Tarjan 算法

李淳风

长郡中学

2024 年 9 月 29 日

无向图的割点与桥

给定无向连通图 $G = (V, E)$:
若对于 $x \in V$，从图中删去节点 x 以及所有与 x 关联的边之后，G 分裂成两个或两个以上不相连的子图，则称 x 为 G 的割点。
若对于 $e \in E$，从图中删去边 e 之后，G 分裂成两个不相连的子图，则称 e 为 G 的桥或割边。
一般无向图的割点与桥就是它的各个连通块的割点和桥。
接下来我们要学习的是由著名计算机科学家 Robert Tarjan 的名字命名的 Tarjan 算法。它能够在线性时间复杂度内求出无向图的割点和桥，进一步可以求出无向图的双连通分量。在有向图方面，Tarjan 算法能够求出有向图的强连通分量、必经点和必经边。
Tarjan 算法基于无向图的深度优先遍历。在之前的学习中，我们已经了解了树和图的深度优先遍历，冰引入了“时间戳”的概念。

无向图的割点和桥

时间戳

在图的深度优先遍历过程中，按照每个节点第一次被访问的时间顺序，依次给予 N 个节点 $1 \sim N$ 的整数标记，该标记就被称为时间戳，记为 $dfn[x]$。

搜索树

在无向连通图中任选一个节点出发进行深度优先遍历，每个点只访问依次。所有发生递归的边 (x, y)（换言之，从 x 到 y 是对 y 的第一次访问）构成一棵树，我们把它称为“无向连通图的搜索树”。

追溯值

除了时间戳之外，Tarjan 算法还引入了一个“追溯值” $low[x]$。设 $subtree(x)$ 表示搜索树中以 x 为根的子树，$low[x]$ 定义为以下节点的时间戳的最小值：

- $subtree(x)$ 中的节点
- 通过 1 条不在搜索树上的边，能够到达 $subtree(x)$ 的节点。

换句话来说，就是从 $subtree(x)$ 中任意节点出发，经过至多一条不在树上的边，能够到达的节点中 dfn 的最小值。

无向图的割点和桥

割边判定法则

无向边 (x, y) 是桥，当且仅当搜索树上存在 x 的一个子节点 y，满足：

$$dfn[x] < low[y]$$

根据定义，这说明从 $subtree(y)$ 出发，在不经过 (x, y) 的前提下，不管走哪条边，都无法到达 x 或比 x 更早被访问的节点。若把 (x, y) 删除，则 $subtree(y)$ 就好似形成了一个封闭的环境，与节点 x 以及更早被访问的节点没有边相连，图断开成了两部分，因此 (x, y) 是割边。

反之，若不存在这样的子节点 y 使得 $dfn[x] < low[y]$，则说明每个 $subtree(y)$ 都能绕行其它边到达 x 或比 x 更早被访问的节点，(x, y) 自然就不是割边。容易发现，桥一定是搜索树中的边，并且一个简单环中的边一定都不是桥。

需要注意的是，我们不能在 DFS 的时候仅仅记录每个节点的父节点。当 (x, fa) 之间有多条边时，这些边都不是桥，但程序有可能把它们判断为同一条边，进而错讲它判断为桥。因此，我们可以在 DFS 时记录邻接表中边的编号，而我们在存储时是把无向边对应的两条边成对存储在下标“2 和 3”“4 和 5”“6 和 7”处的。如果使用 $vector$ 存储图的话，特殊判断重边也是种方法。

割边判定法则

```
void tarjan(int x,int in_edge){
    dfn[x]=low[x]=++num;//时间戳
    for(int i=head[x];i;i=Next[i]){
        int y=ver[i];//找到儿子节点
        if(!dfn[y]){
            tarjan(y,i);//第一次访问 y
            low[x]=min(low[x],low[y]);//更新 low
            if(low[y]>dfn[x])//判断是否是桥
                bridge[i]=bridge[i^1]=true;
        }
        else if(i!=(in_edge^1))//不是进入 x 的边，更新 low
            low[x]=min(low[x],dfn[y]);
    }
}
```

割点判定法则

若 x 不是搜索树的根节点（深度优先遍历的起点），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：

$$dfn[x] \leq low[y]$$

特别地，若 x 是搜索树的根节点，则 x 是割点当且仅当 x 在搜索树中拥有不少于两个子节点。
证明方法和割边类似，这里就不再赘述。因为割点判定法则是小于等于号，所以在求割点时，不必考虑父节点和重边的问题，从 x 出发能访问到的所有点的时间戳都能用来更新 $low[x]$。

割点判定法则

```
void tarjan(int x){
    dfs[x]=low[x]=++num;
    int flag=0;
    for(int i=head[x];i;i=Next[i]){
        int y=ver[i];
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x]){
                flag++;
                if(x!=root || flag>1) cut[x]=true;
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
}
```

例题

BLO-Blockade

B 城有 n 个城镇（从 1 到 n 标号）和 m 条双向道路。
每条道路连接两个不同的城镇，没有重复的道路，所有城镇连通。
把城镇看作节点，把道路看作边，整个城市构成了一个无向图。
请你对于每个节点 i 求出，把与节点 i 关联的所有边去掉以后（不去掉节点 i 本身），无向图有多少个有序点对 (x, y)，满足 x 和 y 不连通。
$n \leq 100000, m \leq 500000$。

