图的基本概念及其存储结构

图的基本概念

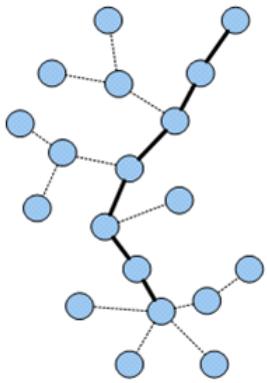
图 G=(V,E) 为节点集V和边集E组成的二元组，它描述了V中节点间的相互关系。

节点（vertex或node），若图的节点数为n，则节点一般用整数1，2，…，n标记。

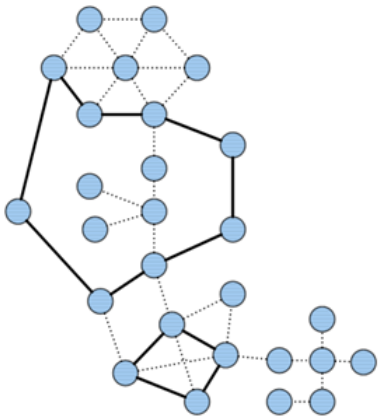
边（edge，arc或link），用无序数对（u，v）表示（1<=u,v<=n），或者表示成u-v。对于边e=（u,v），即可称u和v邻接，亦可称e和u,v关联。

节点的度数，与节点关联的边数。由于每个节点的度不可能超过n-1，因此边数$|E|$的上限为n*(n-1)/2。如果$|E|$达到上限，则称该图为完全图。没有自环和重边的图称为简单图，即该图中任意边的两端都不是同一个节点，任意一对节点间最多被一条边连接。

路径，图的一条路径指的是一个节点序列，该序列中的相邻节点在图上是邻接的。如果节点和边都不重复出现，则称为简单路径；若序列中除了起点和终点相同外没有重复节点和边，称为圈（亦称为环或回路）。



简单路径



环

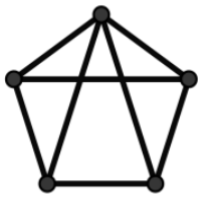
路径与路径之间有相交和不相交的关系。若路径间表示除了起点和终点外没有公共点，则称为不相交路。更严格的定义是，任意节点都不相同的路径叫严格不相交路径；任意边都不相同的路径叫严格边不相交路。

连通图，即图中任意两点之间都有路径，否则称为非连通图 。非连通图有多个连通分量，每个连通分量是一个极大连通子图，因为任意增加一个节点以后将成为非连通图。

[练习]

1.在一个无向图中，如果任意两点之间都存在路径相连， 则称其为连通图。 下图是一个有 5 个顶点、 8 条边的连通图。 若要使它不再是连通图， 至少要删去其中的（ ）条边。

A. 2　　B. 3　　C. 4　　D. 5



2.G是一个非连通无向图(没有重边和自环)，共有28条边，则该图至少有( )个顶点。

A. 10　　B. 9　　C. 11　　D. 8

3.在无向图中，所有顶点的度数之和是边数的( )倍。
A. 0.5　　B. 1　　C. 2　　D. 4

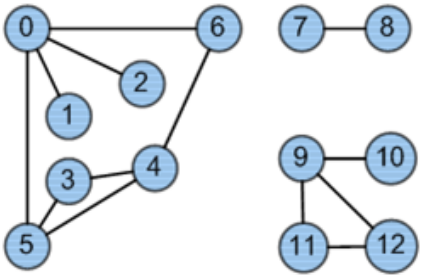
答案：1.B　2.B　3.C

图的简单分类

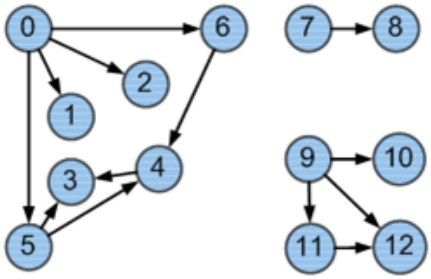
按照广义的分类，任何图都可分为无向图和有向图、无权图和加权图、稀疏图和稠密图。
按照狭义分类，图又可具体分为完全图和补图、树和森林、图的生成树和生成森林、平面图、二分图、相交图和区间图等特殊形态。

a.有向图和无向图

在图G=（V，E）中，如果对于任意的a，b∈V时，当（a，b）∈E时，必有（b，a）∈E（即关系R对称），则称此图为无向图。在一无向图中用不带箭头的边连接两个有关联的节点。



有向图中的边都是单向的，因此边（u，v）是有序数对。有时用带箭头的弧专指有向边。在有向边（u，v）中，u和v分别叫源和目的，或者叫尾和头。忽略有向图中所有边的方向，得到的无向图称为该有向图的基图。



b.无权图和加权图

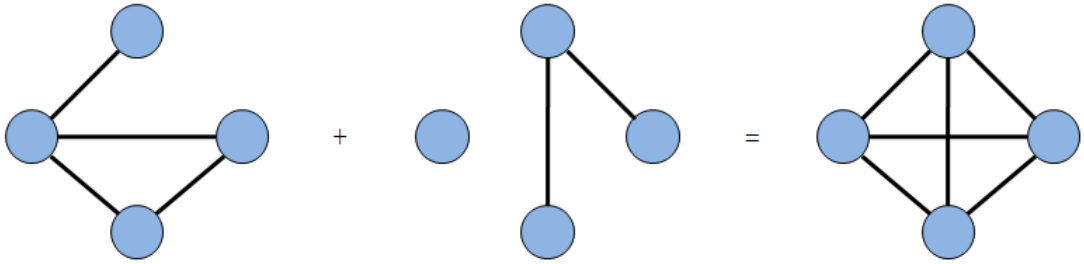
图中的节点和边不带权，则称该图为无权图。若给边或节点加权，则称该图为加权图或带权图。
权，可加于边或节点，通常代表费用、距离等，既可以是正数，也可以是负数。加权有向图一般也称为网络。

c.稀疏图和稠密图

边数$|E|$远小于$（n-1）*n/2$的图称为稀疏图，它的补图为稠密图。这两者之间是相对区分，并无明确界限。稀疏图一般采用时间复杂度为O(ElbE)的算法；稠密图一般采用时间复杂度为V^2的算法。

d.完全图和补图

若图的节点数为n，边数|E|达到n*（n-1）/2（即图中每个节点的度数为n-1），则称该图为完全图。
对于边（u，v），若邻接则改为非邻接，若非邻接则改为邻接，得到的图为原图的补图。
原图和补图的并为完全图，完全子图称为团。



