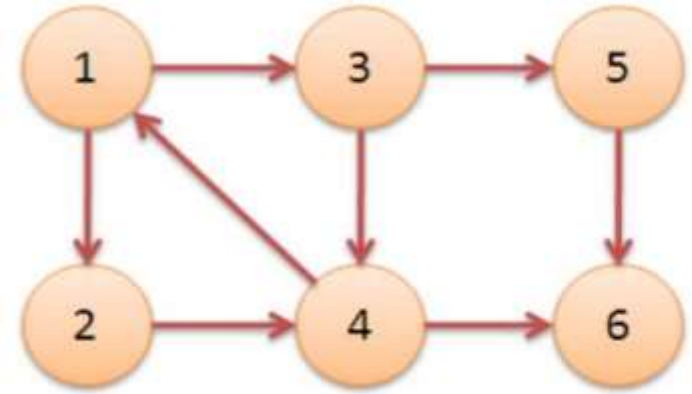
图的连通性

衡水中学

有向图的强连通分量

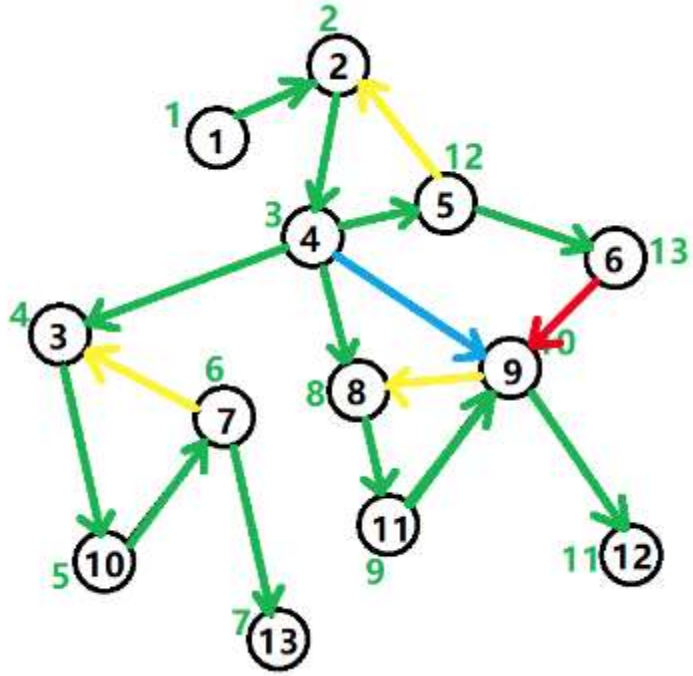
- 在有向图 G 中，如果两个点设两个点 a，b，由 a 有一条路可以走到 b，同时由 b 又有一条路可以走到 a，称两个顶点 a，b 强连通。
- 如果有向图 G 的每两个顶点都强连通，称 G 是一个强连通图。
- 非强连通图有向图的极大强连通子图，称为强连通分量。

有向图的强连通分量



在上图中，{1, 2, 3, 4}，{5}，{6} 三个区域可以相互连通，称为这个图的强连通分量

DFS生成树的边分类



- 树枝边：图中绿色边，DFS 时经过的边，即 DFS 搜索树上的边
- 反祖边：图中黄色边，也叫回边或后向边，与 DFS 方向相反，从某个结点指向其某个祖先的边
- 横叉边：图中红色边，从某个结点指向搜索树中另一子树中的某结点的边，它主要是在搜索的时候遇到了一个已经访问过的结点，但是这个结点并不是当前结点的祖先时形成的
- 前向边：图中蓝色边，与 DFS 方向一致，从某个结点指向其某个子孙的边，它是在搜索的时候遇到子树中的结点的时候形成的

Tarjan算法

Tarjan算法是基于对图深度优先搜索的算法。它维护了一个栈，搜索时，把与当前点相连的未访问的节点入栈，回溯时可以判断栈顶到栈中的节点是否为一个强连通分量。

该算法引入了两个很重要的变量：

- dfn[u]是节点u的时间戳，表示点u在DFS搜索树中是第几个被访问的节点。
- low[u]是u或u的子树能够追溯到的最早的栈中节点的时间戳。
- 当dfn[u]=low[u]时，以u为根的搜索子树上，且在栈中所有节点（从栈顶到u的所有节点）是一个强连通分量。

Tarjan算法

按照深度优先搜索算法搜索的次序对图中所有的结点进行搜索，维护每个结点的 dfn 与 low 变量，且让搜索到的结点入栈。每当找到一个强连通分量，就将该强连通分量包含的结点出栈，同时记录结点个数。在搜索过程中，对于结点 u 和与其邻接点 v（v 不是 u 的父节点）考虑 3 种情况：

- v 未被访问过：那么对 v 进行深搜。在回溯到 u 时，用 low[v] 更新 low[u]。（因为存在从 u 到 v 的直接路径，所以如果 v 能够回到的已经在栈中的结点，u 也一定能够回到），即：

