(BFS)

- S到t最少步数

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。



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

- 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。

```
• 初始状态s进入队列;
• 加入队标记;
• While 队列不空{
•     队首p出队列;
      if(p是目标) return p.dep;
•     for 每个p扩展出的状态state
•         if state is not in queue
•             进入队列;
•             加入队标记;
• }
```

【课后训练】

1.油田(zoj1709/poj1562/uva572)

2.上学路线

2.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】

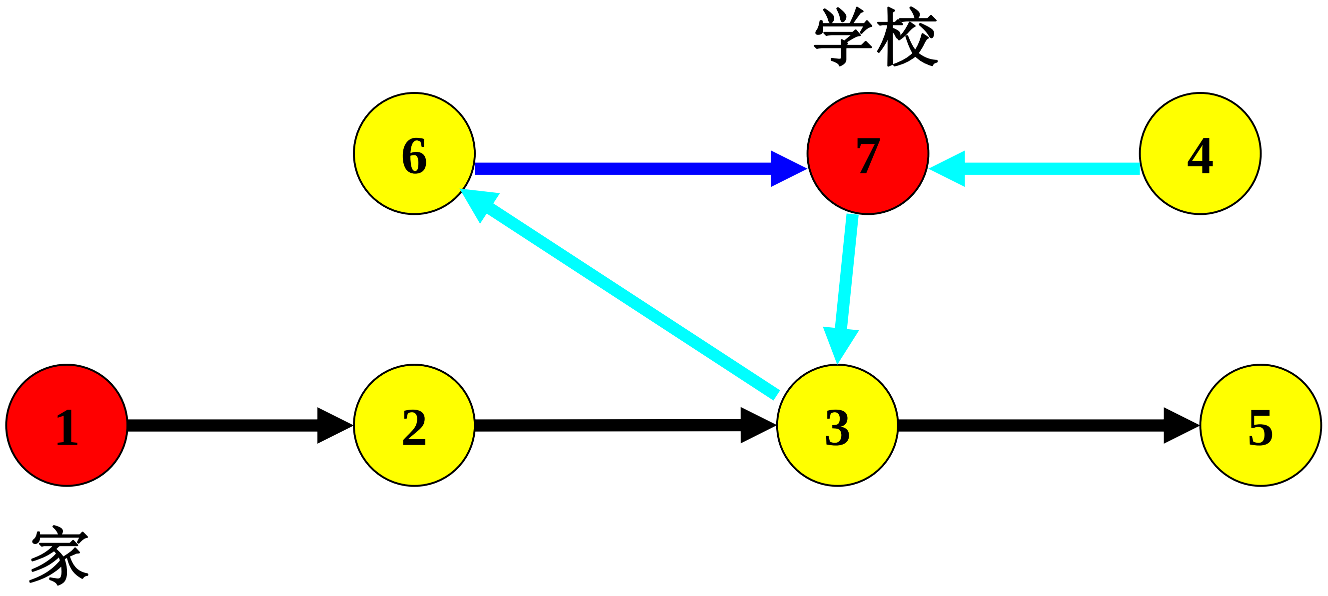
一行。如果无法乘从家到学校，则输出"No Roud!"，否则输出你的程序所找到的最少换车次数，换车次数为0表示不需换车即可到直达·

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

乘车路线：1　2　3　6　7：换了2次车。

【样例输入：】

3 7
6 7
4 7 3 6
1 2 3 5



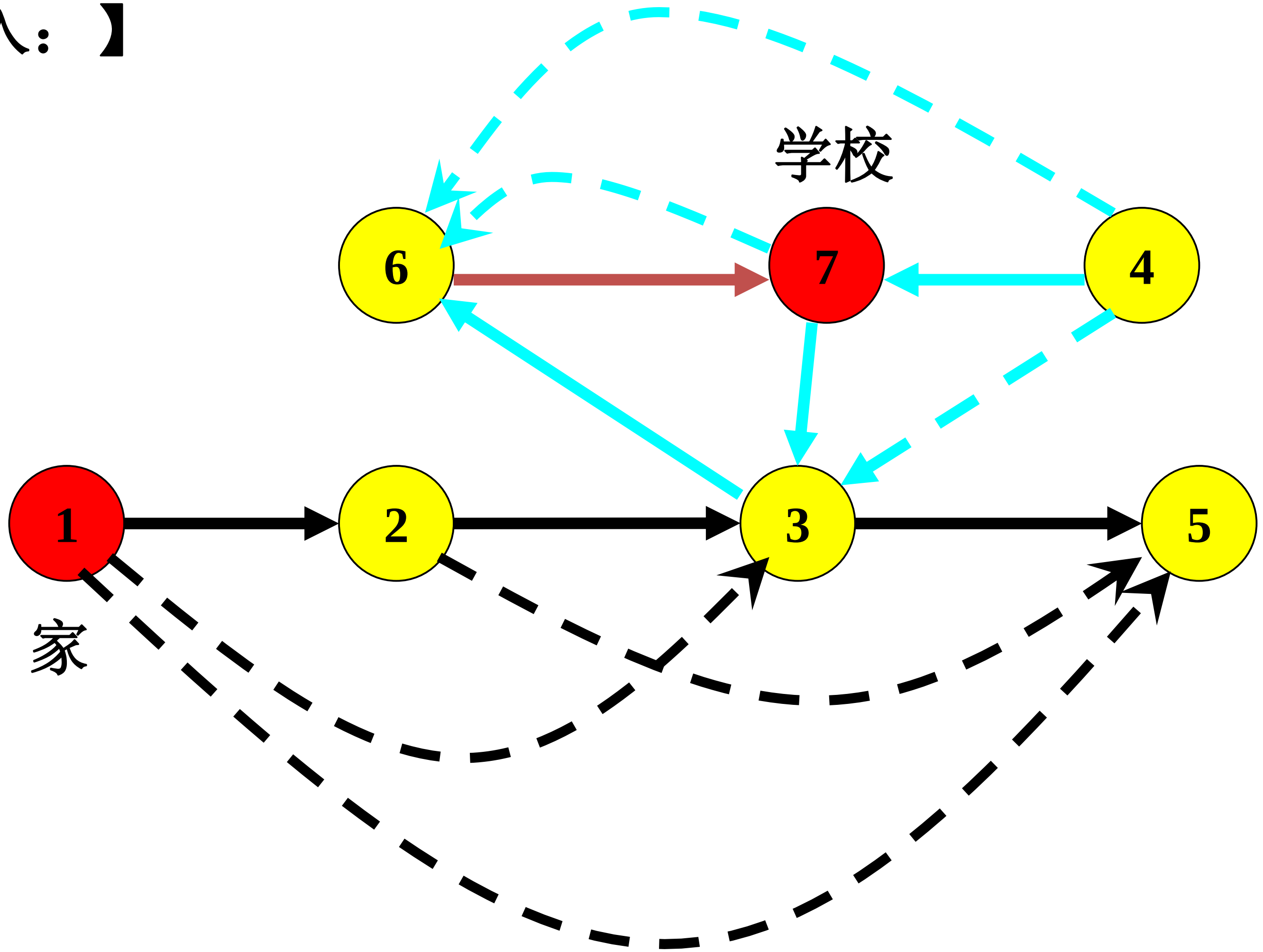
【样例输入：】

3 7

6 7

4 7 3 6

1 2 3 5



构造权为1的有向图

最少换车次数 :d[1,n]-1