原图　　补图　　完全图

[练习]

由四个不同的点构成的简单无向连通图的个数是( )。

A.32　　B. 35　　C.38　　D.41

答案：C
提示：用增边法或删变法，求组合数量和。

e.树和森林

不含圈的连通图称为树。树的集合称为森林。
树是图的一种特殊形态。一个图G是树当且仅当以下任意一个条件成立：
1、G有v-1条边，无圈
2、G有v-1条边，连通
3、任意两点只有唯一的简单路径
4、G连通，但任意删除一条边后不连通

[练习]

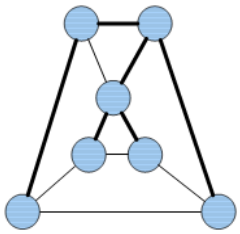
设G是有n个结点、m条边(n≤m)的连通图，必须删去G的( ) 条边，才能使得G变成一棵树。

A.m-n+ 1　　B. m-n　　C. m+n+ 1　　D. n-m+ 1

答案：A

f. 图的生成树和生成森林

包含某图G所有点的树称为图G的生成树。
包含某图G所有点的森林称为图G的生成森林。



[练习]

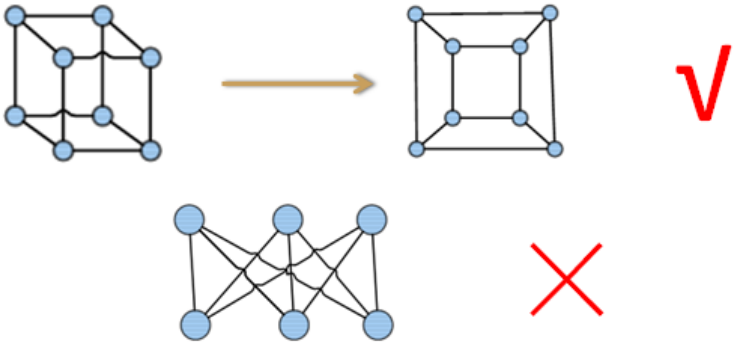
6 个顶点的连通图的最小生成树，其边数为( ) 。

A.6　　B.5　　C.7　　D.4

答案：B

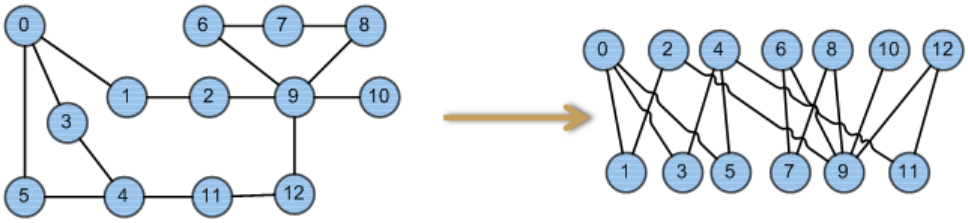
g.平面图

一个图如果能画在二维平面上，且各边没有交点，则该图称为平面图。注意，树是平面图，但反之却不一定。



h.二分图

二分图是一种特殊类型的图：图中的节点集被划分成X和Y两个子集，图中每条边的两个端点一定是一个属于X而另一个属于Y。这样的图只有奇圈，即包含奇数条边的圈。

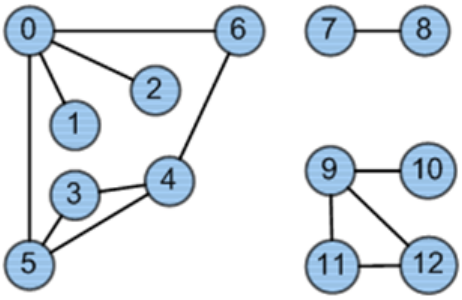


图的存储和遍历

a.利用相邻矩阵存储节点

相邻矩阵是表示接点间相邻关系的二维数组，若G=（V,E）是一个具有n个节点的图，则G得相邻矩阵是如下定义的二维数组A，其规模为n*n：

$$A[I,j]=\begin{cases}1(或权) & (vi,vj\in E)\\ 0(\pm\infty) & (vi,vj\in E)\end{cases}$$



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0	0	0	0	0	0	0
4	0	0	0	1	0	1	1	0	0	0	0	0	0
5	1	0	0	1	1	0	0	0	0	0	0	0	0
6	1	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	1	0	0	0
11	0	0	0	0	0	0	0	0	0	1	0	0	1
12	0	0	0	0	0	0	0	0	0	1	0	1	0

建图代码：

```cpp
#include<iostream>
using namespace std;
int g[1001][1001],n,m,u,v;
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u][v]=1;
        g[v][u]=1;//若为无向图，则一条边需要当两条边来存储
    }
}
```

遍历图，可采用深搜或广搜的方式。

深搜方式：

```cpp
void dfs(int u){
    cout<<u<<" ";//输出当前访问节点，或根据题意对节点进行相应处理
    vis[u]=true;//标记已访问
    for(int i=1;i<=n;i++){
        if(g[u][i]==1&&!vis[i]) dfs(i);//查找下一个邻接点
    }
}
```