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

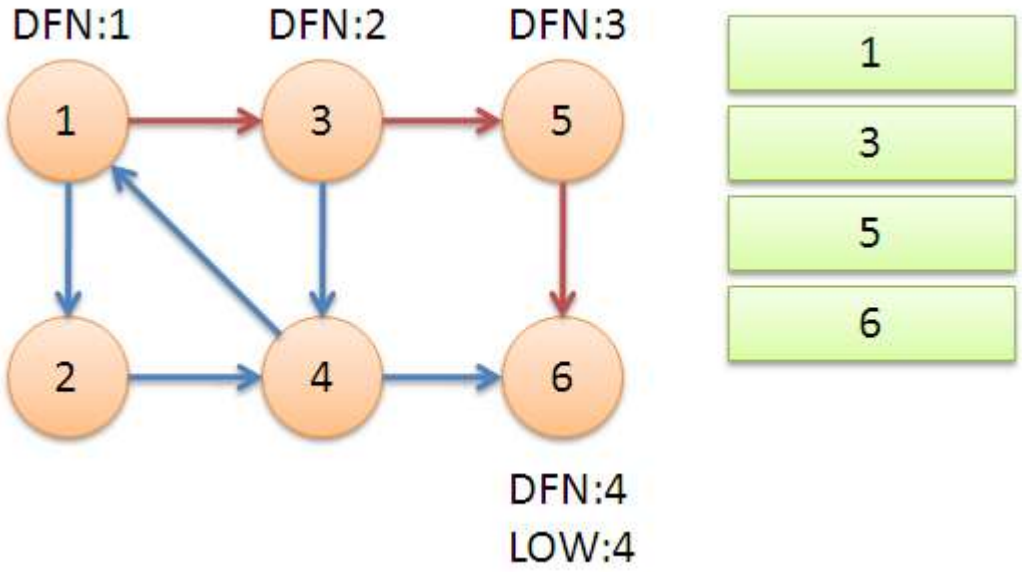
- v 被访问过，且在栈中：说明找到了一个祖先，那么

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

- v 被访问过，且不在栈中：说明 v 已搜索完毕，其所在强连通分量已被处理，所以不用对其做操作。

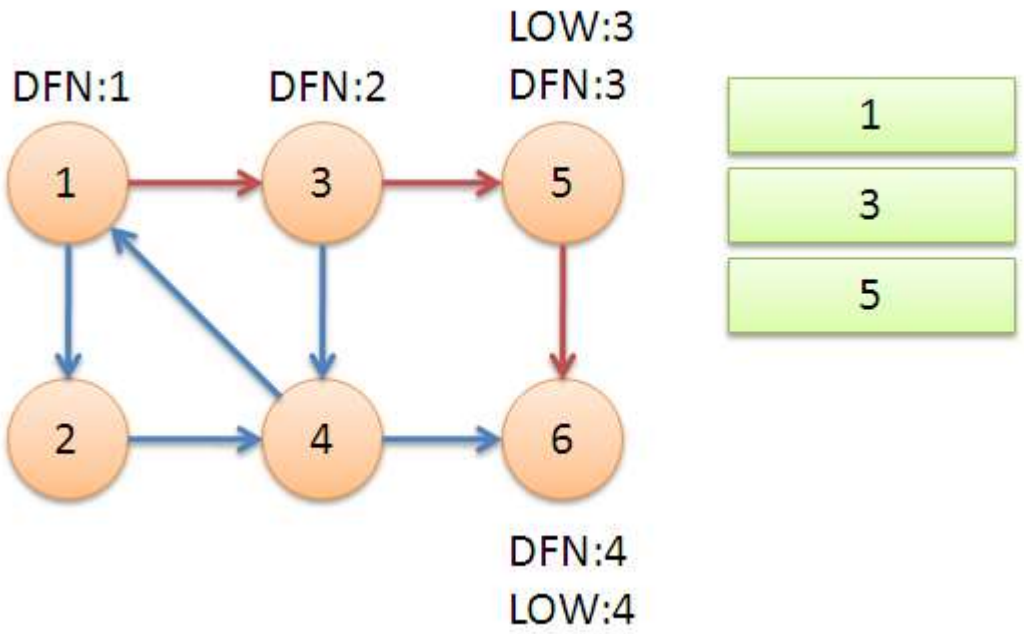
算法演示

从节点 1 开始 DFS，把遍历到的节点加入栈中。搜索到节点u=6时，dfn[6]=low[6]，找到了一个强连通分量。退栈到u=v为止，{6}为一个强连通分量。



