图论算法在信息学竞赛中的应用

海亮高级中学 黄志刚

大纲

- 一、图论的相关定义
- 二、图的存储结构
- 三、图的遍历
  - Euler回路/Euler路问题
  - 例题：灵魂画师/词链/string
- 四、Tarjan算法在无向图中的应用
  - 割点/点双通分量/点双连通分量缩点
  - 割边/边双连通分量/边双连通分量缩点
- 五、Tarjan算法在有向图中的应用
  - 例题：CF427C Checkposts
- 六、Tarjan算法求LCA
- 七、最小生成树

一、图论的定义

- 图(Graph)可用G=(V,E)二元组来表示，即顶点集合V和边集合E组成的二元组。E中的每条边是V中一对顶点(u,v)。

  边 Edge 定点Vertex

- 如果(u,v)是无序对，那么称该图为无向图，否则为有向图。

- 任意两个顶点最多只有一条边(多条边称为重边)，且每个点都没有连接到它自身的边(无自环)的图叫简单图。

顶点和边

- 如果(u,v)是E(G)中的一条边，则称u与v互为邻接顶点。
- 在无向图中，一个顶点v的度是与他相关联的边的条数。
- 在有向图中，顶点的度等于该顶点的入度与出度之和。顶点v的入度是以v为终点的有向边的条数；顶点v的出度是以v为始点的有向边的条数。
- 某些图的边具有与它相关的数，称之为权。这种图称为网络或带权图。

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/Degree

在有向图中：

入度——以该顶点为终点的边的数目和 . ID(3)=2 /InDegree

出度——以该顶点为起点的边的数目和 . OD(3)=1 /OutDegree

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。

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 $e = \dfrac{n*(n-1)}{2}$

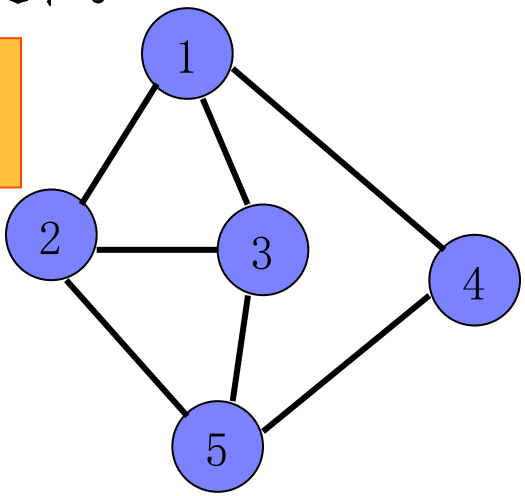


图1:无向图

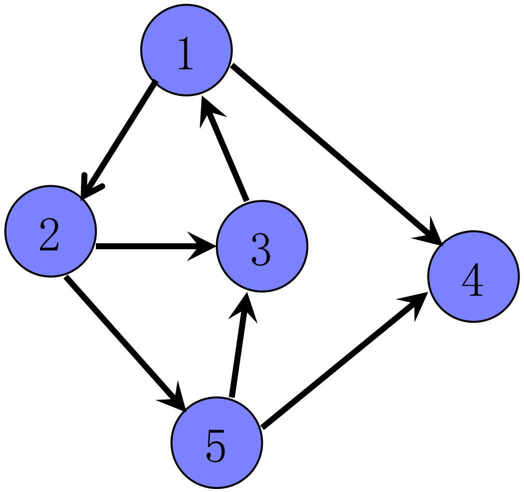


图2：有向图

路径

- 在图G=(V,E)中，若从顶点v_i出发，沿一些边经过一些顶点v_{p1},v_{p2},…,v_{pm}，到达顶点v_j。则称顶点序列(v_i v_{p1} v_{p2} … v_{pm} v_j)为从顶点v_i到顶点v_j的路径。它经过的边(v_i,v_{p1})、(v_{p1},v_{p2})、…、(v_{pm},v_j)应是属于E的边。非带权图的路径长度是指路径上边的条数。带权图的路径长度是指路径上各边的权之和。

- 若路径上各顶点v_1,v_2,…v_m均不互相重复，则称这样的路径为简单路径。若路径上第1个顶点v_1与最后一个顶点v_m重合，则称这样的路径为回路或环。

图的连通性

- 在无向图中，若从顶点v_1到顶点v_2有路径，则称顶点v_1与v_2是连通的。
- 如果图中任意一对顶点都是连通的，则称此图是连通图。非连通图的极大连通子图叫做连通分量。
  Connect Component
- 特别的，一个连通图的生成树是它的极小连通子图，在n个顶点的情况下，有n-1条边。

- 在有向图中，若对于每一对顶点v_i和v_j，都存在一条从v_i到v_j和v_j到v_i的路径，则称此图是强连通图。非强连通图的极大强连通子图叫做强连通分量。
  Strong Connect Component

点连通性

- 设无向图G是连通的，若有结点集 $V_1 \subseteq V$ ，使得图G删除了v_1所有的结点后所得的子图是不连通的，而删除了v_1的任意真子集后，所得的子图仍然是连通图，则称集合v_1为图G的点割集。若某一结点就构成了一个点割集，则称该结点为割点。包含点数最少的割集所包含的点数称为G的点连通度k(G)

边连通性

- 设无向图G为连通的，若有边集 $E_1 \subseteq E$ ，使得图G删除了E_1所有边后所得的子图是不连通的，而删除了E_1的任意真子集后，所得的子图仍然是连通图，则称集合E_1为图G的边割集。若某一边构成边割集，则称该边为桥(或割边)。包含边数最少的边割集所包含的边数称为G的边连通度k’(G)

二、图的存储结构

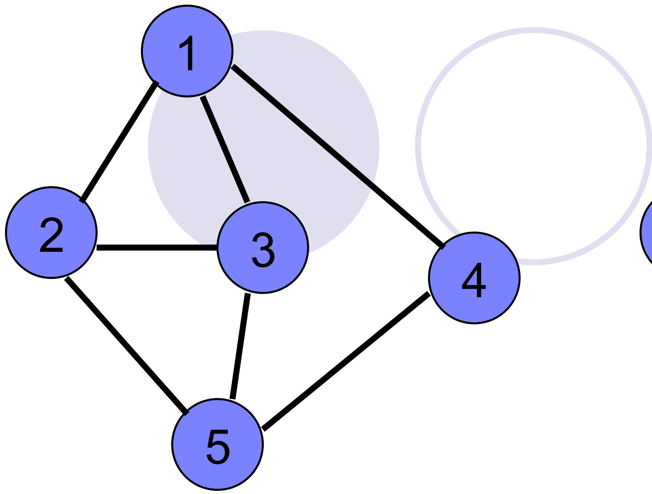
- 邻接矩阵：
  - 设图G=(V,E)有n个顶点，g为一个二维数组，则当(i,j)为图中的边时有g[i][j]=1，否则g[i][j]=0。如果需要存储带权图，则把g[i][j]的值改为对应权w(i,j)的大小。
  - 带权的邻接矩阵无法保存重边。
  - 邻接矩阵的空间复杂度为O(N^2)，查询边的复杂度O(1)，添加边的复杂度O(1)
- 邻接表
  - 将同一个顶点出发的边链接在同一个边链表中，链表的每一个结点代表一条边，叫做边结点。
  - 空间复杂度O(M+N)，查询边的复杂度O(M_i)，M_i为与i相连的边的数目，添加边的复杂为O(1)
- 边集数组

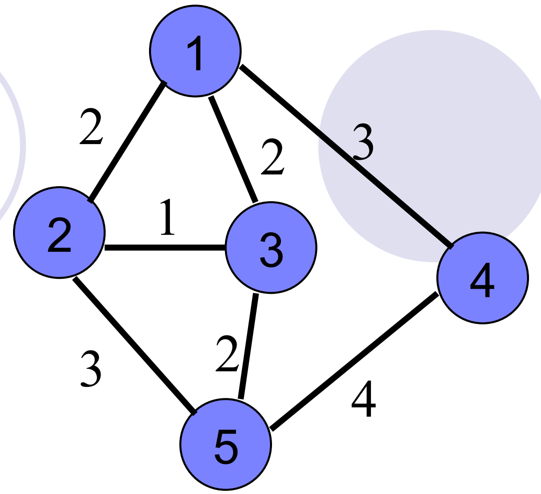
1、图的邻接矩阵

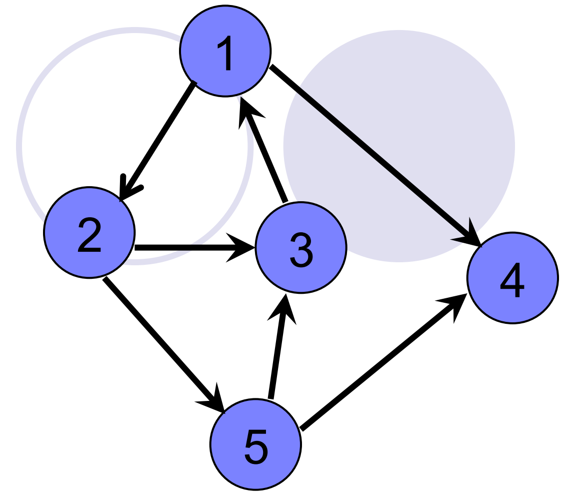
邻接矩阵是表示结点间相邻关系的矩阵。若G=（V，E）是一个具有n个结点的图，则G的邻接矩阵是如下定义的二维数组 a[1..n][1..n]。

注意：n尽量稍微大点。

$$a[i][j]=\begin{cases} 1\ \text{(或权值)：无向图：有边}(i,j)\text{和边}(j,i) \\ \qquad\qquad\qquad\text{有向图：有边}\langle i,j\rangle \\ 0\text{：}\ i\ \text{到}\ j\ \text{无边} \end{cases}$$







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

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

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

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。

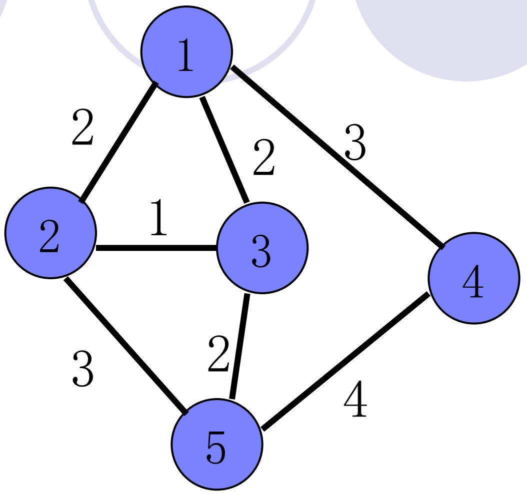
第i行非0 的个数是结点i的度

具体到题目时，数据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
```

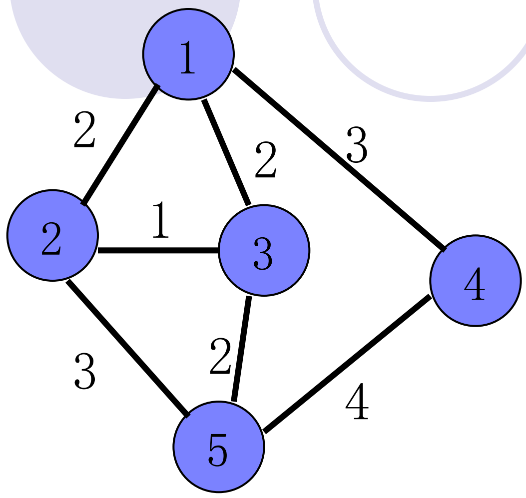