广搜方式：

```cpp
void bfs(int s){
    queue<int> q;
    q.push(s);
    vis[s]=true;
    while(!q.empty()){
        int u=q.front();
        q.pop();
        cout<<u<<" ";//输出当前访问节点，或根据题意对节点进行相应处理
        for(int i=1;i<=n;i++){
            if(g[u][i]==1&&!vis[i]){//查找下一个邻接点
                q.push(i);
                vis[i]=true;//标记已访问
            }
        }
    }
}
```

可分别采用这两种方式，查看同一个图的节点访问顺序。

[练习]

无向图G中有n个节点，m条边。共有k次查询，查询节点u与v之间是否可达。

代码参考：

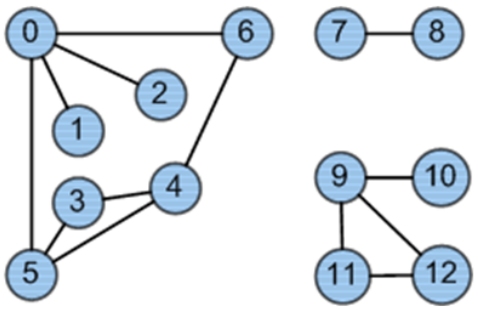
```cpp
#include<iostream>
#include<cstring>
using namespace std;
bool g[101][101],vis[101],flag;
int n,m,k,u,v;
```

```
void dfs(int u,int v){
    if(u==v){
        flag=true;
        exit;
    }
    vis[u]=true;
    for(int i=1;i<=n;i++){
        if(g[u][i]&&!vis[i]){
            dfs(i,v);
        }
    }
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u][v]=g[v][u]=true;
    }
    cin>>k;
    for(int i=1;i<=k;i++){
        cin>>u>>v;
        memset(vis,false,sizeof(vis));
        flag=false;
        dfs(u,v);//从u为起点进行搜索，看是否能遍历到v
        if(flag) cout<<"ok"<<endl;
        else cout<<"no way"<<endl;
    }
}
```

相邻矩阵存储方式在建图和遍历中使用方便，通过标记进行增边删边的方式也很快捷。但是这种方式消耗的空间非常大，且空间的利用率并不高。

b.利用链表存储相邻节点

这种方式常用vector来进行维护。在图的邻接表中，每个节点相连的所有边由一条链表保存。



	0	1	2	3	4	5	6	7	8	9	10	11	12
0	1	2	5	6									
1	0												
2	0												
3	4	5											
4	3	5	6										
5	0	3	4										
6	0	4											
7	8												
8	7												
9	10	11	12										
10	9												
11	9	12											
12	9	11											

建图代码：

```cpp
#include<iostream>
#include<vector>
using namespace std;
int n,m,u,v;
vector<int> g[1001];
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
}
```

如果是有权图，一条边除了要记录源头和目的外，还有权值等其他数据，我们也可以将每条边的参数封装为结构体来存储，如：

```cpp
#include<iostream>
#include<vector>
using namespace std;
int n,m,u,v,w;
struct edge{
    int v,w;//v为指向节点，w为权值
};
vector<edge> g[1001];
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v>>w;
        g[u].push_back((edge){v,w});
        g[v].push_back((edge){u,w});
    }
}
```

遍历图，同样可采用深搜或广搜的方式。

深搜方式：

```cpp
void dfs(int u){
    cout<<u<<" ";//输出当前访问节点
    vis[u]=true;//标记已访问
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];//找到邻接点
        if(!vis[v]) dfs(v);
    }
}
```

广搜方式：

```cpp
void bfs(int s){
    queue<int> q;
    q.push(s);
    vis[s]=true;
    while(!q.empty()){
        int u=q.front();
```

```
        q.pop();
        cout<<u<<" ";
        for(int i=0;i<g[u].size();i++){
            int v=g[u][i];//找到邻接点
            if(!vis[v]){
                q.push(v);
                vis[v]=true;
            }
        }
    }
}
```

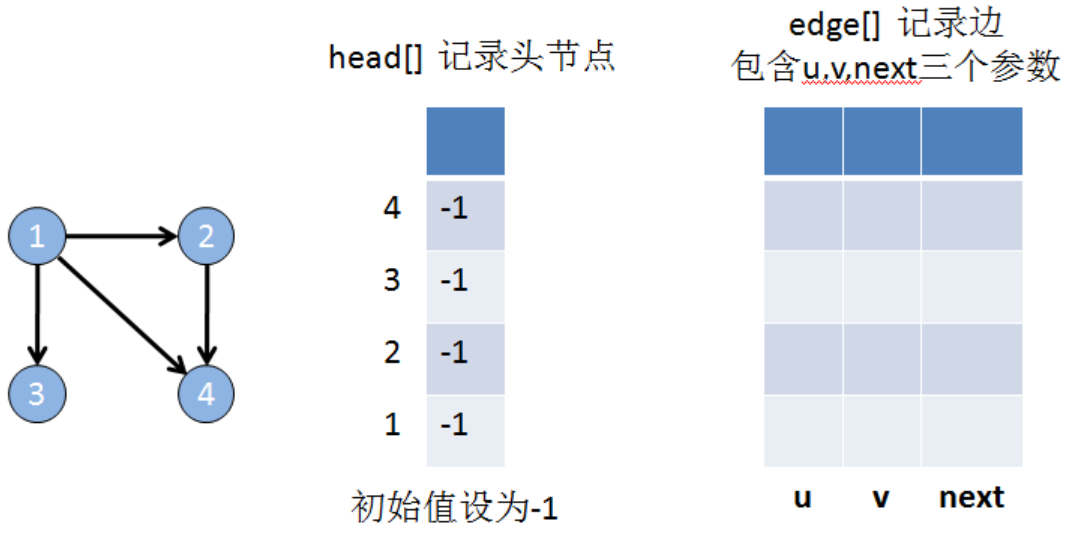
利用链表存储占空空间小，访问效率高。但是在遇到需要删边的情况，会涉及到大量的数据移动，效率低甚至易出错。

c.前向星（静态链表）存储

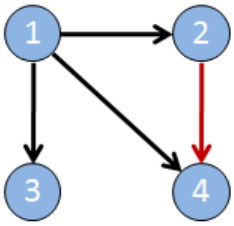
与第二种方法类似，以数组形式来模拟链表，以此来优化删边操作的效率，尤其是需要进行多次删边的情况。