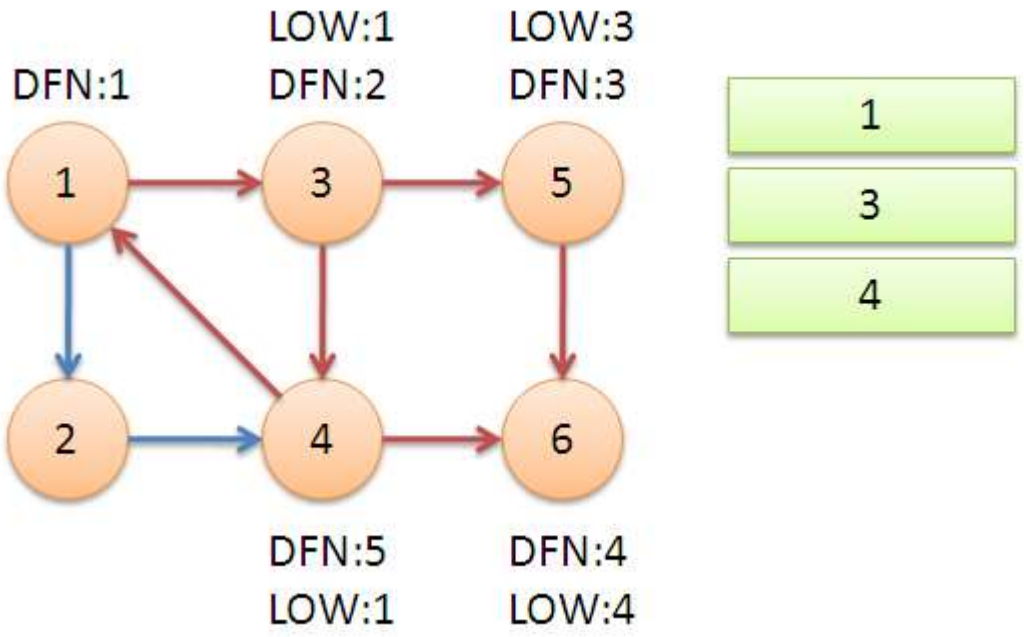
算法演示

返回节点5，发现dfn[5]=low[5]，退栈后{5}为一个强连通分量。



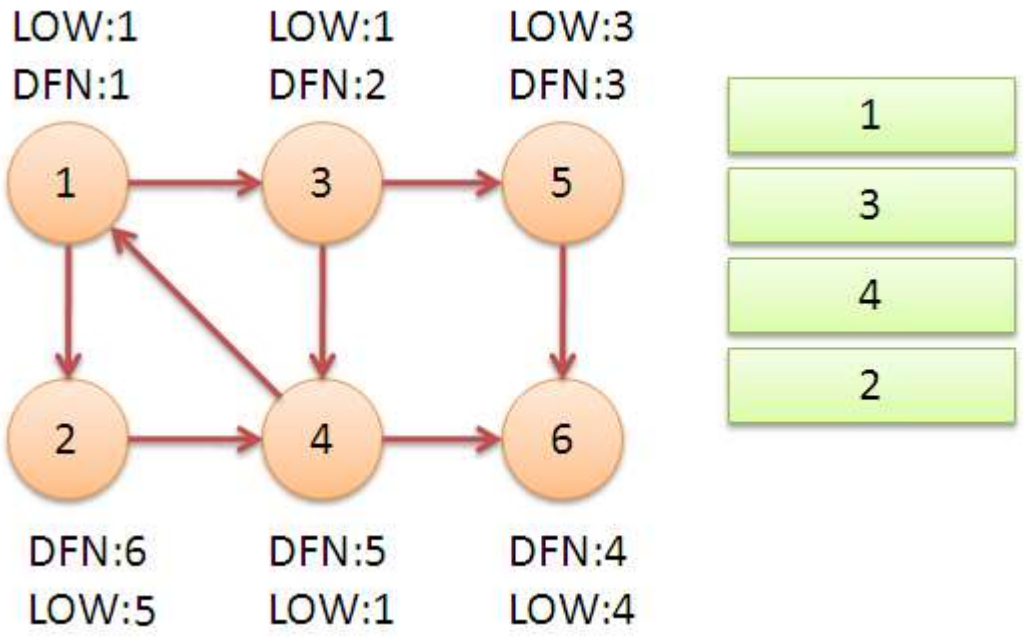
算法演示

返回节点3，继续搜索到节点4，把4加入堆栈。发现节点4向节点1有后向边，节点1还在栈中，所以low[4]=1。节点6已经出栈，(4,6)是横叉边，返回3，(3,4)为树枝边，所以low[3]= low[4]=1。



算法演示

继续回到节点1，最后访问节点2。访问边(2,4)，4还在栈中，所以low[2]=dfn[4]=5。返回1后，发现dfn[1]=low[1]，把栈中节点全部取出，组成一个连通分量{1,3,4,2}。



代码实现

```
void tarjan(int x) {
    dfn[x] = low[x] = ++num;
    stk.push(x);
    v[x] = 1;
    for (int i = head[x]; i; i = next[i]) {
        if (!dfn[to[i]]) { // <x, to[i]> 是树边
            tarjan(to[i]);
            low[x] = min(low[x], low[to[i]]);
        }
        else if (v[to[i]]) { // <x, to[i]> 是后向（返祖）边
            low[x] = min(low[x], dfn[to[i]]);
        }
    }
    if (low[x] == dfn[x]) { // 栈中x以上的点构成一个强连通分量
        int y;
        ++t; // 强连通分量个数加 1
        do {
            y = stk.top();
            stk.pop();
            v[y] = 0; // 标记节点出栈
            belong[y] = t; // 记录当前节点属于哪个强连通分量
        } while (y != x);
    }
}
```

【例1】信息传递

- 建好有向图
- 找强连通分量，同时记录每个强连通分量中节点的个数
- 找节点个数最小的强连通分量

Tarjan缩点

- 有向图的强连通分量的主要作用是为了缩点，缩点之后每一个环成为一个点
- 枚举每一条边，每一条边的两个端点，看是否在同一个强连通分量中，如果在一个强连通中，则忽略，如果不在，则把两个端点所属的连通分量编号作为结点编号进行加边建立新图
- 新图为有向无环图DAG，在DAG上进行一些操作相对来说比较简单

```
//tarjan缩点重新建图
for (auto e : edge) {
    if (belong[e.from] != belong[e.to]) {
        addedge2(e.from, e.to, e.w);
    }
}
```

【例2】POJ1236 Network of Schools

【问题描述】

一些学校联接在一个计算机网络上，学校之间存在软件支援协议，每个学校都有它应支援的学校名单（A学校支援学校B，并不表示B学校一定支援学校A）。当某校获得一个新软件时，无论是直接获得还是通过网络获得，该校都应立即将这个软件通过网络传送给它应支援的学校。因此，一个新软件若想让所有联接在网络上的学校都能使用，只需将其提供给一些学校即可。

1. 最少需要将一个新软件直接提供给多少个学校，才能使软件能够通过网络被传送到所有学校？

2.最少需要添加几条新的支援关系，使得将一个新软件提供给任何一个学校，其他所有学校就都可以通过网络获得该软件。

【分析】

- 求强连通分量，缩点。第一问是零入度点个数；第二问是Max(零入度点个数，零出度点个数)，只有一个SCC时特判。

【例2】POJ1236 Network of Schools

求解答案部分参考代码

```
//vector存图，在graph中，共有cnt个连通分量
int incnt, outcnt = 0; //分别记录入度/出席为0的点的个数
//遍历每条边，不在同一连通分量的，统计入度和出度
for (int i = 1; i <= n; i++) {
    for (auto j : graph[i]) {
        if (belong[i] != belong[j])
            in[belong[j]]++, out[belong[i]]++;
    }
}
for (int i = 1; i <= cnt; i++) {
    if (!in[i]) incnt++;
    if (!out[i]) outcnt++;
}
//后面输出答案，略
```