```
Int const maxn=100
Int a[maxn][maxn]
memset(a,0, sizeof(a));
cin>>n;
for (int i=0;i<n;i++)
    for(int j=0;j<n;j++)
      cin>>a[i][j];
```

2）、给出边的顶点。

如输入文件：两个顶点及权值

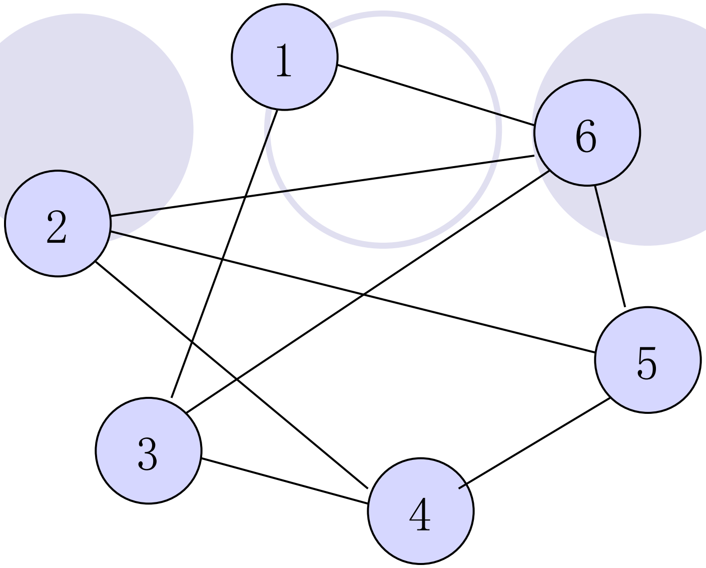
5
7
1 2 2
1 3 2
1 4 3
2 3 1
2 5 3
3 5 2
4 5 4




```
Cin>>n>>m;
for (int k=0;k<m;k++){
        cin>>i>>j>>x;
        a[i][j]=a[j][i]=x;
    }
```

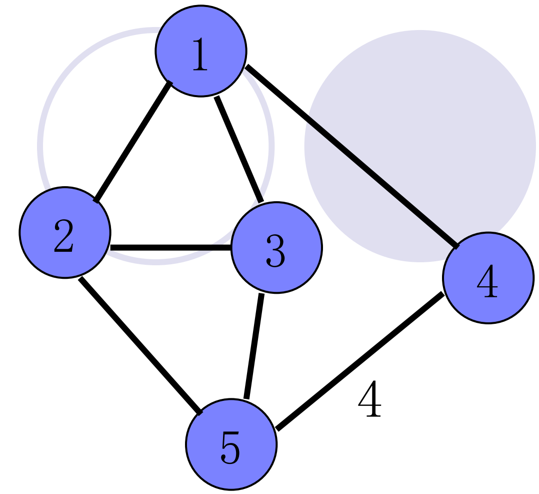
3)、给出每个顶点的邻接点

```
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
```



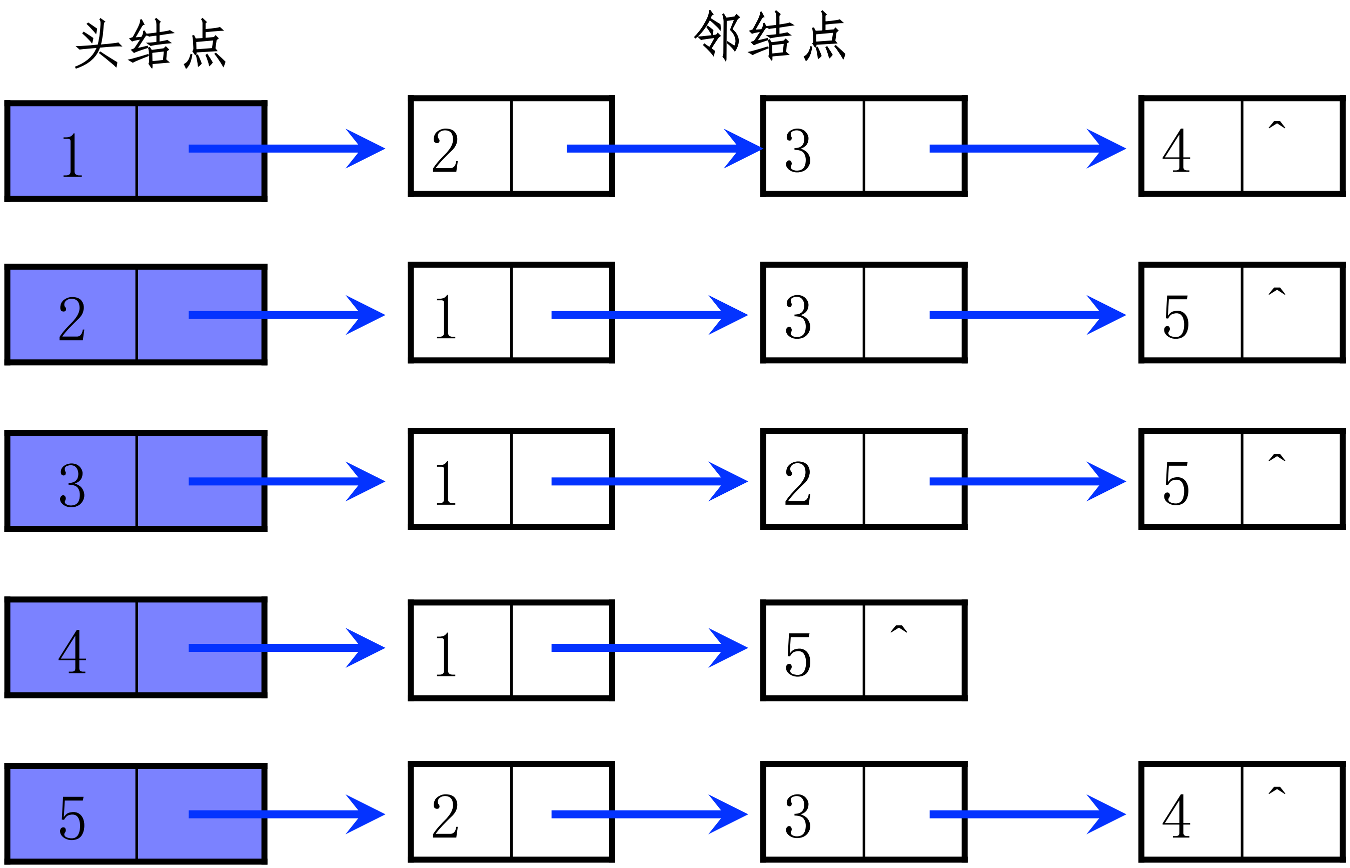

```
cin>>n
for (int i=0;i<n;j++){
    cin>>k
    for (int j=0;j<k;j++)
     {
        cin>>x;
        a[i][x]=a[x][i]=1;
      }
  }
```

2、邻接表



无权图:设置结点指针

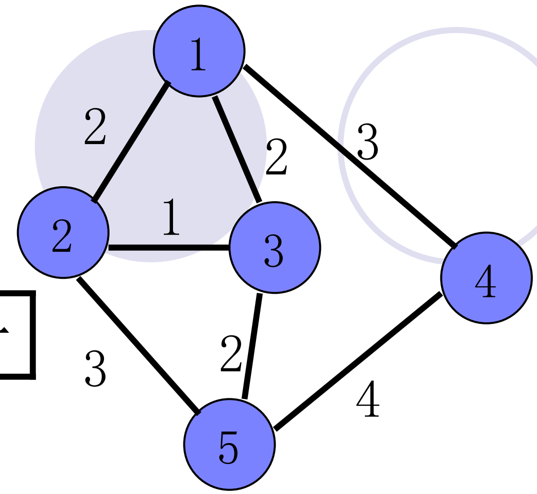
结点	邻接点指针



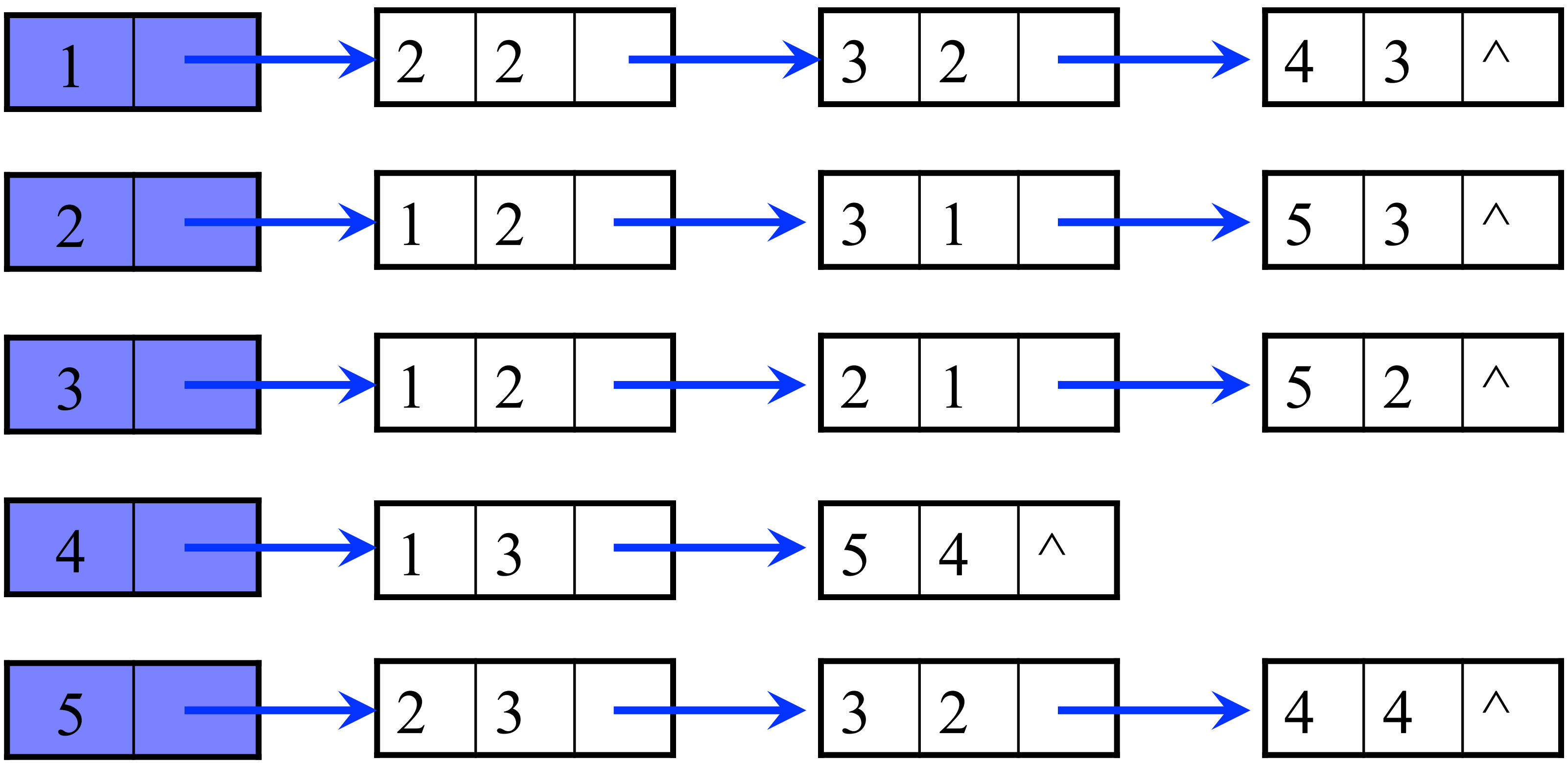
有权图：

结点	邻接点指针

邻接点	边权值	下一个邻接点指针



头指针　　　　邻接点指针



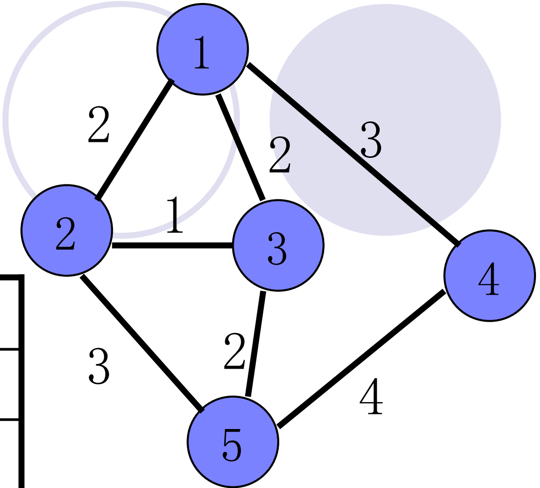
用数组模拟邻接表/边链表

head[x]数组

结点的编号	第一条边的地址
1	5
2	9
3	11
4	13
5	14

e数组

地址	Y结点	权值v	下条边
1	2	2	0
2	1	2	0
3	3	2	1
4	1	2	0
5	4	3	3
6	1	3	0
7	3	1	2
8	2	1	4
9	5	3	7
10	2	3	0
11	5	2	8
12	3	2	10
13	5	4	6
14	4	4	12



输入数据

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

示范

边链表