在G（G=（V，E），|V|=n，|E|=e）的静态链表表的定义中，使用了两个一维数组head[1……n]和edge[1……tot]，其中edge[1……tot]记录每一条边的信息，包括源头、目的和next三个参数的信息，head[1……n]表示节点i在edge表中的首指针。当读入一条边的信息（u,v），在edge[++tot]中新申请一个单元的空间进行存储，此时edge[tot].next的值指向head[u]的值，意味将这条边的信息对应到源头节点u，从尾插入。head[u]的值修改为tot，意味着edge[tot]存储着以u为源头的当前最末边。以此类推，以u为源头的所有边将关联在一起。遇到需要删边的情况，则依据边的位置相应修改相邻节点的next、head[u]的值，即修改邻边的关联即可。

如图：



1.读入边（2,4）



head[] 记录头节点

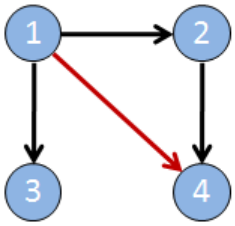
edge[] 记录边
包含u,v,next三个参数

	head[]	u	v	next
4	-1			
3	-1			
2	1			
1	-1	2	4	-1

tot=1

初始值设为-1

2.读入边（1,4）



head[] 记录头节点

edge[] 记录边
包含u,v,next三个参数

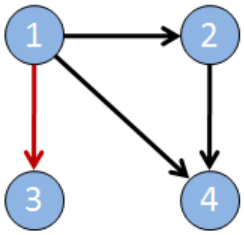
	head[]	u	v	next
4	-1			
3	-1			
2	1	1	4	-1
1	2	2	4	-1

tot=2

1

初始值设为-1

3.读入边（1,3）



head[] 记录头节点

edge[] 记录边
包含u,v,next三个参数

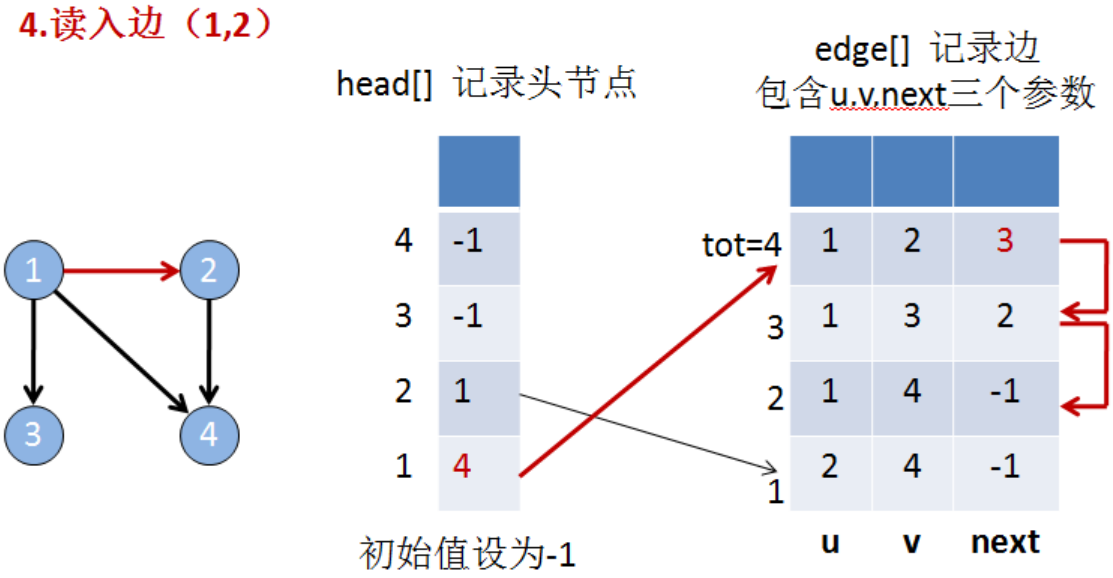
	head[]	u	v	next
4	-1			
3	-1	1	3	2
2	1	1	4	-1
1	3	2	4	-1