【例3】Poj2762 Going from u to v or v to u

【题目大意】

判断一个有向图是否单向连通，即对于图中任意两点u和v，若u能到v，或者v能到u，二者满足其一，这两点就连通。

【分析】

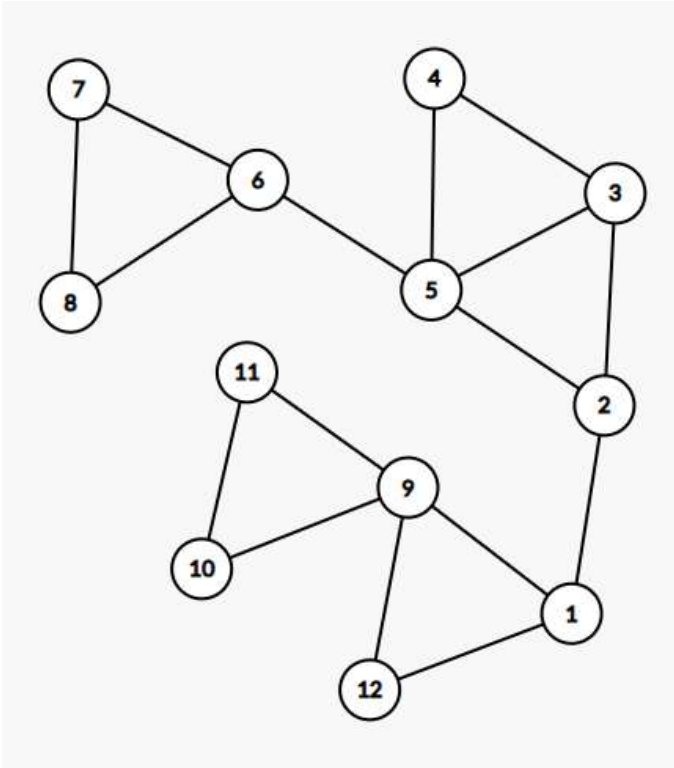
- 首先用tarjan求出所有强连通分量并缩点，只有最后所有缩点能连成一条链时，满足要求。
- 因此寻找一个入度为0的点，用拓扑序列的顺序DP往下找一个最长链，看链上的点的个数是否等于所有缩点的个数即可。
- 第二步也可以用拓扑排序，队列中一直只有一个入度为0的点时满足要求。

无向图上Tarjan算法的应用

一些定义：

1. 割点：若删掉某点后，原连通图分裂为多个互不连通的子图，则称该点为割点。

右图中 1, 2, 5, 6, 9 是割点

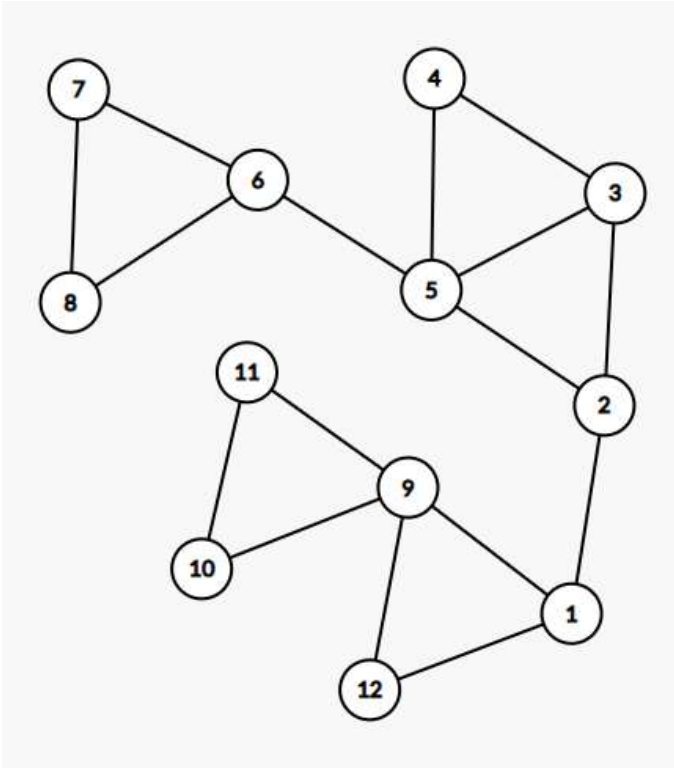


无向图上Tarjan算法的应用

一些定义：

2. 割边(桥)：删掉它之后，图必然会分裂为两个或两个以上的互不连通的子图。

边<1, 2>, <5, 6>是桥



无向图上Tarjan算法的应用

一些定义：

3. 边双连通：在一张连通的无向图中，对于两个点 u 和 v，如果无论删去哪条边（只能删去一条）都不能使它们不连通，我们就说 u 和 v 边双连通。

4. 边双连通分量：不存在割边的极大双连通子图（再加入一个或一些点及对应的边后，图不连通或存在割边）

性质

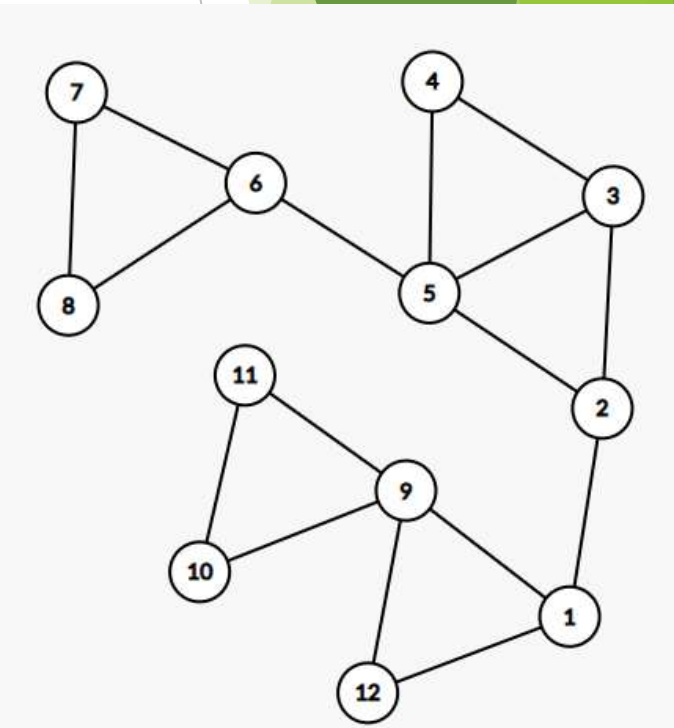
- 割边不属于任意边双，而其它非割边的边都属于且仅属于一个边双。
- 一个边双中的任意两个点，它们之间都有至少两条边不重复的路径。