```
struct edge{
    int y,v,nxt;          //y表示这条边的终点编号，v是权值；
 } e[maxm+10];               //nxt表示同起点下条边的编号是多少
int head[maxn+10];     //起点表 head[x]表示由x出去的第一条边的下标是多少

void insert(int u,int v,int w)//u为起点,v为终点,w为权值。
{
   e[++tot].y=v; e[t].v=w;       //tot表示有tot条边,是个全局变量。
   e[tot].nxt=head[u]; head[u]=tot;
}

void init(){
    scanf("%d %d %d",&n,&p,&m);
   for (int i=0;i<m;i++)     {
        int xx,yy,zz;
        scanf("%d%d%d",&x,&y,&z);
     insert(x,y,z);
     insert(y,x,z);  //这里插入的是无向图，所以两条边都要插入。
  }
}
```

在使用边链表，有个很常见的写法

```
for (int i=head[x];i;i=e[i].nxt)
```

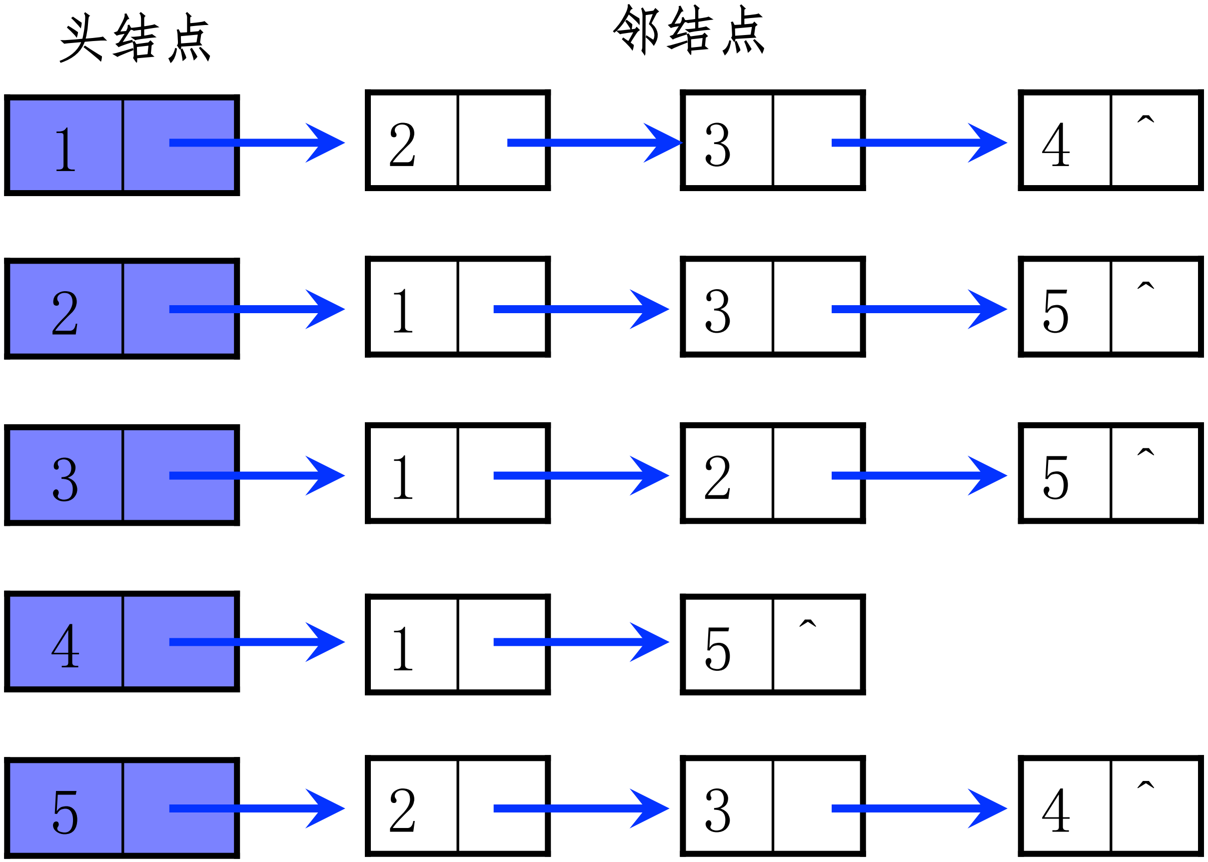
head[x]　head[x&1]，无向边的2条记录

可以很方便地引用。

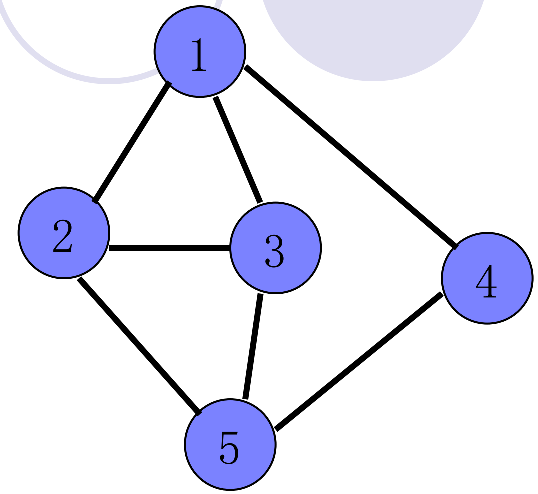
visit[x>>1]=false　可以很方便地删除无向边

现在边链表应用，还是比较常见的。

邻接矩阵和邻接表的优缺点：



邻接表



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

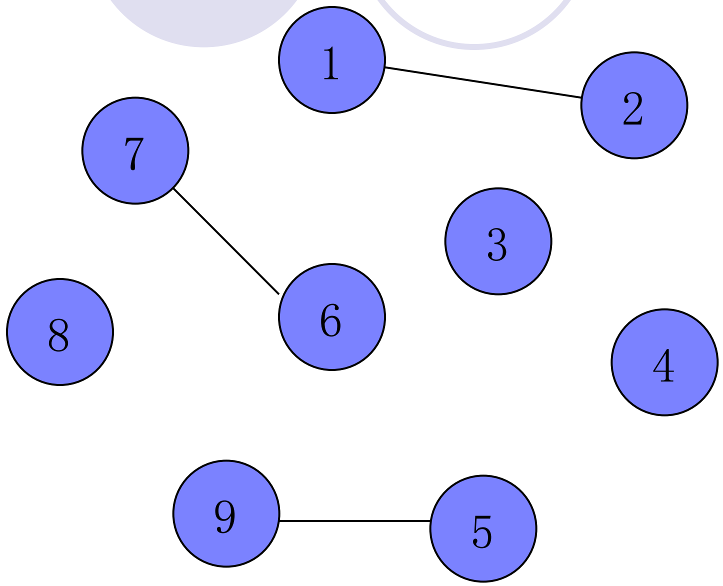
邻接矩阵

邻接矩阵：代码书写简单，找邻接点慢
采用二维数组的静态存储结构
一般点数|v|小于 等于5000的时候，用邻接矩阵。

邻接表：代码书写较复杂，找邻接点快
采用动态存储结构（指针或用数组模拟）
一般点数|v|大于等于5000，并且边得个数不是很多的时候，用邻接表，并且现在一般都是用数组来模拟。

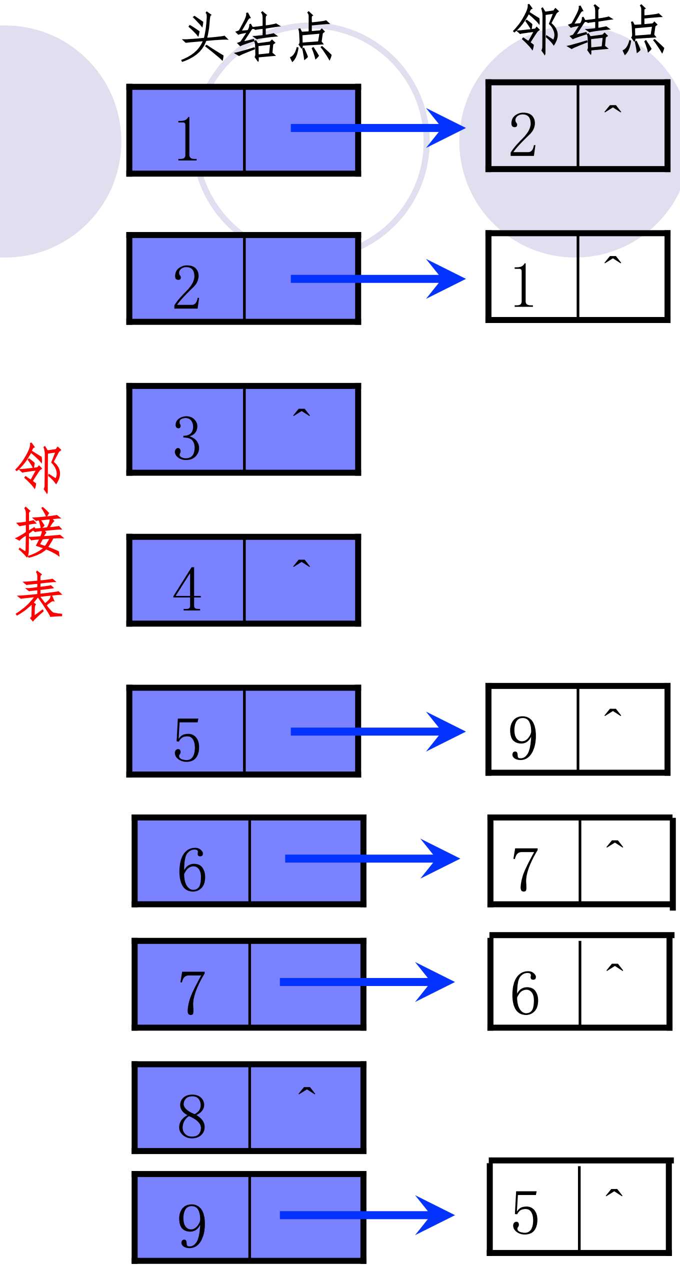
数组模拟邻接表的速度会快一点，并且能避免一些错误。

2、边集数组-稀疏图



0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0

邻接矩阵



3、边集数组-稀疏图

```
Struct edge{
    int x,y  //起点和终点
    int v    //权值
}e[maxm];