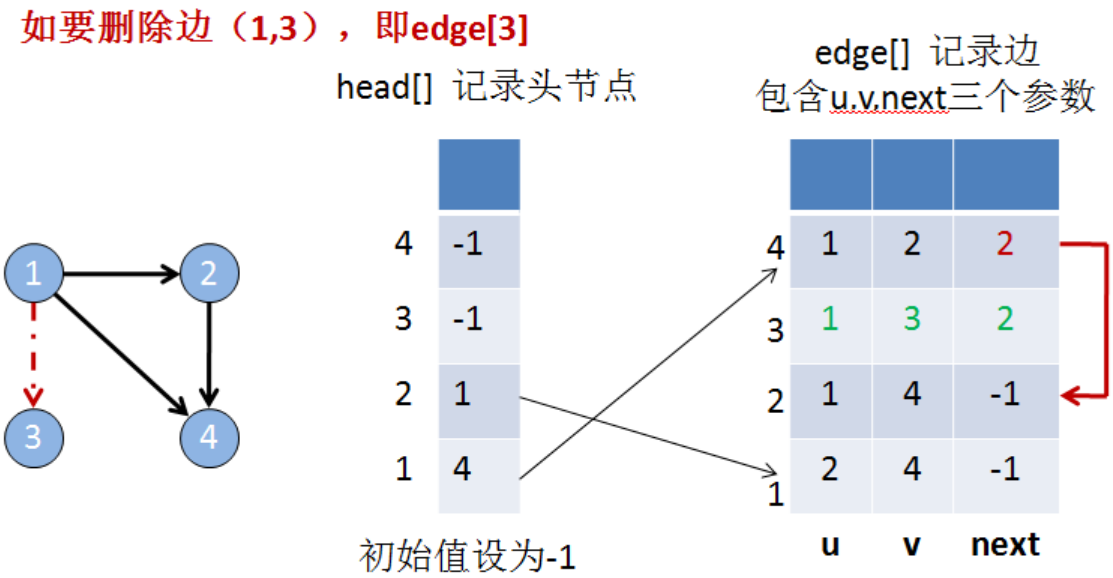
tot=3

2

1

初始值设为-1





以无向图为例，建图及深搜遍历代码入下：

```cpp
#include<iostream>
#include<cstring>
using namespace std;
int n,m,u,v,tot,head[1001];
bool vis[1001];
struct edge{
    int u,v,next;
}g[100001];
void addedge(int u,int v){
    g[++tot].u=u;//新申请空间
    g[tot].v=v;
    g[tot].next=head[u];
    head[u]=tot; //从尾插入
}
void dfs(int u){
    cout<<u<<" ";
    vis[u]=true;
    for(int i=head[u];i!=-1;i=g[i].next){//注意i的取值和边界
        int v=g[i].v;
        if(!vis[v]) dfs(v);
    }
}
```

```
int main(){
    cin>>n>>m;
    memset(head,-1,sizeof(head));
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        addedge(u,v);
        addedge(v,u);//如图为无向图，需将一条边当两条不同向的边存储
    }
    for(int i=1;i<=n;i++){
        if(!vis[i]) dfs(i);
    }
}
```

判环问题

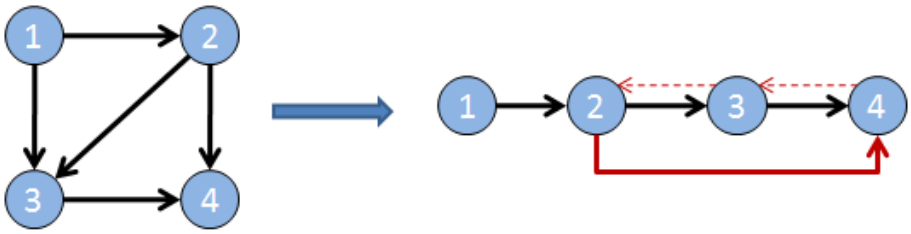
a.无向图判环

无向图的判环问题并不难理解，在遍历过程中我们会对已访问过的节点进行标记，如在搜索过程中能访问到已访问过的点，则说明图中存在环。但要注意的是，所谓已访问过的节点，不能是当前节点的父节点，因为在无向图中一条边是当作两条不同向的边来存储的。如忽略这点，则所有无向图都存在环，这是不正确的。

```
#include<iostream>
#include<vector>
using namespace std;
int n,m,u,v;
bool vis[1001],flag;
vector<int> g[1001];
void dfs(int u,int fa){
    vis[u]=true;
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        if(v==fa) continue;//v为u的父节点，忽略不计
        if(!vis[v]) dfs(v,u);
        else flag=true;//访问到已访问过的点，有环
    }
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for(int i=1;i<=n;i++){
        if(!vis[i]) dfs(i,-1);//-1表示i为根节点
    }
    if(flag) cout<<"have circle";
    else cout<<"have no circle";
}
```

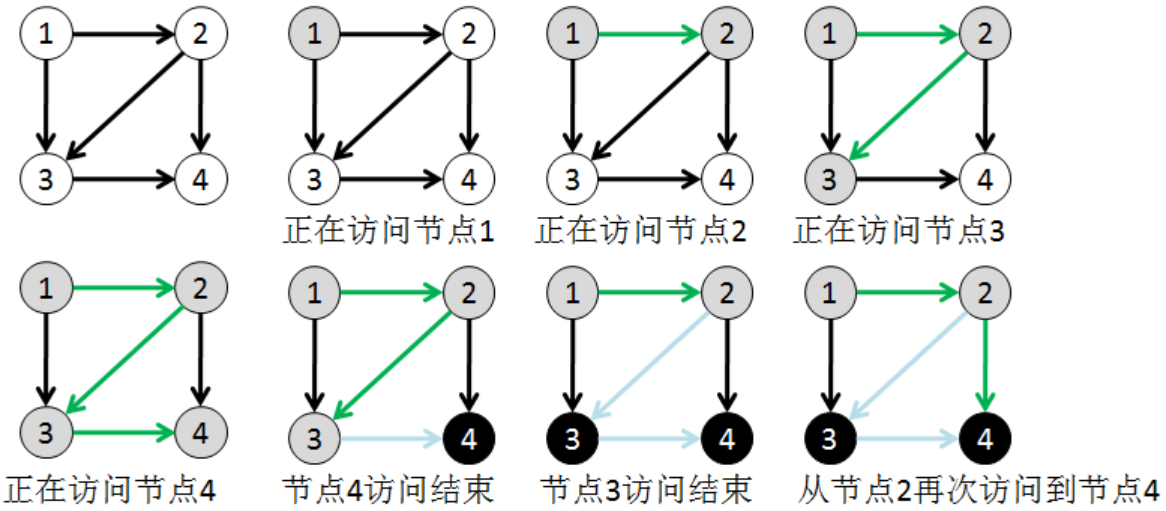
b.有向图判环

在有向图中，如果也仅停留于访问到已访问过的非父节点的思路，则会产生许多“无中生有”的环。如图：

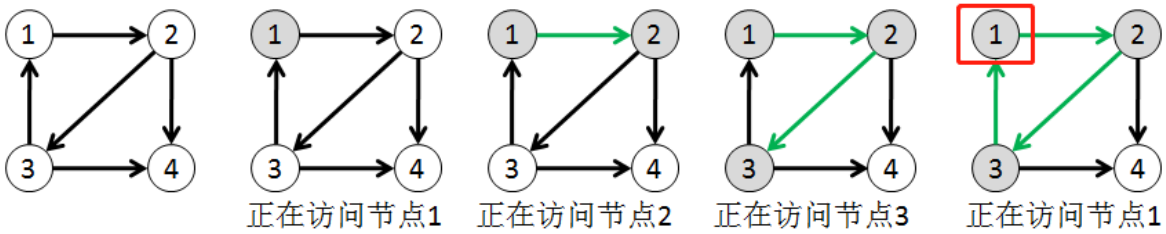


以深搜的方式来遍历该图，首先会先访问到节点1，接着节点2、节点3、节点4，将该路径的节点都标记为已访问。由于节点4没有其他路径可继续走，则返回到节点3。节点3也无其他路径可走，则返回到节点2。此时，节点2存在另一条路径，再次访问到节点4，而节点4已被标记为已访问，如果以此断定有环，明显与实际不相符。