右图的边双连通分量：

（1, 9, 10, 11, 12）

（2, 3, 4, 5）

（6, 7, 8）



无向图上Tarjan算法的应用

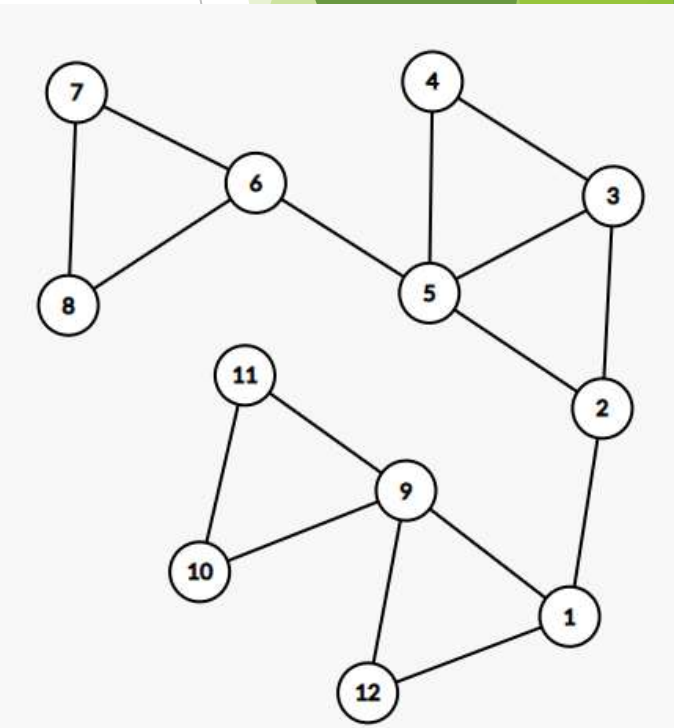
一些定义：

5. 点双连通：在一张连通的无向图中，对于两个点 u 和 v，如果无论删去哪个点（只能删去一个，且不能删 u 和 v 自己）都不能使它们不连通，我们就说 u 和 v 点双连通。

6. 点双连通分量：不存在割点的极大双连通子图（再加入一个或一些点及对应的边后，图不连通或存在割点）

性质

- 除了一种比较特殊的点双，其他的点双都满足：任意两点间都存在至少两条点不重复路径。
- 图中任意一个割点都在至少两个点双中。
- 任意一个不是割点的点都只存在于一个点双中。



右图的点双连通分量：

(1, 9, 12) (9, 10, 11) (1, 2) (2, 3, 4, 5) (6, 7, 8) (5, 6)