e[i].x;  e[i].y   e[i].v
```

实际问题，边集数组也大有可为

三、图的遍历

- 一般分为两种，广度优先遍历和深度优先遍历。
- 广度优先遍历：从一个点出发，按最短路径长度从小到大的顺序遍历，用一个队列实现。如果使用邻接矩阵，时间复杂度为O(N^2)；使用邻接表则时间复杂度为O(N+M)
- 深度优先遍历：从一个点出发，沿着边尽量走到没有访问过的顶点。如果没有未访问过的顶点，则沿边的相反方向退一步。一般用栈实现。用的空间较少，时间复杂度同上。

可行性遍历问题

- 常见的问题有下面两种

  1、Hamilton路问题

  通过图G中每个顶点一次且仅一次的路径称作该图的一条哈密尔顿路.

  2、Euler回路与Euler路问题

  在图G中存在一条回路把所边经过且每条边只经过一次。

- Hamilton问题已经被证明是NP完全问题，尚未发现多项式算法

- 下面介绍Euler回路与Euler路问题

1、Euler回路

- 无向图的Euler回路

  如果一个无向图所有的顶点的度为偶数，那么该图可以用起始点与终止点相同的一笔画出，这一笔经历的路线叫做无向图的Euler回路。

- 有向图的Euler回路

  如果一个有向图的所有顶点的入度等于出度，那么该图可以用起始点与终止点相同的一笔画出，这一笔经过的路径叫做有向图的Euler回路。

- 混合图的Euler回路？

求Euler回路的套圈算法

- 首先检查图G是否符合一笔画的条件。如果符合，那么标记顶点1为待查找状态。过程Euler(i)寻找开始于顶点i并且结束于顶点i的Euler回路，具体步骤如下：
  - 寻找从i出发的环 $P_1P_2 \cdots P_x$ ($P_1=P_x=i$)
  - 标记顶点$P_{1 \to x}$为待查找状态
  - 对所有处在待查找状态的结点P_j递归调用过程Euler(P_j)
  - 将Euler(P_j)找到的环$Q_1Q_2..Q_y$插入到环$P_1P_2 \cdots P_x$中，得到回路$P_1P_2 \cdots P_jQ_2 \cdots Q_yP_{j+1} \cdots P_x$
- 时间复杂度为O(M)

例题：灵魂画师

有一天一位灵魂画师画了一张图，现在要你找出欧拉回路，即在图中找一个环使得每条边都在环上出现恰好一次。一共两个子任务：

1、这张图是无向图。（50分）　2、这张图是有向图。（50分）

【输入格式】

第一行一个整数t，表示子任务编号。$t \in \{1,2\}$，

如果 t=1 则表示处理无向图的情况，如果 t=2 则表示处理有向图的情况。

第二行两个整数 n,m，表示图的结点数和边数。

下来m行中，第i行两个整数 vi,ui 表示第i 条边（从1开始编号）。

保证 $1 \leqslant vi,ui \leqslant n$。

如果 t=1 则表示 vi 到 ui 有一条无向边。

如果 t=2 则表示 vi 到 ui 有一条有向边。图中可能有重边也可能有自环。

【输出格式】

如果不可以一笔画，输出一行 “NO”。

否则，输出一行 “YES”，接下来一行输出一组方案。

如果t=1，输出m个整数 p1,p2,…,pm。令e=｜pi｜，那么e表示经过的第i条边的编号。如果pi 为正数表示从 ve 走到 ue，否则表示从ue 走到ve。

如果 t=2，输出m 个整数p1,p2,…,pm。其中pi 表示经过的第i 条边的编号。

解题分析

先从度数判断是否有解：

- 若G为有向图，欧拉回路的点的出度等于入度。
- 若G为无向图，欧拉回路的点的度数为偶数。
- 然后判断连通性，并且用套圈法输出路径。

一些小技巧：

- ①用边链表存图，如果是无向图则idx隔一个存一条边，且idx从2开始。这样写的作用就是可以良好地标记，比如第一条无向边里idx=2、3分别对应一条正反边，2和3除2都对应1，那么我们只需标记vis[1]就好了，因为欧拉回路只需要用到其中一条边。如果是有向图，那idx隔两个存一条边，有向图因为要防止两条当一条用，所以要idx要隔2存一条边。
- ②在遍历邻接表时加一个前向弧优化，快了超多。

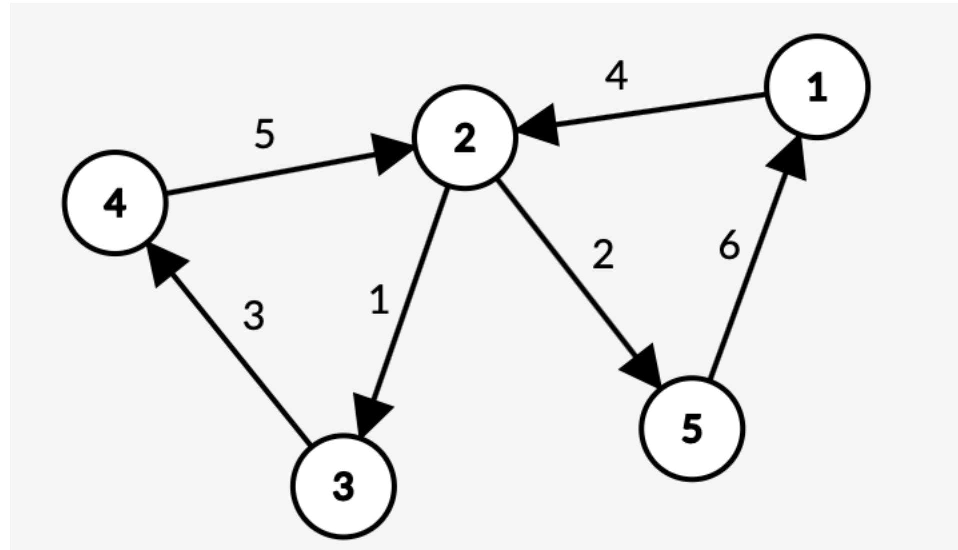
```
int main(){
    int t,a,b;
    scanf("%d%d%d",&t,&n,&m);
    init();
    for(int i=1;i<=m;i++){
        scanf("%d%d",&a,&b);
        addedge(a,b,i);//  i 边编号
        indegree[b]++;
        outdegree[a]++;
        if (t==1) //无向图
            addedge(b,a,-i);
        else
            tot++;// 有向图 偶数存 空一个
    }
    bool flag=true;
    if (t==1){
        for(int i=1;i<=n;i++){
            if ( (indegree[i]+outdegree[i])%2 ){
                flag=false;
                break;
            }
        }
    }
    else{
        for(int i=1;i<=n;i++){
            if (indegree[i]!=outdegree[i]){
                flag=false;
                break;
            }
        }

    }
```

```
void init (){
    tot=2;
}
void addedge(int u,int v,int w){
    e[tot].to=v;
    e[tot].w=w;//边编号
    e[tot].next=head[u];
    head[u]=tot;
    tot++;
}
int che;
void dfs(int u){
  // cout<<++che<<"dfs"<<u<<endl;
    for(int i=head[u];i;i=e[i].next){
        int y=e[i].to;
        if (!visit[i>>1]) {
            visit[i>>1]=true;
            dfs(y);
            ans[++cnt]=e[i].w;
        }
    }
}
```

这个实现复杂度不够好，每次应该直接把这条边从边标里干掉，如果只是打标记的话每次都要遍历全部边，这个会被卡成o(n^2)

- 如何保证每条边被访问一次

```
void dfs(int u){
   for(int i=head[u];i;i=e[i].next){
        int y=e[i].to;
        head[u]=i;////前向弧优化
        if (!visit[i>>1]) {
            visit[i>>1]=true;
            dfs(y);
            i=head[u];
            ans[++cnt]=e[i].w;
        }
    }
}
```




```
hzg — -bash — 80×24
Last login: Sun Dec  1 12:50:56 on ttys000
hzgdeMBP:~ hzg$ /Users/hzg/Desktop/图论在信息学竞赛应用
画师/uoj117
2
5 6
2 3
2 5
3 4
1 2
4 2
5 1
YES
6 4 1 3 5 2
hzgdeMBP:~ hzg$
```

- 在线画图 https://csacademy.com/app/graph_editor/

2、Euler路问题

- 如果一个无向图恰有两个顶点x,y的度为奇数，那么该图可以用起始点于x与终止点于y的一笔画出，这一笔经过的路线叫做无向图的Euler路。

- 如果一个有向图中，顶点x出度比入度大1，顶点y入度比出度大1，其余所有顶点入度等于出度，那么该图可以用起始点于x与终止点于y的一笔画出，这一笔经历的路线叫做有向图的Euler路。

- 求解Euler路问题可以沿用求解Euler回路的套圈算法解决。

例题　单词游戏

- 有N个盘子，每个盘子上写着一个仅由小写字母组成的英文单词。
- 你需要给这些盘子按照合适的顺序排成一行，使得相邻两个盘子中，前一个盘子上面单词的末字母等于后一个盘子上面单词的首字母。
- 请你编写一个程序，判断是否能达到这一要求。如果能，请给出一个合适的顺序。

例题 词链 PKU 2337/洛谷 p1127

题目描述

展开

如果单词X的末字母与单词Y的首字母相同，则X与Y可以相连成$X.Y$。（注意：X、Y之间是英文的句号“.”）。例如，单词 `dog` 与单词 `gopher`，则 `dog` 与 `gopher` 可以相连成 `dog.gopher`。

另外还有一些例子：

`dog.gopher`

`gopher.rat`

`rat.tiger`

`aloha.aloha`

`arachnid.dog`

连接成的词可以与其他单词相连，组成更长的词链，例如：

`aloha.arachnid.dog.gopher.rat.tiger`

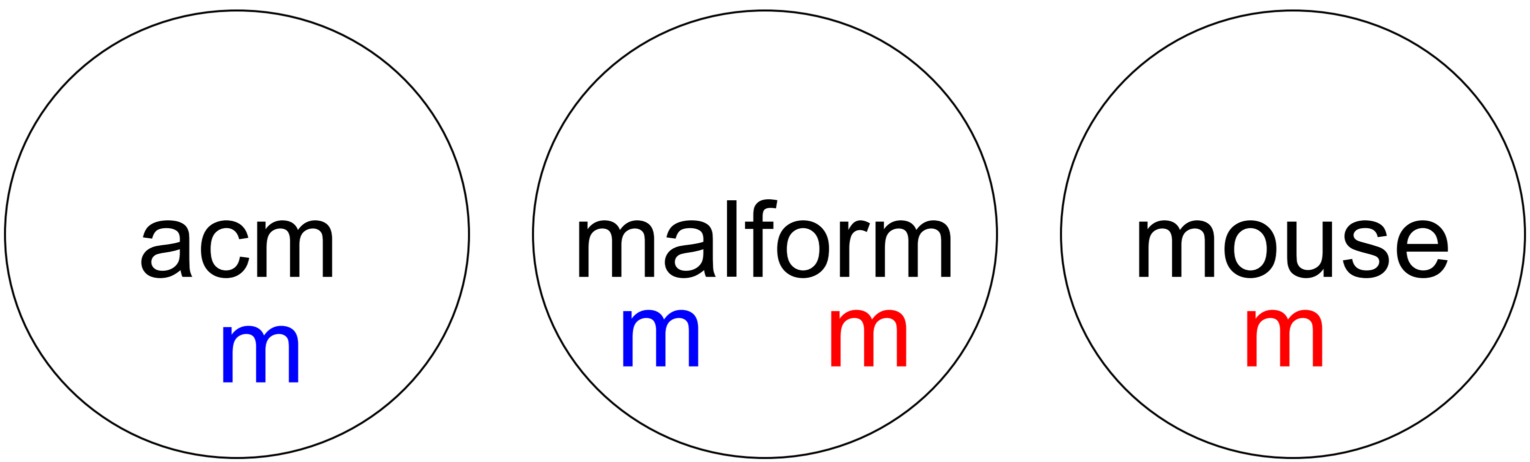
注意到，“.”两边的字母一定是相同的。

现在给你一些单词，请你找到字典序最小的词链，使得这些单词在词链中出现且仅出现一次。

输入格式

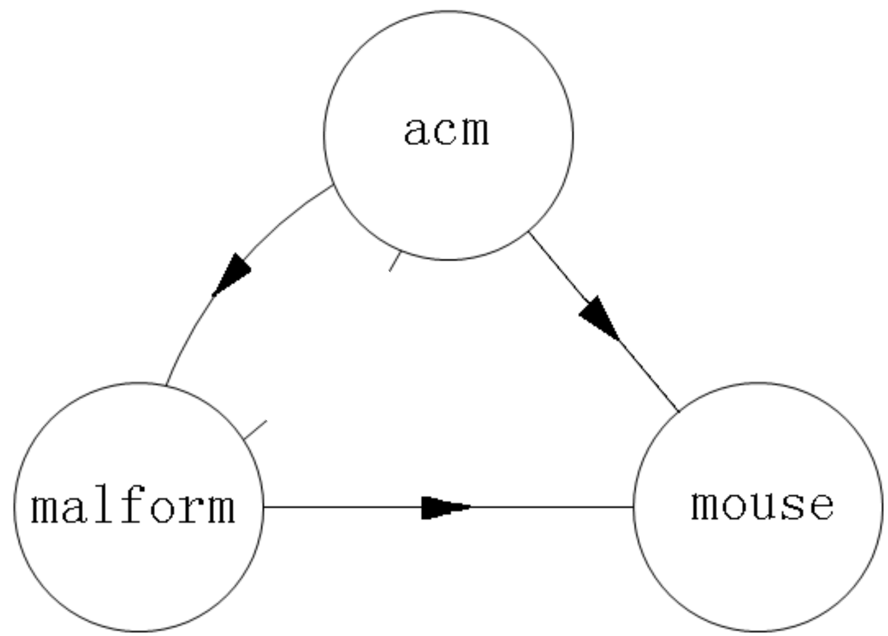
第一行是一个正整数$n(1 < n < 1000)$，代表单词数量。

样例



模型1

- 将每个单词看作一个顶点。
- 如果单词B能连接在单词A后面，那么从A向B连一条有向边。



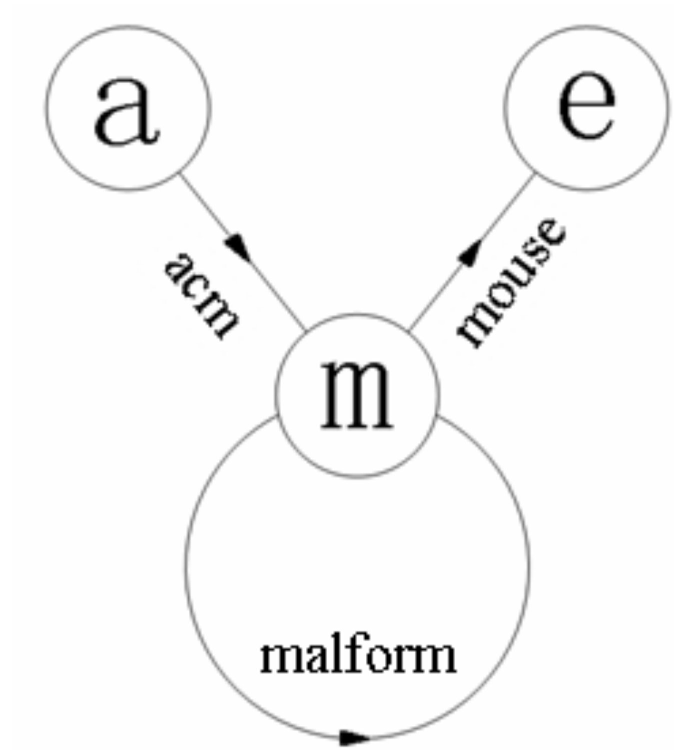
模型1

- 问题转化为在图中寻找一条不重复地经过所有顶点的路径，即**哈密尔顿路**。
  见代码实现，p1127-dfs.cpp
  时间复杂度大概 O(n^n) 的吧,状态压缩可以到O(2^n*n).没写过
- 但是，求哈密尔顿路是一个十分困难的问题，这样的建模没有给解题带来任何便利。我们必须另辟蹊径。