我们可以在访问过程中，对每个节点进行染色标记访问情况，如



访问到已访问结束的点，并不存在环。只有访问到正在访问的节点，才说明有环的存在。如




```cpp
#include<iostream>
#include<vector>
using namespace std;
int n,m,u,v,vis[1001];
bool flag;
vector<int> g[1001];
void dfs(int u){
    vis[u]=1;//正在访问
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        if(vis[v]==0) dfs(v);//未访问，继续搜索
        if(vis[v]==1) flag=true;//访问到正在访问的节点，有环
    }
    vis[v]=-1;//访问结束
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
```

```
        g[u].push_back(v);
    }
    for(int i=1;i<=n;i++){
        if(!vis[i]) dfs(i);
    }
    if(flag) cout<<"have circle";
    else cout<<"have no circle";
}
```

A.图的m着色问题

【题目描述】

给定无向连通图G和m种不同的颜色。用这些颜色为图G的各顶点着色，每个顶点着一种颜色。如果有一种着色法使G中每条边的2个顶点着不同颜色，则称这个图是m可着色的。图的m着色问题是对于给定图G和m种颜色，找出所有不同的着色法。
对于给定的无向连通图G和m种不同的颜色，编程计算图的所有不同的着色法。

【输入】

输入数据。第1行有3个正整数n，k 和m，表示给定的图G有n个顶点和k条边，m种颜色。顶点编号为1，2，...，n。接下来的k行中，每行有2个正整数u,v，表示图G 的一条边(u,v)。

【输出】

程序运行结束时，将计算出的不同的着色方案数输出

样例输入	样例输出
5 8 4 1 2 1 3 1 4 2 3 2 4 2 5 3 4 4 5	48

【提示】

$n <= 100$

【分析】

建图，构建各个顶点间的关联。深搜遍历分配各个顶点的颜色，累计可行方案数。

【代码】

```
#include<iostream>
#include<vector>
using namespace std;
int n,k,m,u,v,ans,c[101];
bool vis[101];
vector<int> g[101];
bool check(int u,int cl){//判断u节点可否使用cl颜色
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
```

```
        if(c[v]==cl) return false;
    }
    return true;
}
void dfs(int u){
    if(u>n){
        ans++;
        return;
    }
    for(int i=1;i<=m;i++){
        if(check(u,i)){
            c[u]=i;//为u节点分配i颜色
            dfs(u+1);
            c[u]=0;//解锁，重新考虑下一种分配方案
        }
    }
}
int main(){
    cin>>n>>k>>m;
    for(int i=1;i<=k;i++){
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);//无向图
    }
    dfs(1);
    cout<<ans;
}
```

B.树

【问题描述】

小明终于忙玩了各种各样的课程，终于可以继续学习算法了。
他在图论书上看到了树，树有许许多多特殊的性质。小明一下子就喜欢上了这种特殊的树。
于是，他发明了自己的对于无向图的评分方法。
一个无向图的分数定义为，各个连通块是树的数量。
现在给定一个n个点m条边的无向图，问在小明的评分方法下，分数为多少。
一个连通块是树，当且仅当边数比点数少1。

【输入】

第一行两个整数n和m，表示图的点数和边数。
第二行有m对整数，u和v表示，结点u和节点v之间有边。给出的无向图不存在重边。

【输出】

输出一行包括一个整数，表示无向图的评分，也就是树的数量。

样例输入	样例输出
8 6 1 2 1 3 2 4 5 6 6 7 5 7	2

【提示】

数据范围
20%的数据，$1 <= n <= 2000$
100%的数据，$1 <= n <= 100000$,$0 <= m <= min(n*(n-1)/2, 200000)$

【分析】

无向图判环模板题

【代码】

```
#include<iostream>
#include<vector>
using namespace std;
int n,k,m,u,v,ans;
bool flag,vis[100001];
vector<int> g[100001];
void dfs(int u,int fa){
    vis[u]=true;
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        if(v==fa) continue;//同边
        if(!vis[v]) dfs(v,u);
        else flag=false;//有环
    }
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        g[u].push_back(v);
        g[v].push_back(u);
    }
    for(int i=1;i<=n;i++){
        if(!vis[i]){//未访问过，有可能为树
            flag=true;
            dfs(i,-1);//根节点
            ans+=flag;
        }
    }
    cout<<ans;
}
```

C. Cow Picnic

【题目描述】

奶牛们正在进行一次野餐！农民John的K头牛，其中第i头牛在N个农场中的某一个内吃草。农场用1到N连续编号。农场通过M条单向的道路连接(没有一条道路连接同一个农场）。
奶牛们要聚集到同一个农场来举行野餐，但是(由于道路是单向的)某些牛可能只能到达某一部分农场。
请帮助奶牛们确定有多少农场，使大家都能够到达的，这样伙伴们就可以在那些地方选择野餐的地点。

【输入】

第1行:三个用空格分开的整数，分别是：K,N,和M。
第2..K+1行:第i+1行包含一个整数(在1..N内)，表示第i头牛所在的农场编号。
第K+2..M+K+1行:每行两个用空格分开的整数，分别是A和B(都在1..N内，而且A≠B)，表示有一条单向的道路，从农场A直接连接到农场B。