Tarjan求割点

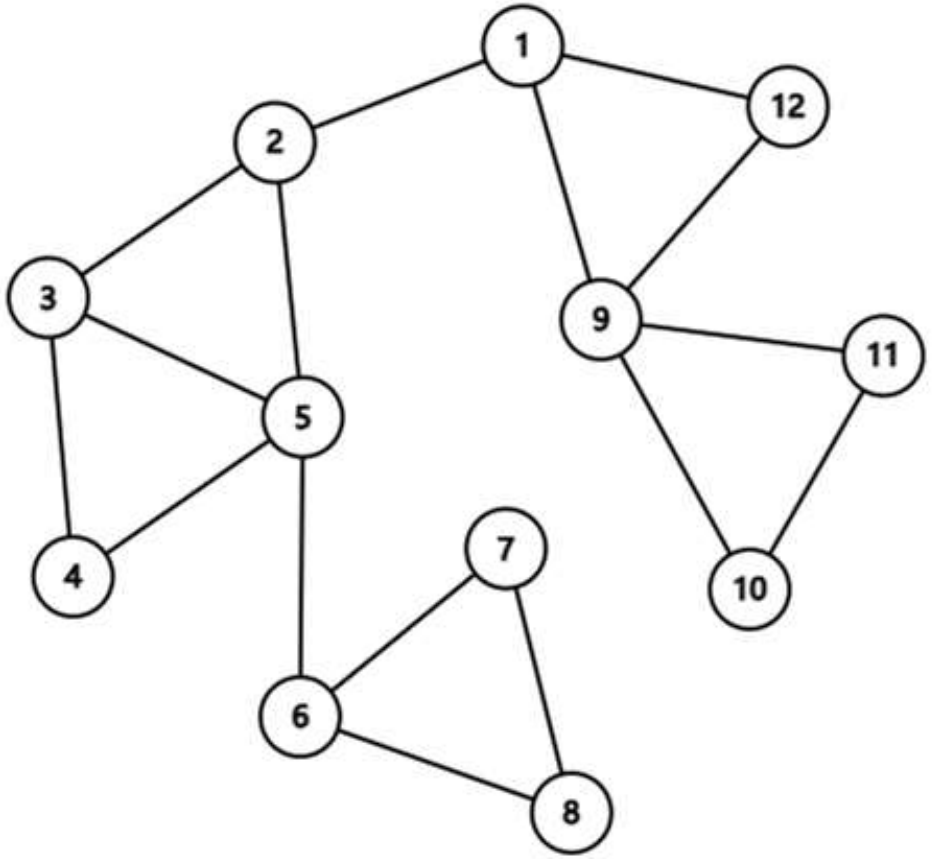
Tarjan算法中，我们得到了dfn和low两个数组，

low[u]=min(low[u], dfn[v])——(u,v)为后向边；

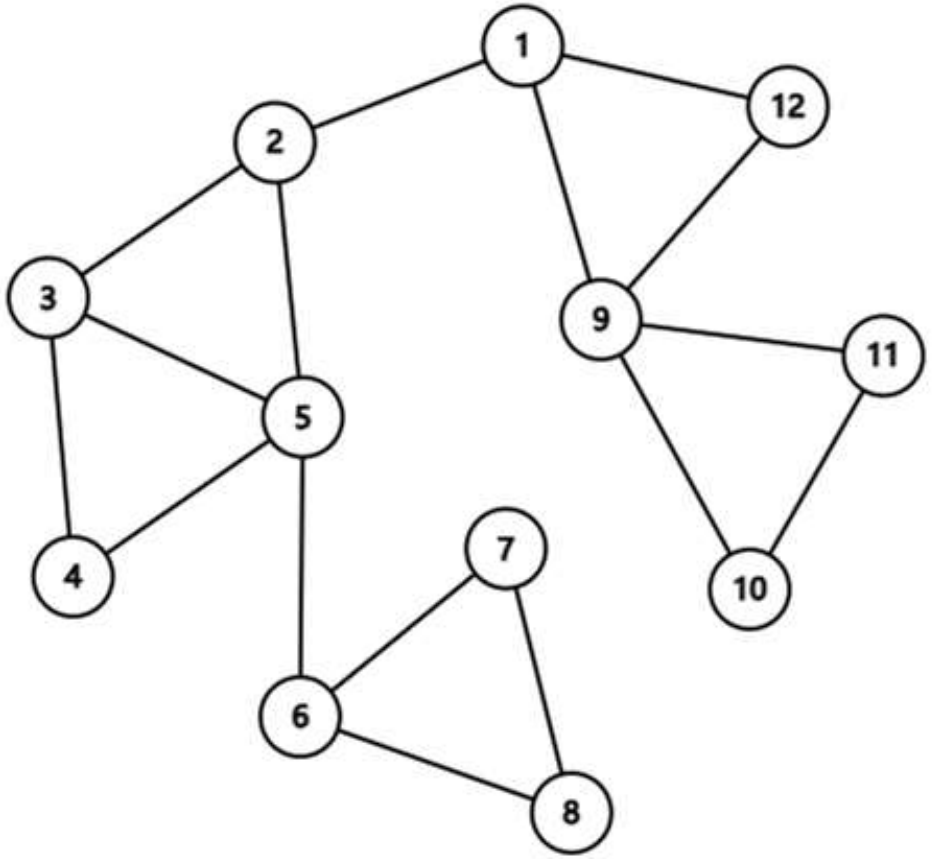
low[u]=min(low[u], low[v])——(u,v)为树枝边，v为u的子树；

若low[v]>=dfn[u]，则u为割点，节点v的子孙和节点u形成一个块。因为这说明v的子孙不能够通过其他边到达u的祖先，这样去掉u之后，图必然分裂为两个子图。这样我们处理点u时，首先递归u的子节点v，然后从v回溯至u后，如果发现上述不等式成立，则找到了一个割点u，并且u和v的子树构成一个块。

Tarjan求割点



Tarjan求割点



Tarjan求割点
代码实现

```
void tarjan(int now) {
    dfn[now] = low[now] = ++num;
    int son = 0;
    for (int i = head[now]; i; i = nxt[i]) {
        if (!dfn[to[i]]) {
            son++;
            tarjan(to[i]);
            low[now] = min(low[now], low[to[i]]);
            if (dfn[now] <= low[to[i]]) {
                // 如果是起点，至少要有两棵子树才是割点
                // root为dfs搜索树的起点
                if (now != root || son > 1) cut[now] = true;
            }
        } else {
            low[now] = min(low[now], dfn[to[i]]);
        }
    }
}
```

Tarjan求割点

- 题库例题D 备用交换机

Tarjan求割边（桥）

由于无向图存边是双向存，从 u 递归到 v 时，在找 v 的邻接点时，必然会找回 u，此时我们可能有多种处理方法

方法一：

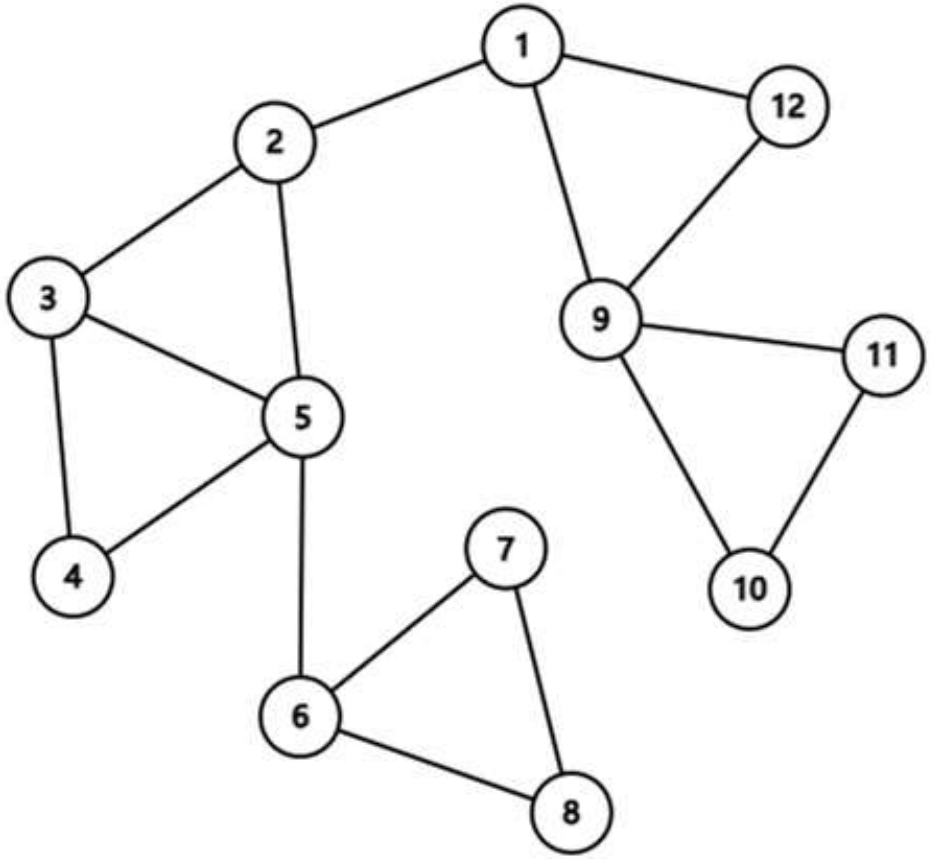
递归传参加一个父亲的编号，判断若low[v]>dfn[u]，则(u,v)为割边。