```
void add_edge(int u,int v){
    to[++en]=v,nxt[en]=head[u],head[u]=en;
}
void dfs(int u){//暴力 来走哈密尔顿路 不重复地遍历所有的点
    if (flag) return ;
    sta[top++]=u;
    vis[u]=true;
    int v;
    for (int i=head[u];i;i=nxt[i]){
        v=to[i];
        if (!vis[v]) dfs(v);
    }
    if (top==n) flag=true;
    else vis[u]=false,--top;
}
int main(){
    cin>>n;
    for (int i=1;i<=n;i++)
        cin>>s[i];
    sort(s+1,s+n+1);
    for (int i=1;i<=n;i++)
        len[i]=s[i].size();
    for (int i=1;i<=n;i++)
        for (int j=n;j>=1;j--)
            if (i!=j && s[i][len[i]-1]==s[j][0])
                add_edge(i,j);
    for (int i=1;i<=n;i++)
        cnt[s[i][0]-'a']++,cnt[s[i][len[i]-1]-'a']--;
        //  欧拉的计算 和判断
    int k=0,h;
    for (int i=0;i<26;i++){
        if (cnt[i]==1) k++,h=i;
        if (cnt[i]==2) k=2;
        if (k==2) break;
    }
    if (k==2) cout<<"***\n";
```

模型2

- 以26个英文字母作为顶点。
- 对于每一个单词，在图中从它的首字母向末字母连一条有向边。



模型2

- 问题转化为在图中寻找一条不重复地经过所有边的路径，即**欧拉路径**。
- 这个问题能够在O(|E|)时间内解决。

```
void dfs(int a, int id){
    int i = x[a], b;
    while(d[i].a == a){
        if(!v[i]){
            v[i] = 1;
            dfs(d[i].b, i);
        }
        i++;
    }
    ans[++m] = id;
}
int main(){
    int i, j, n, a, b, c;
    scanf("%d", &n);
    for(i = 1; i <= n; i++){
        scanf("%s", d[i].s);
        c = strlen(d[i].s);
        d[i].a = d[i].s[0];
        d[i].b = d[i].s[c-1];
    }
    sort(d + 1, d + i);
    for(i = 1; i <= n; i++){
        du[d[i].a]++, du[d[i].b]++;
    }
    for(i = 'z'; i >= 'a'; i--) if(du[i] & 1) cnt++, p = i;
    if(cnt == 1 || cnt > 2){
        printf("***");
        return 0;
    }
    if(!p) p = d[1].a;
    for(i = n; i >= 1; i--) x[d[i].a] = i;
    dfs(p, x[p]);
    if(m != n + 1){
```

小结

- 比较以上两个模型，模型1过于直接，模型2则打破了“顶点表示元素，边表示元素之间关系”的思维定势，将元素表示在边上，而顶点则起到连接各个元素的作用。

例题：字符串（string）

【题目描述】

有一个长度为n的字符串，但他不小心把这个字符串丢掉了。幸运的是，他曾经记下了这个字符串的所有长度为m的子串。请你帮他还原出这个字符串。

【输入数据】

第一行两个整数n,m。接下来n-m+1行每行一个长度为m的字符串。字符集为小写字母。

【输出数据】

一行一个长度为n的字符串表示答案。保证有解，如果有多组解可以输出任意一组。

【样例输入】

6 3

aba

aab

abb

bab

【样例输出】

aababb

【数据范围】

对于10%的数据，n<=10。

对于另外30%的数据，m=2。

对于另外20%的数据，保证原字符串随机且m>=200。

对于100%的数据，2<=m<=n<=100000，n*m<=200000。

- 考虑m=2的情况。假设子串ab出现了k次，那么原串中就有恰好k个b接在a后面。那么对每个子串ab我们就连一条a到b的边，然后求出欧拉路就能还原出字符串。
- m>2时，类似地，每个子串连一条(前m-1个字符构成的字符串)到(后m-1个字符构成的字符串)的边，然后求欧拉路。
- 求欧拉路的经典算法：找到合适的起点开始dfs所有点，按回溯顺序记下每条边，逆序就是答案。
- 时间复杂度O(nm)

m=2
aa
ab
ba
ab

m=3
aba
aab
abb
bab

```cpp
string s,w[100010],ans;
map<string,int> h;
vi a[100010];
bool q;
inline void edge(string s,string t){
    if(!h[s]){
       h[s]=++p;
       w[p]=s;
      }
    if(!h[t]){
       h[t]=++p;
       w[p]=t;
      }
    int i=h[s],j=h[t];
    a[j].pb(i);
    x[j]++;
    x[i]--;
}
inline void add(int i){
    if(!q)
      {
       q=1;
       ans=w[i];
      }
    else
      ans.pb(w[i][m-2]);
}
inline void dfs(int i){
    int j,k;
    while(a[i].size()){
       j=a[i].size()-1;
       k=a[i][j];
       a[i].resize(j);
       dfs(k);
      }
    add(i);
}
```