【输出】

输出共一行一个整数，表示通过这些单向道路，所有牛都能够到达的农场的数量。

样例输入	样例输出
2 4 4 2 3 1 2 1 4 2 3 3 4	2

【提示】

[输入说明]
4<--3
^ ^
| |
| |
1-->2
农场地图如上图所示，两头牛分别在农场2和3号。

[输出说明]奶牛们能够集中到农场3或4。

【数据规模】

对于40%的数据：$1 \leq K \leq 10$；$1 \leq N \leq 10$；$1 \leq M \leq 100$；
对于60%的数据：$1 \leq K \leq 50$；$1 \leq N \leq 100$；$1 \leq M \leq 500$；
对于70%的数据：$1 \leq K \leq 50$；$1 \leq N \leq 100$；$1 \leq M \leq 1,000$；
对于100%的数据：$1 \leq K \leq 100$；$1 \leq N \leq 1,000$；$1 \leq M \leq 10,000$；$A \neq B$；

【分析】

该问题是有向图的建图及遍历问题。枚举每一头牛，第i头牛从所处位置a[i]为起点，搜索其所能到达的所有农场，每次到达一个农场，该农场的计数加1。最后对所有农场进行遍历，计数达到k的说明所有牛都能到达。

【代码】

```
#include<iostream>
#include<vector>
#include<cstring>
using namespace std;
int k,n,m,c[1001],q[101],s,t,ans;
vector<int> g[1001];
bool vis[1001];
void dfs(int v){
    c[v]++;
    vis[v]=true;
    for(int i=0;i<g[v].size();i++){
        if(!vis[g[v][i]])dfs(g[v][i]);
```

```
    }
}
int main(){
    cin>>k>>n>>m;
    for(int i=1;i<=k;i++){
        cin>>q[i];
    }
    for(int i=1;i<=m;i++){
        cin>>s>>t;
        g[s].push_back(t);
    }
    for(int i=1;i<=k;i++){
        memset(vis,false,sizeof(vis));
        dfs(q[i]);
    }

    for(int i=1;i<=n;i++){
        if(c[i]==k)ans++;
    }
    cout<<ans;
}
```

D.狼和羊

【题目描述】

米基家的后院养着一群羊，米基由于疲劳睡着了，这时一群饿狼钻进了后院开始攻击羊群，后院是由许多个方格构成的长方形区域，每个方格中用字符'.'表示空地，'#'表示栅栏，'o'表示羊，'v'表示狼，羊和狼所在的格子都是空地。
如果从一个空地A沿着水平方向或垂直方向经过一系列的空地能够到达空地B，则称空地A和空地B属于同一个羊圈。对于能够逃离后院的空地我们认为它不属于任何一个羊圈。
当一个羊圈中羊的数量大于狼的数量时，它们会用它们的尖角顶死该羊圈中的狼，否则就将被狼吃掉，最后每个羊圈中只会剩下一种动物。
写一个程序统计战斗结束后所有羊圈中羊的总数和狼的总数。

【输入】

输入文件的第一行包含两个用空格隔开的自然数R和C，其中$3 \leq R,C \leq 250$，R表示米基家后院的行数，C表示列数，接下来的R行每行包含C个字符，每个字符表示一个格子的情况。

【输出】

输出文件仅一包含两个用一个空格隔开的整数，表示要求的羊的数量和狼的数量。

样例输入	样例输出
9 12 . ###. #####. . #. oo#. . .#v#. #. . o#. # .#. #. #. . ##o# . . . #. #. #v#o###. #. #. . #v# . . . . #. #. . . v#v####. . #### .#vv .o# . . . . . ####.	3 5

【分析】

矩阵存图，也可理解为将每一个点与上下左右的邻接点（非'#'标记）形成路径进行构图。利用深搜或广搜的思路求封闭于区域内的连通块，对连通块内的数据进行处理。

【代码】

```
#include<iostream>
using namespace std;
int a[252][252],so,sv,to,tv,r,c;
int xx[4]={-1,0,1,0};
int yy[4]={0,1,0,-1};
bool flag;
void dfs(int x,int y){
    if(x<1||y<1||x>r||y>c){
        flag=false;
        return;
    }
    if(a[x][y]==0)return;
    if(a[x][y]==2)so++;
    if(a[x][y]==3)sv++;
    a[x][y]=0;//标记已访问
    for(int i=0;i<4;i++){
        int nx=x+xx[i];
        int ny=y+yy[i];
        dfs(nx,ny);
    }
}
int main(){
    cin>>r>>c;
    for(int i=1;i<=r;i++){
        for(int j=1;j<=c;j++){
            char ch;
            cin>>ch;
            if(ch=='.')a[i][j]=1;
            if(ch=='o')a[i][j]=2;
            if(ch=='v')a[i][j]=3;
        }
    }
    for(int i=1;i<=r;i++){
        for(int j=1;j<=c;j++){
            if(a[i][j]>0){
                so=0;sv=0;
                flag=true;
                dfs(i,j);
                if(flag){
                    if(so>sv){
                        to+=so;
                    }else{
                        tv+=sv;
                    }
                }
            }
        }
    }
    cout<<to<<" "<<tv;
}
```

E.传话

【题目描述】

一个朋友网络，如果a认识b，那么如果a第一次收到某个消息，那么会把这个消息传给b，以及所有a认识的人。 如果a认识b，b不一定认识a。 所有人从1到n编号，给出所有“认识”关系，问如果i发布一条新消息，那么会不会经过若干次传话后，这个消息传回给了i，$1 <= i <= n$。

【输入】

第一行是n($n < 1000$)和m($m < 10000$)，表示人数和认识关系数。
接下来的m行，每行两个数a和b，表示a认识b。$1 <= a,b <= n$。认识关系可能会重复给出，但一行的两个数不会相同。