方法二：

我们记录每条边的标号（一条无向边拆成的两条有向边标号相同），记录每个点的父亲到它的边的标号，如果边(u,v)是v的父亲边，就不能用dfn[u]更新low[v]。

这样如果遍历完v的所有子节点后，发现low[v]=dfn[v]，说明u的父亲边(u,v)为割边。

Tarjan求割点



Tarjan求割边（桥）
方法一：判断dfn[u] < low[v]

```
/*
边从edge数组下标0的位置开始存储
每条无向对称存放的下标分别为(0,1) (2,3) (4,5)...
0^1=1 1^1=0 2^1=3 3^1=2 ...
可以通边的下标做异或，快速判断同一条边
*/
void tarjan(int now, int fa) {
    dfn[now] = low[now] = ++num;
    bool first = true; //标记是否第一次找到父结点
    for (int i = head[now]; i != -1; i = edge[i].nxt) {
        int to = edge[i].to;
        //第一次回到父结点，可以认为是无向图的反向边
        if (to == fa && first) {
            first = false; //记着修改标记
            continue;
        }
        if (!dfn[to]) {
            tarjan(to, now);
            low[now] = min(low[now], low[to]);
            if (dfn[now] < low[to]) {
                cnt++; //找到一条割边
                bridge[i] = bridge[i ^ 1] = true; //边i和i^1是割边
            }
        } else {
            low[now] = min(low[now], dfn[to]);
        }
    }
}
```

Tarjan求割边（桥）
方法二：判断low[v]==dfn[v]

```
// 此方法为遍历完某个子节点v之后
// 判断子节点与父节点之间的边是否为割边
void tarjan(int x) {
    vis[x] = 1;
    dfn[x] = low[x] = ++num;
    for (int i = head[x]; i; i = next[i]) {
        // 如果子节点没有被访问过，直接递归访问，此时经过一条父亲边
        // 如果从子节点中往回找到了父亲，此时判断一下使用的边的编号
        // 如果等于从父节点过来经过的边的编号，此时不能走
        if (!vis[to[i]]) {
            p[to[i]] = edge[i]; //记录父亲边的编号
            tarjan(to[i]);
            low[x] = min(low[x], low[to[i]]);
        } else if (p[x] != edge[i]) {//不是父亲边才更新
            low[x] = min(low[x], dfn[to[i]]);
        }
    }
    if (p[x] && low[x] == dfn[x]) f[p[x]] = 1; //是割边
}
```

Tarjan求割边（桥）
方法三：前面两种综合一下

```
// 所有边在数组中从下标为0的位置开始存储
// 一条无向边，假设存在0和1的位置，则
// 0^1=1    1^1=0
// 可以用此方法判断是否为同一条边
void tarjan(int now) {
    vis[now] = 1;
    dfn[now] = low[now] = ++num;
    for (int i = head[now]; i != -1; i = nxt[i]) {
        if (i == p[now] ^ 1) continue;
        if (!dfn[to[i]]) {
            p[to[i]] = i; // 记录边的编号
            tarjan(to[i]);
            low[now] = min(low[now], low[to[i]]);
            if (dfn[now] > low[to[i]]) { // (now, to[i])是割边
                cnt++;
                cut[i] = cut[i^1] = 1; // 记录边是割边
            }
        } else {
            low[now] = min(low[now], dfn[to[i]]);
        }
    }
}
```

Tarjan求割边（桥）

- 题库例题G 旅游航道

Tarjan求双连通分量

- 若一个无向图中的去掉任意一个节点（一条边）都不会改变此图的连通性，即不存在割点（桥），则称作点（边）双连通图。

- 一个无向图中的每一个极大点（边）双连通子图称作此无向图的点（边）双连通分量。求双连通分量可用Tarjan算法。

Tarjan求双连通分量
1.点双连通分量

- 对于**点双连通分支**，实际上在**求割点**的过程中就能顺便把每个点双连通分支求出。建立一个栈，存储当前双连通分支，访问每个点时把这个点入栈。
- 如果某时满足dfn(u)<=low(v)，说明u是一个割点，那么把点从栈顶一个个取出，直到遇到了点u（u不取出），取出的这些点和u组成一个点双连通分支。
- 割点可以属于多个点双连通分支，其余点和每条边只属于且属于一个点双连通分支。

Tarjan求双连通分量
1.点双连通分量

- 找割点，栈中割点上面的点构成一个点双
- 注意重边的处理

```
void tarjan(int now, int fa) {
    vis[now] = true;
    dfn[now] = low[now] = ++dfs_clock;
    stk[++top] = now;

    bool first = true;
    int son = 0;
    for (int i = head[now]; i; i = edge[i].next) {
        int to = edge[i].to;
        // 第一次回父节点，pass
        if (first && to == fa) {
            first = false;
            continue;
        }
        if (!dfn[to]) {
            son++;
            tarjan(to, now);
            low[now] = min(low[now], low[to]);

            // now是割点，栈中now上面的点构成一个点双
            if (dfn[now] <= low[to]) {
                cut[now] = true; // 特判根节点
                cnt++;
                // now不能直接出栈，它有可能属于多个点双
                dslt[cnt].push_back(now);
                int x;
                do {
                    x = stk[top--];
                    dslt[cnt].push_back(x);
                } while (x != to);
            }
        } else {
            low[now] = min(low[now], dfn[to]);
        }
    }
    if (fa == -1 && son == 1) cut[now] = false;
}
```

Tarjan求双连通分量
2.边双连通分量

- 对于**边双连通分支**，求法更为简单。只需在**求出所有的桥**以后，把桥边删除，原图变成了多个连通块，则每个连通块就是一个边双连通分支。
- 桥不属于任何一个边双连通分支，其余的边和每个顶点都属于且只属于一个边双连通分支。

Tarjan求双连通分量
2.边双连通分量

- 找割点，栈中now及上面的点构成一个边双

- 由于要记录桥的编号，因此处理重边问题就直接用边的编号来处理了，当然也可以用上个代码中的first判断

```
// 第二个参数为父边的编号
void tarjan(int now, int eid) {
    vis[now] = true;
    dfn[now] = low[now] = ++dfs_clock;
    stk[++top] = now;

    for (int i = head[now]; i; i = edge[i].next) {
        int to = edge[i].to;
        int id = edge[i].id;
        if (id == eid ^ 1) continue; // 父节点过来的反向边

        if (!dfn[to]) { // to 未访问过
            tarjan(to, id);
            low[now] = min(low[now], low[to]);
        } else {
            low[now] = min(low[now], dfn[to]);
        }
    }
    // 所有孩子遍历完再统计
    // 相等：此时父边eid是桥
    // now及以上节点构成边双
    if (dfn[now] == low[now]) {
        cut[eid] = true;
        cnt++;
        int x;
        do {
            x = stk[top--];
            bslt[cnt].push_back(x);
        } while (x != now);
    }
}
```