```cpp
}
int main()
{
    freopen("string.in","r",stdin);
    freopen("string.out","w",stdout);
    int i;
    scanf("%d%d",&n,&m);
    for(i=1;i<=n-m+1;i++)
      {
       cin>>s;
       edge(s.substr(0,m-1),s.substr(1,m-1));
      }
    for(i=1;i<=p;i++)
      if(x[i]==1)
        break;
    if(i>p)
      i=1;
    dfs(i);
    cout<<ans;
    return 0;
}
```

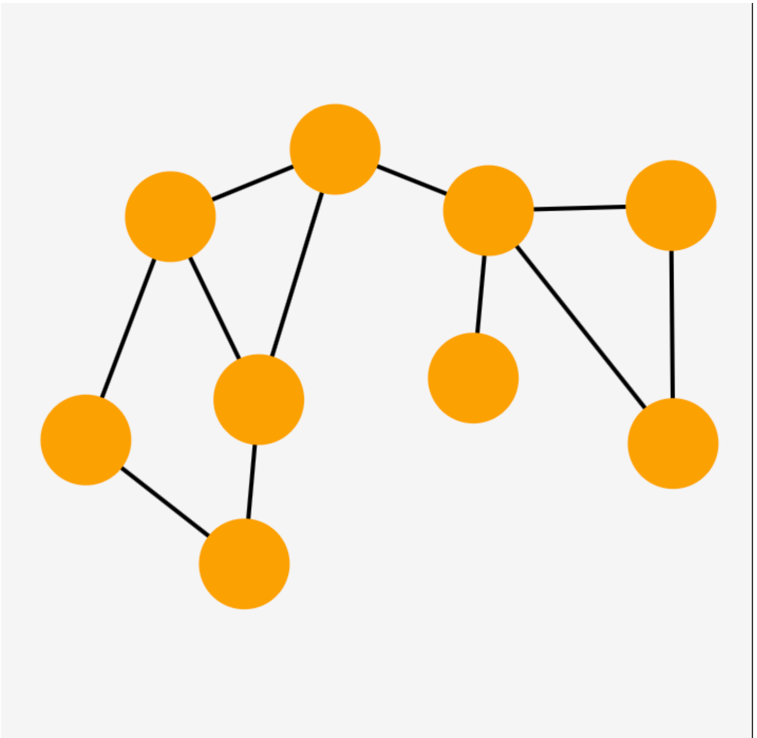
四、Tarjan在无向图中的应用

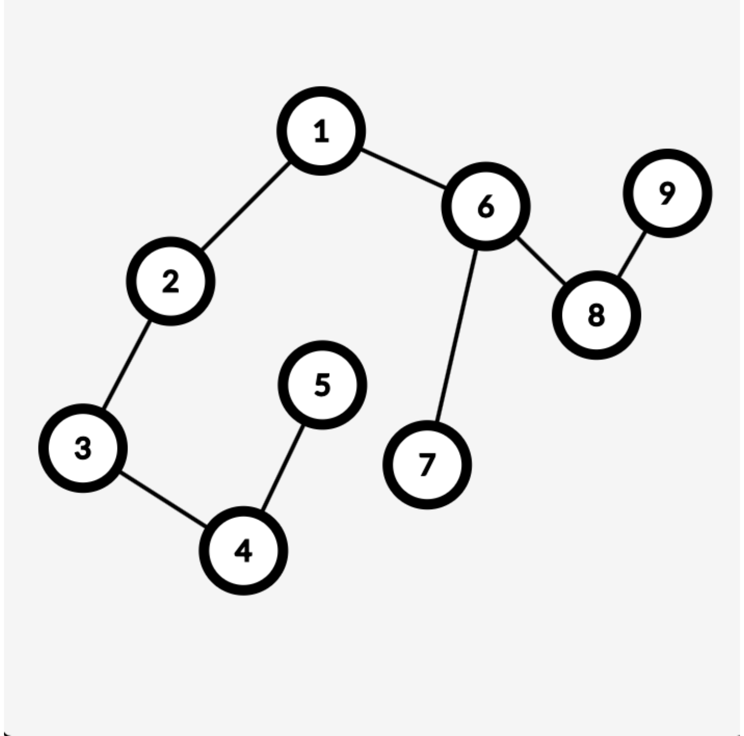
Tarjan其人

- Tarjan，他曾在AT&T贝尔实验室（1980-1989），浩信科技（1997-2001），康柏（2002年）和惠普（2006年至今）工作
- 设计了众多算法及数据结构：并查集、LCA的离线求法、求强连通分量、LCT、splay、斐波那契堆等
- 1986年获图灵奖
- 统称为Tarjan算法。其中最著名的有三个，分别用来求解
  - 1）无向图的双连通分量
  - 2）有向图的强连通分量
  - 3）最近公共祖先问题

在Tarjan算法里面，有三个概念非常重要：

- 一个是搜索树,无向连通图中任选一个点出发进行dfs，所发生递归的边构成一棵树。在图论的连通性问题中，从搜索树的角度来进行分析，常常会让思路更清晰。
- 一个是时间戳，dfn[x]；意为深度优先数，即代表节点x第一次被访问的时间顺序；





一个是追溯值，low[x]意为通过反向边能到达的最小dfn。

即下列节点的时间戳的最小值：

1、x为根的子树中的节点；

2、通过1条不在搜索树上的边，能够到达x子树内，这样的节点；

tarjan 算法核心，dfs遍历每个节点x,求出low[x]:

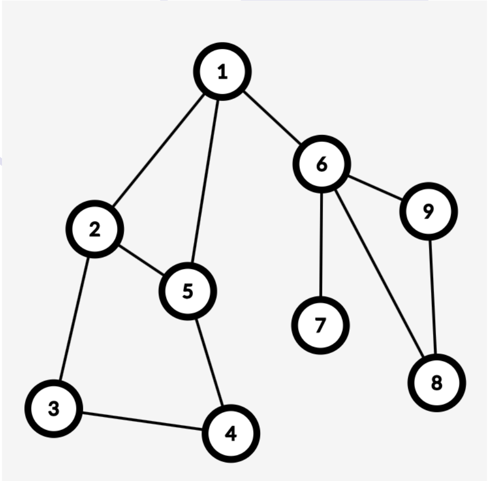
第一次访问节点x，先low[x]=dfn[x]=时间戳;

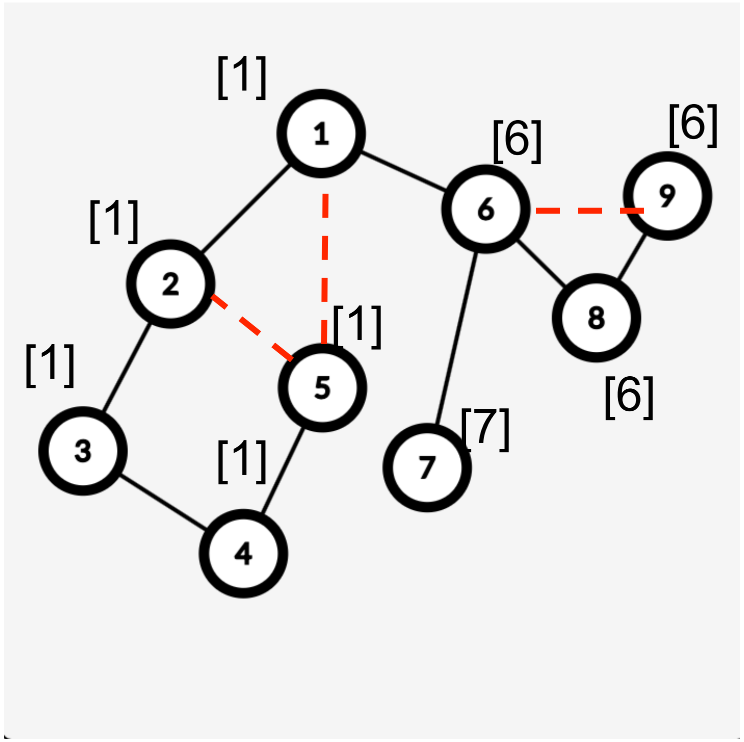
对x出发的每条边(x,y)，dfs(y);

如果是搜索树上的边，

low[x]=min(low[x],low[y])

如果不是搜索树上的边,low[x]=min(low[x],dfn[y])





无向图中割点的概念:

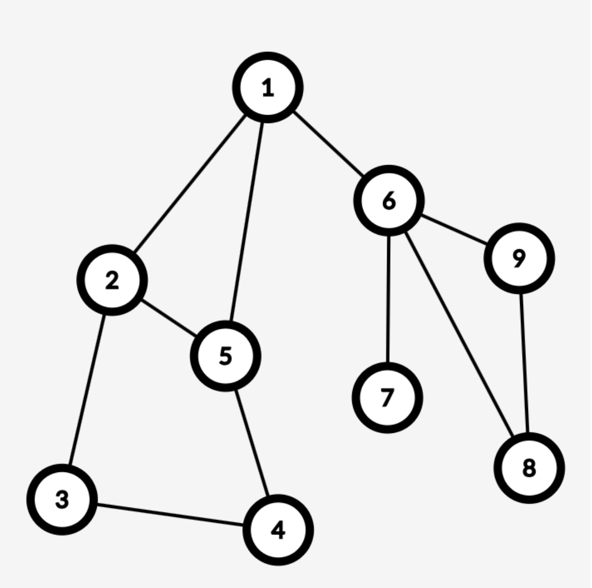
割点：一个结点称为割点（或者割顶）当且仅当去掉该节点及其相关的边之后的子图不连通。

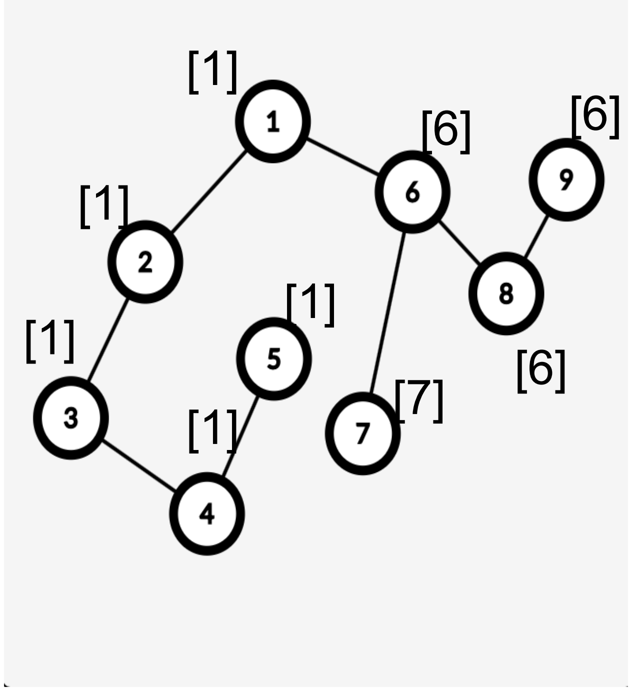
若x不是搜索树的根节点，则x是割点当且仅当搜索树上存在x的一个子节点y，满足:

dfn[x]<=low[y]

若x是搜索树的根节点，则x是割点当且仅当搜索树上存在至少2个不同子树上的子节点y1,y2，满足

dfn[x]<=low[y1] &&dfn[x]<=low[y2]





见代码实现tarjan_cut.cpp

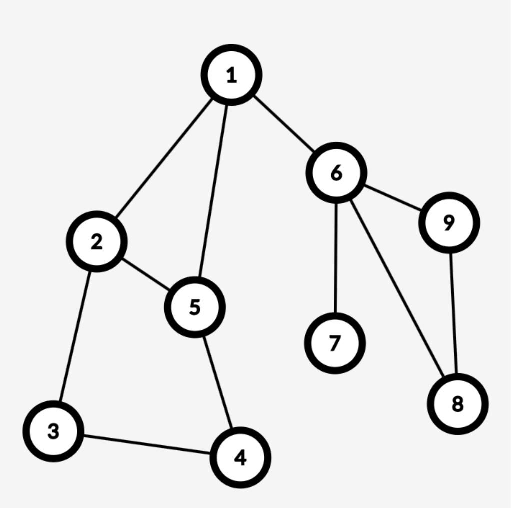
无向图中桥的概念:

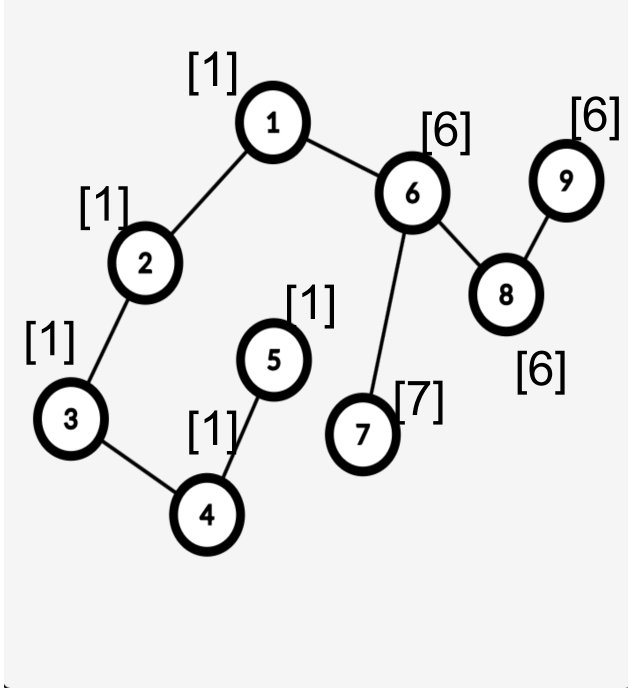
桥:一条边称为桥(或者割边)当且仅当去掉该边之后的子图不连通。

无向图上的一条边(x,y)是桥,当且仅当搜索树上存在x的一个子节点y,满足

dfn[x]<low[y]

说明从y的子树出发,在不经过(x,y)的前提下,不管走什么边,都无法到达x或比x更早访问的节点。若(x,y)删除,则y子树就好像形成一个封闭的环境,与节点x没有边相连。





见代码实现tarjan_brige.cpp

无向图的双连通分量

- 若一张无向连通图不存在割点，则称它为“点双连通图”
- 若一张无向连通图不存在桥，则称它为“边双连通图”
- 无向图的极大点双连通子图成为“点双连通分量”，“v-DCC”

vertex double connected component

- 无向图的极大边双连通子图成为“边双连通分量”，“e-DCC”

edge double connected component