【输出】

一共n行，每行一个字符T或F。第i行如果是T，表示i发出一条新消息会传回给i；如果是F，表示i发出一条新消息不会传回给i。

样例输入	样例输出
4 6 1 2 2 3 4 1 3 1 1 3 2 3	T T T F

【提示】

答案小于Maxlongint

【分析】

有向图判环问题，求以i为起点和终点的环是否存在。

【代码】

```
#include<iostream>
#include<vector>
#include<cstring>
using namespace std;
vector<int> g[1001];
int n,m,a,b,vis[1001];
bool dfs(int u,int t){
    vis[u]=1;
    for(int i=0;i<g[u].size();i++){
        int v=g[u][i];
        if(vis[v]==0){
            if(dfs(v,t))return true;
        }else{
            if(vis[v]==1){
                if(v==t)return true;
            }
        }
    }
    vis[u]=-1;
```

```
    return false;
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>a>>b;
        g[a].push_back(b);
    }
    for(int i=1;i<=n;i++){
        memset(vis,0,sizeof(vis));
        if(dfs(i,i)){
            cout<<"T"<<endl;
        }else{
            cout<<"F"<<endl;
        }
    }
}
```

F.寻找道路

【题目描述】

在有向图 G 中，每条边的长度均为 1，现给定起点和终点，请你在图中找一条从起点到 终点的路径，该路径满足以下条件：
路径上的所有点的出边所指向的点都直接或间接与终点连通。
在满足条件 1 的情况下使路径最短。
注意：图 G 中可能存在重边和自环，题目保证终点没有出边。 请你输出符合条件的路径的长度。

【输入】

第一行有两个用一个空格隔开的整数 n 和 m，表示图有 n 个点和 m 条边。
接下来的 m 行每行 2 个整数 x、y，之间用一个空格隔开，表示有一条边从点 x 指向点y。
最后一行有两个用一个空格隔开的整数 s、t，表示起点为 s，终点为 t。

【输出】

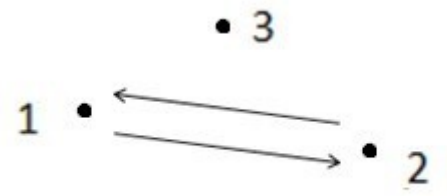
输出只有一行，包含一个整数，表示满足题目描述的最短路径的长度。
如果这样的路径不存在，输出-1。

样例输入	样例输出
3 2 1 2 2 1 1 3	-1
6 6 1 2 1 3 2 6 2 5 4 5 3 4 1 5	3

【提示】

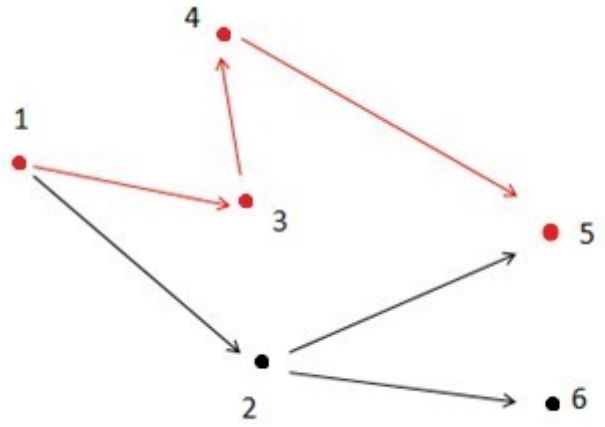
对于30%的数据，0 < n ≤ 10，0 < m ≤ 20；
对于60%的数据，0 < n ≤ 100，0 < m ≤ 2000；
对于100%的数据，0 < n ≤ 10,000，0 < m ≤ 200,000，0 < x，y，s，t ≤ n，x ≠ t。

[输入输出样例1说明]



如上图所示，箭头表示有向道路，圆点表示城市。起点1与终点3不连通，所以满足题目描述的路径不存在，故输出-1。

[输入输出样例2说明]



如上图所示，满足条件的路径为1->3->4->5。注意点2不能在答案路径中，因为点2连了一条边到点6，而点6不与终点5连通。

【分析】

本题是在一个有向图中求最短路，但并没有这么简单。关键点在于要求此路径上所有点的所有出边指向的点都必须与终点相通。因此，需要先确定哪些点可到达目的地，则遍历的时候只能遵循这些点去找路径。同时还得判断每次遍历到的点是否存在某一出边指向的点不可达t，如有则该路径不可行。

【代码】

```
#include<iostream>
#include<queue>
#include<vector>
using namespace std;
int n,m,u,v,s,t;
vector<int> gs[100001],gt[100001];
queue<int> qt;
struct point{
    int x,s;
};
queue<point> qs;
bool vist[100001],viss[100001];
void bfs_t(){
    qt.push(t);
    vist[t]=true;
```

```
    while(!qt.empty()){
        int f=qt.front();
        qt.pop();
        for(int i=0;i<gt[f].size();i++){
            int u=gt[f][i];
            if(!vist[u]){
                vist[u]=true;//标记能到达t的点
                qt.push(u);
            }
        }
    }
}
bool check(int x){
    for(int i=0;i<gs[x].size();i++){
        int u=gs[x][i];
        if(!vist[u]) return false;//判断是否存在某一出边指向的点不可达t
    }
    return true;
}
int bfs_s(){
    qs.push((point){s,0});
    viss[s]=true;
    while(!qs.empty()){
        point f=qs.front();
        qs.pop();
        for(int i=0;i<gs[f.x].size();i++){
            int v=gs[f.x][i];
            if(v==t) return f.s+1;//可达目的地t
            if(!viss[v]&&vist[v]&&check(v)){
                viss[v]=true;
                qs.push((point){v,f.s+1});
            }
        }
    }
    return -1;
}
int main(){
    cin>>n>>m;
    for(int i=1;i<=m;i++){
        cin>>u>>v;
        gs[u].push_back(v);
        gt[v].push_back(u);
    }
    cin>>s>>t;
    bfs_t();
    cout<<bfs_s();
}
```