边双联通分量的构造

- **一个有桥的连通图，如何把它通过加边变成边双连通图？**方法为首先求出所有的桥，然后删除这些桥边，剩下的每个连通块都是一个双连通子图。把每个双连通子图收缩为一个顶点，再把桥边加回来，最后的这个图一定是一棵树，边连通度为1。
- 统计出树中度为1的节点的个数，即为叶节点的个数，记为leaf。则至少在树上添加(leaf+1)/2条边，就能使树达到边双连通。
- 具体方法为，首先在两个最近公共祖先最远的两个叶节点之间连接一条边，这样可以把这两个点到祖先的路径上所有点收缩到一起，因为一个形成的环一定是双连通的。然后再找两个最近公共祖先最远的两个叶节点，这样一对一对找完，恰好是(leaf+1)/2次，把所有点收缩到了一起。

题库H分离的路径

Poj2942 Knights of the Round Table

- 题目大意：给定n个骑士和他们之间的仇恨关系，规定召开圆桌会议时，两个有仇恨的骑士不能坐在相邻位置，且召开一次圆桌会议的骑士人数必须为奇数，求有多少骑士永远不可能参加某一次圆桌会议。
- 模型转化：首先建立原图的补图，即没有仇恨的骑士间连边。容易想到，**几个骑士可以召开圆桌会议的条件是它们构成一个奇环**，那么它们就可以按照这个奇环坐到桌子旁边开会。那么问题转化为：**求有多少个骑士不包含在任何奇环内**。

Poj2942 Knights of the Round Table

- 引理：若某个点双连通分量中有一个奇环，则在该点双联通分量中，奇环外的点一定也在某一个奇环内。
- 证明：如果双连通分量内有一个奇环，由点双连通分量性质，奇环外的某个点A一定可以通过某些边与奇环上两个不相同的点B、C相连。假设A连到B、C的过程中一共途径了K个点，并且点B、C一定把奇环分为了两部分，一半有奇数个点，一半有偶数个点。假如k为偶数，则这k个点与奇数个点的一半可以构成一个奇环；假如k为奇数，k与偶数个点的一半构成一个奇环。证毕。
- 所以我们用Tarjan算法找出所有点双联通分量，然后判定每个点双联通分量是不是二分图，就可以知道它有没有奇环。

最近公共祖先LCA
倍增法

- f[i][j]表示节点i的2^j次祖先。
- f[i][j]=f[f[i][j-1]][j-1]
- 可以通过一次bfs求出f数组。
- 询问lca(x,y)时，假设x的深度比y大，否则交换x、y。
- 把x调整到与y同一深度，然后x、y再同时向上走直到汇合。
- 这是一个在线算法，时间复杂度为O(NlogN)-O(logN)。

倍增法求LCA——代码实现

```
void bfs()
{
 q.push(1); v[1]=1; d[1]=1;
 while(q.size())
 {
      x=q.front(); q.pop();
      for(i=head[x];i;i=next[i])
           if(!v[y=ver[i]])
           {
                q.push(y); v[y]=1;
                d[y]=d[x]+1;
                f[y][0]=x;
                for(j=1;j<20;j++) f[y][j]=f[f[y][j-1]][j-1];
           }
 }
}
```

倍增法求LCA——代码实现

```
int lca(int x,int y)
{
 if(d[x]>d[y]) swap(x,y);
 for(int i=t;i>=0;i--)
       if(d[f[y][i]]>=d[x]) y=f[y][i];
 if(x==y) return x;
 for(int i=t;i>=0;i--)
       if(f[x][i]!=f[y][i]) x=f[x][i],y=f[y][i];
 return f[x][0];
}
```

最近公共祖先LCA
欧拉序+ST算法

- 欧拉序：对树进行一次深度优先搜索，每当经过一个点时，就把它的时间戳记录下来，这样形成的序列被称为这棵树的欧拉序。
- 树上两个点的最近公共祖先，就是欧拉序中这两个点第一次出现的位置之间时间戳最小的节点。
- 因此可以用RMQ问题中的ST算法来维护欧拉序。
- 这也是一个在线算法，时间复杂度为O(NlogN)-O(1)。

最近公共祖先LCA
Tarjan算法

- 我们先读入所有的询问并对这些询问构建一个邻接表。
- 在遍历到u时，先tarjan遍历完u的子树，则u和u的子树中的节点的最近公共祖先就是u，并且u和【u的兄弟节点及其子树】的最近公共祖先就是u的父亲。
- 用一个color数组，正在访问的节点标记为1，未访问的标记为0，已经访问到的即在【u的子树中的】及【u的已访问的兄弟节点及其子树中的】标记为2。
- 再维护一个并查集，访问完节点u的一个子树时，就把这个子树的根节点的fa改为u。访问完u的所有子树后，考虑所有与u相关的询问lca(u,v)，那么lca(u,v)就是v所在并查集的根。
- 这是一个离线算法，时间复杂度为O(Nα(N))，约为O(N)。

Tarjan算法求LCA——代码实现

```
void tarjan(int x)
{
 fa[x]=x,color[x]=1;
 int i,y;
 for(i=head[x];i;i=next[i])
      if(color[y=ver[i]]==0)
      {
           tarjan(y);
           fa[y]=x;
      }
 for(i=headquery[x];i;i=nextquery[i])
      if(color[y=query[i]]==2) ans[i]=get(y);
 color[x]=2;
}
```

2-SAT问题

- 2-SAT问题的基本模型：给定n对数，每一对中必须且仅能取一个数，某些数i、j之间有矛盾不能同时被取，求其可行性以及一种方案。
- 1.构图(若取i后必须取j则在i和j之间连边)；
- 2.用Tarjan求强连通分量；
- 3.若n对数中的某一对数在同一个强连通分量里，则无解，否则一定有解；
- 4.如果还需要输出一组解，可以自底向上拓扑排序，每次找到能够取出的零出度点i，标记与i同一对的点i_及i_的前驱结点为不能取。
- 该算法的时间复杂度为O(N+M)。例题：Poj3207。