定理：

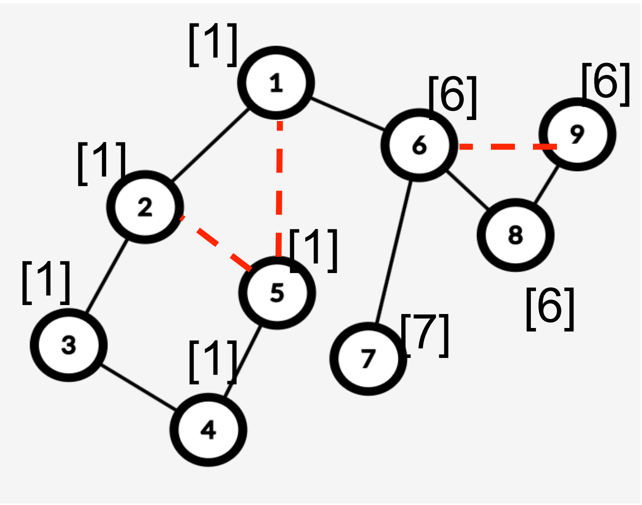
- 一张无向连通图是“点双连通图”，当且仅当满足下列两个条件之一：

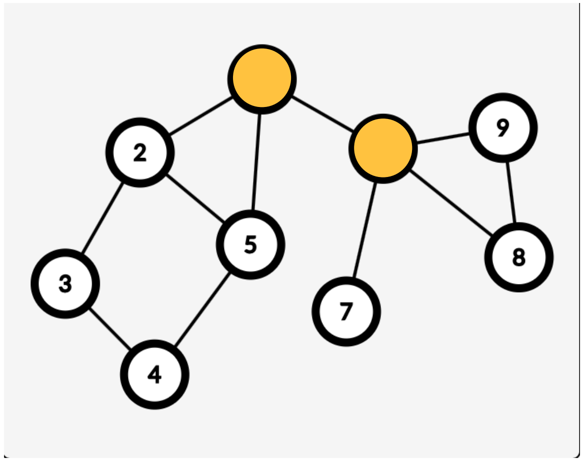
1、图的顶点数不超过2.

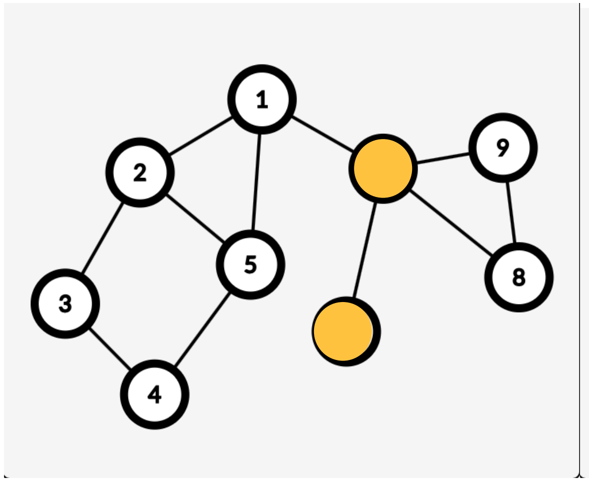
2、图中的任意两点都同时包含在至少一个简单环中，其中简单环指的是不自交的环，也就是我们通常画出的环。（证明略）

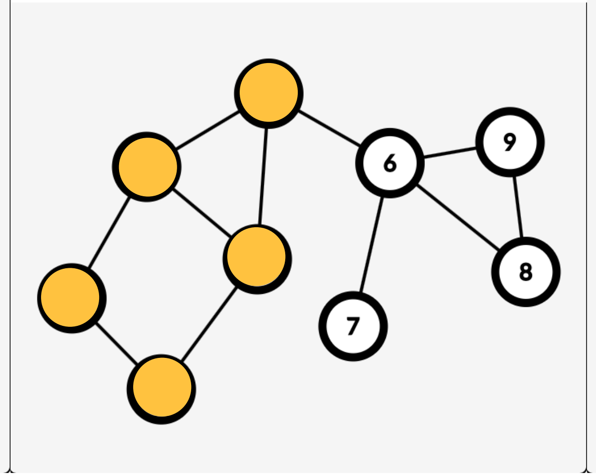
- 一张无向连通图是“边双连通图”，当且仅当任意一条边都包含在至少一个简单环。

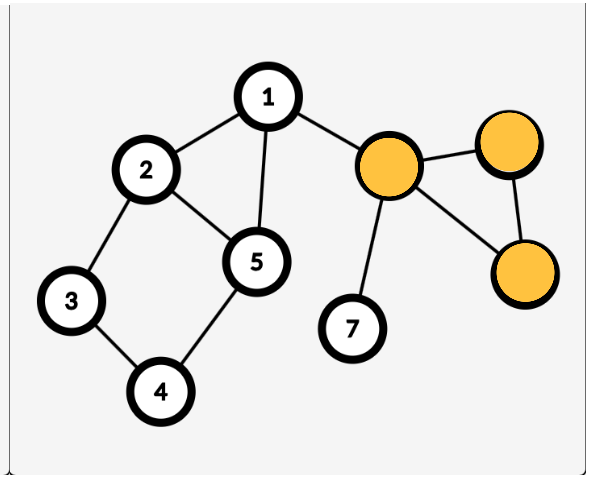
求“点双连通分量 v-DCC”











求“点双连通分量 v-DCC”

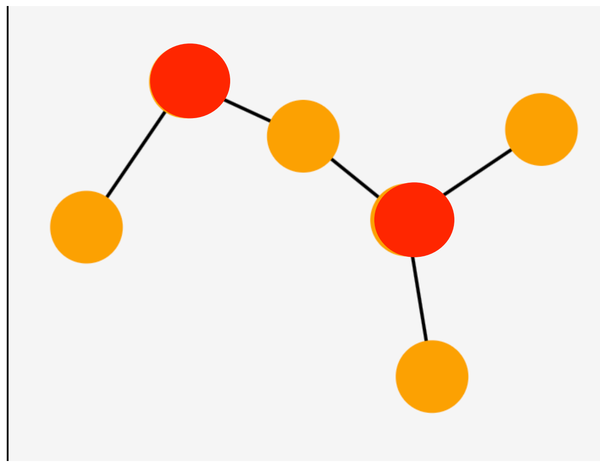
- 解法：结合tarjan算法

1、当一个节点第一次被访问时，把该节点入栈；
2、根据割点判断法制，当递归回溯时发现dfn[x]<=low[y] 时，无论x是否为根则从栈顶不断弹出节点，直至节点y弹出；刚才所有弹出的节点与节点x构成一个点双连通子图v-dcc；

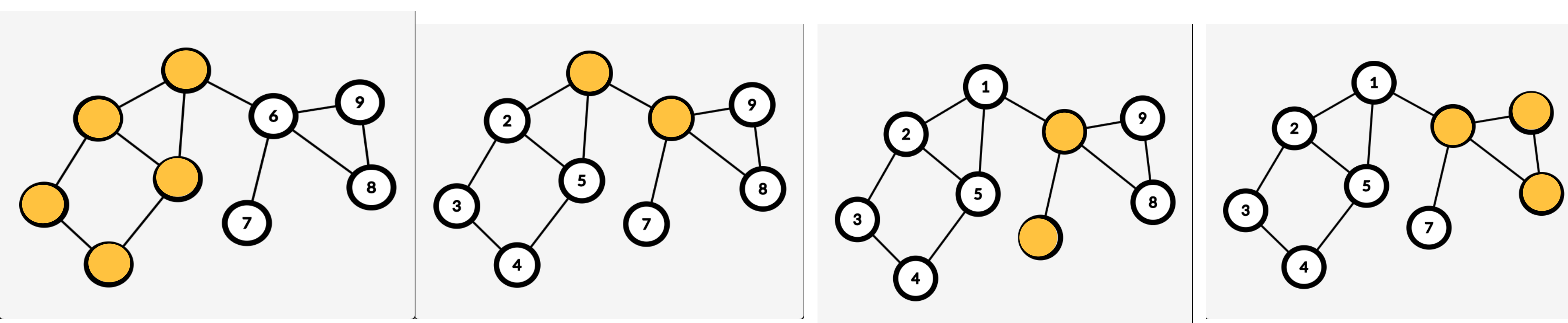
- 见代码实现 v-dcc.cpp

将“点双连通分量”缩点

- 解法：v-dcc的缩点比e-dcc的缩点更复杂。因为一个割点可能属于多个v-dcc。
- 设图中共有p个割点和t个v-dcc。缩点后就会包含p+t个节点的新图。把每个v-dcc
- 和每个割点都作为新图中的节点，并在每个割点与包含它的所有v-dcc之间连边。



- 见代码实现 e-dcc-sd.cpp

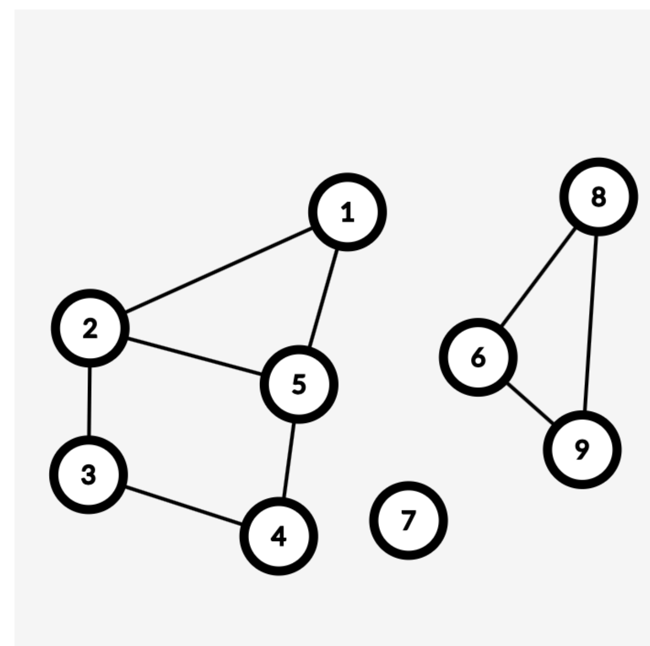


求“边双连通分量/e-DCC”

- 解法：求出无向图中所有的桥，把桥都删除后，余下的若干连通块，每个连通块就是一个边双连通分量

实现：先用tarjan算法标记出删除所有的桥。然后对整个无向图dfs，标记出每个连通块。

- 见代码实现 e-dcc.cpp

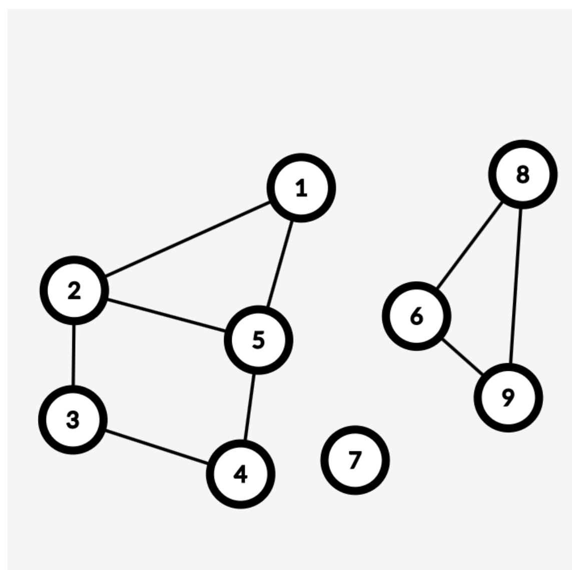


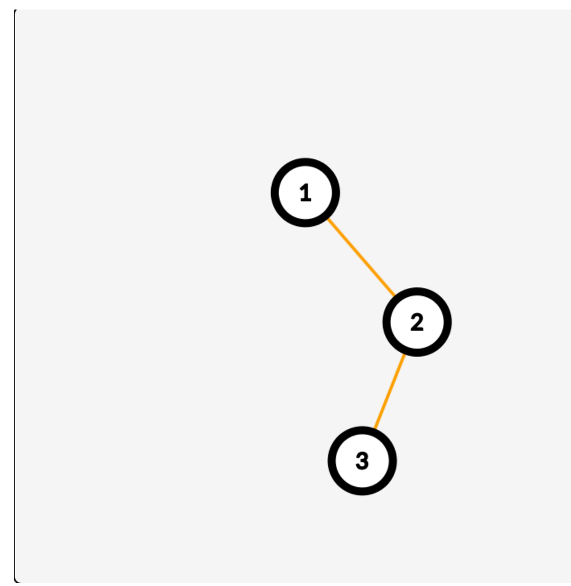
将“边双连通分量”缩点

- 解法：求出无向图中所有的桥，把桥都删除后，余下的若干连通块，每个连通块就是一个边双连通分量。然后把一个边双连通分量看成一个节点，把桥(x,y)看作连接c[x]和c[y]的无向边，构建出一个树（森林）。

实现：需要开一个新的邻接表。

- 见代码实现
- e-dcc-sd.cpp





五、Tarjan算法在有向图中的应用

给定有向图G=(V, E)，若满足从一个点r出发能够到达V中的所有的点，称G是一个“流图”(Flow Graph)，记为(G,r)，称r为流图的源点。

与无向图的dfs遍历类似，定义“流图”的三个重要概念：搜索树、时间戳dfn[x]、追溯值low[x]。

- 搜索树：从源点r出发dfs遍历，每个点只访问一次，所发生递归的边构成一棵以r为根的树，称为流图(G,r)的搜索树。
- 时间戳：在dfs遍历过程中，按照每个节点第一次被访问的时间顺序，一次给予每个节点1～N的整数标记，即时间戳dfn[x]，
- 流图中的每条有向边(x,y)，必然是以下四种之一：

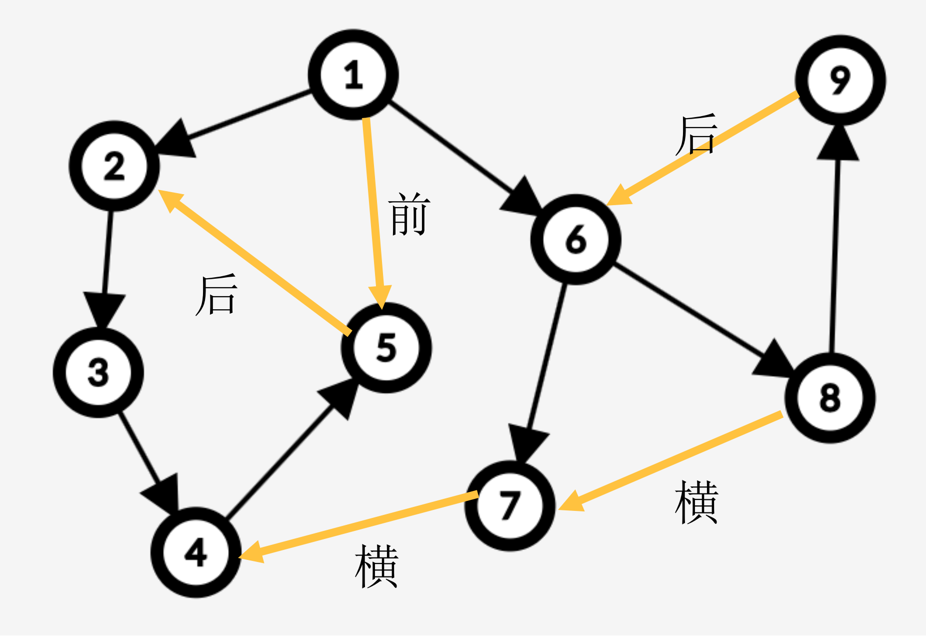
1、树枝边，指搜索树中的边，即x是y的父亲节点；

2、前向边，指搜索树中x是y的祖先节点；

3、后向边，指搜索树中y是x的祖先节点；

4、横叉边，指除了以上三种，

满足dfn[y]<dfn[x]



有向图的强连通分量

- 给定一张有向图，若对于图中任意两个节点x,y,即存在从x到y的路径，也存在在从y到x的路径，则称该有向图为强连通图。
- 有向图的极大强连通子图称为“强连通分量”，记为SCC。即

Strongly Connected Component。含义与双连通分量“极大”含义相类似。

一个环一定是强连通图，Tarjan算法的核心就是对于搜索树上的每个点，尽量找到与它一起能构成环的所有节点。
“树枝边”和“前向边”
“后向边”(x,y)非常有用，可以和搜索树上从y到x的路径一起构成环。
“横向边”(x,y)如果能从y出发找到一条路径回到x的祖先节点，
那么(x,y)就是有用的。

在dfs的过程中，怎么找到通过“后向边”和“横向边”构成的环？？

开一个栈，维护如下节点信息：

- 1、当前访问到x节点，栈中保存了搜索树上x的祖先节点。记集合anc(x). 如果当前节点x有后向边(x,y)，y属于这个集合anc(x)。则找到

已经被遍历但尚未检查完毕(即DFS树中当前结点x与根的路径上的点)；

- 2、保存已经访问过的节点，这些节点中有节点y存在一条路径到达x祖先即集合anc(x);即找到一条有效的横向边。

综上，栈中的节点就是能与从x出发的“后向边”和“横向边”形成环的节点。

追溯值

设subtree(x) 表示流图的搜索树中以x为根的子树，low[x]定义为满足以下条件的节点的最小时间戳：

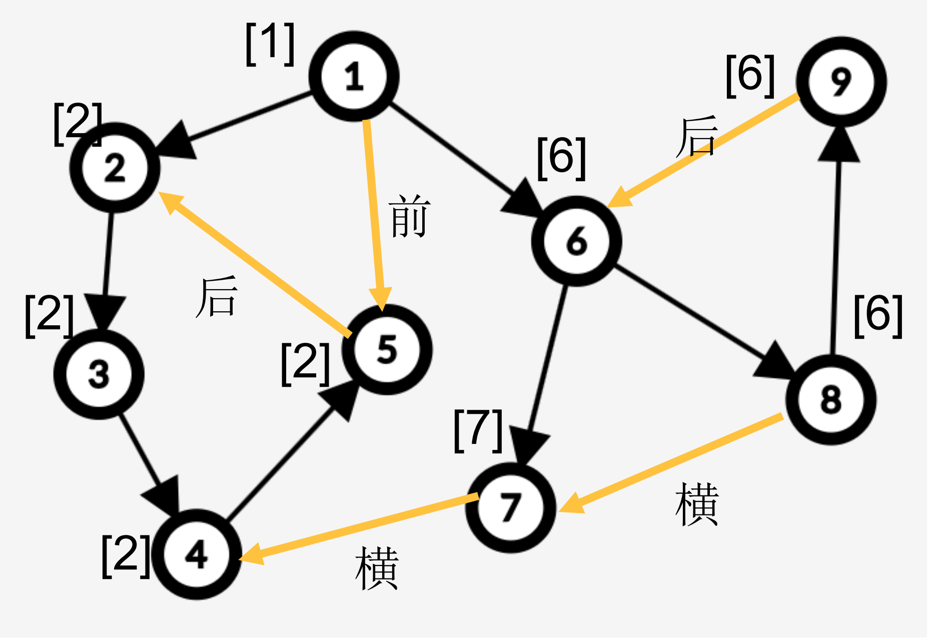
- 1、该点在栈中
- 2、存在一条从subtree(x)出发的有向边，以该点x为终点

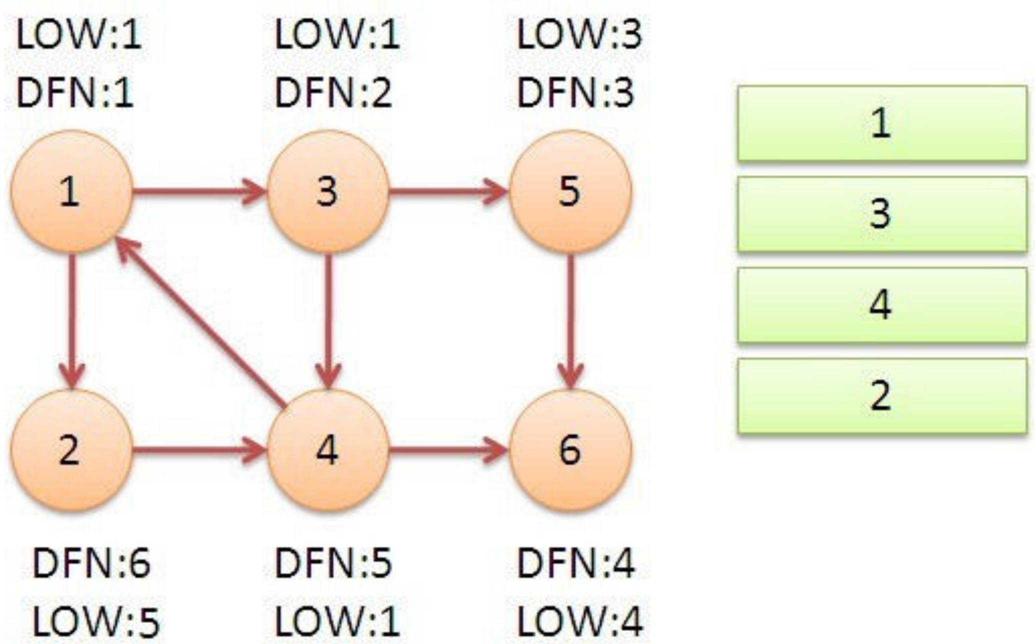
计算”追溯值”low[x]的流程:

- 1、第一次访问节点x，x入栈，low[x]=dfn[x];
- 2、扫描从x出发的每条边(x,y)

  (1) 若y没被访问过，则说明(x,y)是树枝边，递归访问y。回溯后 low[x]=min(low[x],low[y])

  (2) 若y被访问过并且在栈中，则low[x]=min(low[x],dfn[y]);
- 3、从x出发的每条边(x,y)，都访问过后。判断是否有low[x]=dfn[y]。若成立，则不断从栈中弹出节点，直至x出栈;





- 强连通分量 在计算追溯值的过程中，若从x回溯前，有low[x]=dfn[x]成立，则栈中从x到栈顶的所有节点构成一个强连通分量

CF427C Checkposts

一共给你N个点，M条有向边。其中每个点都有其自己对应的权值，作为城市的市长，你希望设定警察局来保护所有的城市。如果我们在点i处设立了一个警察局，那么其点i是被保护的，而且如果一个点j，能够保证有路径从i到j，并且能够保证有路径从j回到i，那么点j也是被保护的。 问将所有城市都保护起来的最小花费，以及对应最小花费有多少种设定的方式。

题目分析

首先对于第一问肯定是求强连通分量，然后对每个强连通分量的中的最小值求和即可。对于第二问，算出每个强连通分量内有多少个该分量中的最小值，然后根据乘法原理乘起来就行了。注意取模是针对第二问，第一问不用取模！

- 见代码实现 scc.cpp

六、Tarjan算法离线求LCA

LCA 问题有多种求法，Tarjan 求 LCA 是一种离线的算法，
也就是说它一遍求出所有询问的点对(x,y)的LCA，而不是要求哪两个点再去求。

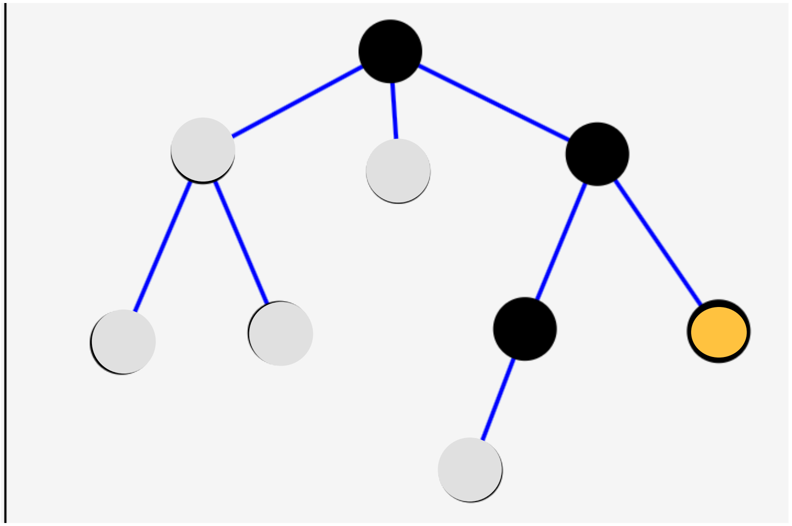
- Tarjan 是一种 DFS 遍历的思想：

从根结点去DFS遍历这棵树时，原树中的节点分为三类：

1. 已经访问完毕并且回溯的节点，记为灰色点；

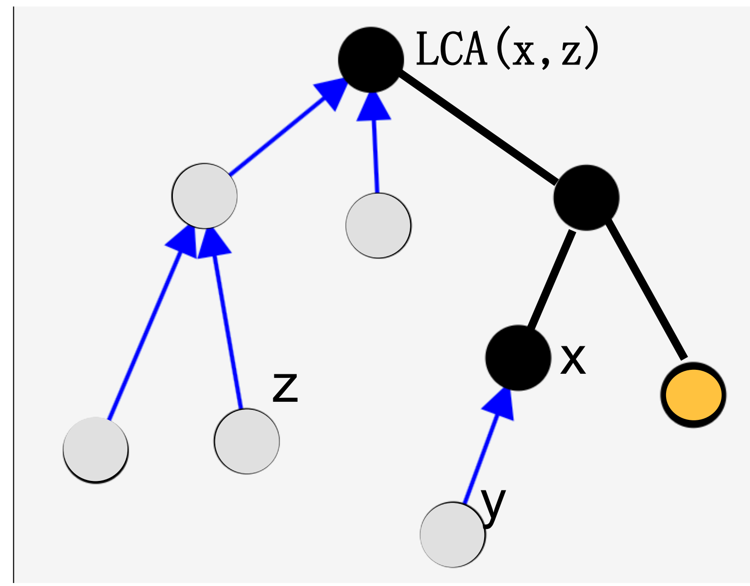
2. 已经递归访问但还没回溯的节点。这些节点是当前正在访问的节点x到根的路径上的节点，记为黑色点；

3. 还没访问的节点，极为橙色点



当遍历到某一个结点(称之为 x) 时，

- 1.将当前结点标记为已经访问。
- 2.递归遍历所有它的子节点(称之为 y)，并在递归执行完后用并查集合并 x 和 y。
- 3.完成第2步，检查与当前节点x有查询关系的结点(称之为 z)，如果 z 已经访问，那么 x 与 z 的 LCA 就是getfa(z)，也就是z向上走到根，第一个遇到的黑色节点，(这个利用并查集来优化实现)，输出或者记录下来就可以了。



- 见代码实现 tarjan-lca.cpp

七、生成树问题

- 最小生成树(Prim,Kruskal)
- 最大边最小的生成树(Kruskal)
- 最大边与最小边差最小的生成树
  - 枚举最小边,然后在比它权值大的边集中用Kruskal求解。
- K小生成树(次小生成树)
  - 首先可以证明，次小生成树为最小生成树替换一条边后得到。构造最小生成树，设生成树为T。然后枚举非生成树的边(x,y)，加入该边后就需要删除在T上x,y之间路径上的边权最大的边。因此定义f(i,j)为生成树上i结点到j结点间路径上的最大值边，其中i,j∈T。所以答案就为min(|T|+w(i,j)-f(i,j))
  - 算法瓶颈在于求f(i,j)。如果硬做是$O(N^2)$，可以利用LCA+倍增思想做到O(MlogN)的复杂度。

- 谢谢，聆听！