例题

BLO-Blockade

B 城有 n 个城镇（从 1 到 n 标号）和 m 条双向道路。
每条道路连接两个不同的城镇，没有重复的道路，所有城镇连通。
把城镇看作节点，把道路看作边，整个城市构成了一个无向图。
请你对于每个节点 i 求出，把与节点 i 关联的所有边去掉以后（不去掉节点 i 本身），无向图有多少个有序点对 (x, y)，满足 x 和 y 不连通。
$n \leq 100000, m \leq 500000$。

根据割点的定义，若节点 i 不是割点，那么把所有与 i 关联的边去掉之后，只有 i 与其它 $n-1$ 个节点不连通，其余 $n-1$ 个节点之间还是连通的。注意要求的是有序点对，因此对答案的贡献为 $2(n-1)$。

BLO-Blockade

若节点 i 是割点，则把节点 i 关联的所有边去掉之后，图会分成若干个连通块，并且对于 i 的儿子 j，如果 $dfn[i] \leq low[j]$，则以 j 为根的子树会与其它节点分割开来。设这些满足条件的儿子节点为 $s_1, s_2, \cdots, s_t$，则图会分成至多 $t+2$ 个连通块。其中：

- 节点 i 单独一个连通块；
- 每个 $s_k (1 \leq k \leq t)$ 及其在搜索树中的子树构成一个连通块；
- 还可能有一个连通块，由除了上述节点之外剩下的节点构成。

我们不妨在 Tarjan 算法执行深度优先遍历的时候，顺带求出每棵子树的大小，即可知道每个连通块的大小，就可以对于每个 i 求出它对应的有序对的数量了。

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被称为“点双连通分量”，简记为“v-DCC”。无向图的极大边双连通子图被称为“边双连通分量”，简记为“e-DCC”。二者统称为“双连通分量”，简记为“DCC”。
在上面中，我们称一个双连通子图 G' “极大”，是指不存在包含 G' 的更大的子图 G''，满足 G'' 也是双连通子图。

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被称为“点双连通分量”，简记为“v-DCC”。无向图的极大边双连通子图被称为“边双连通分量”，简记为“e-DCC”。二者统称为“双连通分量”，简记为“DCC”。
在上面中，我们称一个双连通子图 G' “极大”，是指不存在包含 G' 的更大的子图 G''，满足 G'' 也是双连通子图。

一张图是点双连通图，当且仅当满足下列两个条件之一：

- 图的顶点数不超过 2。
- 图中任意两点都同时被包含在至少一个简单环中。其中“简单环”是指不自交的环，也就是我们通常画出的环。

一张图是边双连通分量，当且仅当任意一条边都被包含在至少一个简单环中。

边双连通分量

边双连通分量的计算非常容易。只需求出无向图中所有的桥，把桥都删除后，无向图会分成若干个连通块，每个连通块就是一个“边双连通分量”。
在具体的程序实现中，一般先用 Tarjan 算法标记出所有的桥边。然后，再对整个无向图执行深度优先遍历（遍历的过程中不访问桥边），划分出每个连通块即可。

边双连通分量

边双连通分量的计算非常容易。只需求出无向图中所有的桥，把桥都删除后，无向图会分成若干个连通块，每个连通块就是一个“边双连通分量”。
在具体的程序实现中，一般先用 Tarjan 算法标记出所有的桥边。然后，再对整个无向图执行深度优先遍历（遍历的过程中不访问桥边），划分出每个连通块即可。
把每个 e-DCC 看作一个节点，我们记录每个点 x 对应的 e-Dcc 编号 $c[x]$，把桥边 (x, y) 看作连接编号为 $c[x]$ 和 $c[y]$ 的 e-DCC 对应节点的无向边，会产生一棵树（若原来的无向图不连通，则产生森林）。这种把 $e-DCC$ 收缩为一个节点的方法就称为“缩点”。对于缩点之后的新图，我们新开一个邻接表进行存储即可。

点双连通分量

点双连通分量是一个极其容易误解的概念，它与“删除割点后图中剩余的连通块”是不一样的。
若某个节点为孤立点，则它自己单独构成一个 v-DCC。除孤立点之外，点双连通分量的大小至少为 2。根据 v-DCC 中的“极大”性，虽然桥不属于任何一个边双连通分量，但是割点可能同时属于多个点双连通分量。
为了求出“点双连通分量”，需要在 Tarjan 算法的过程中维护一个栈，并按照如下方法维护栈中的元素：

- 当一个节点第一次被访问时，把该节点入栈。
- 当割点判定法则中的条件 $dfn[x] \leq low[y]$ 成立时，无论 x 是否为根，都要：
  - 从栈顶不断弹出节点，直至节点 y 被弹出；
  - 刚才弹出的所有节点与节点 x 一起组成一个 v-DCC。

```
void tarjan(int x){
    dfn[x]=low[x]=++num;
    stack[++top]=x;//x 入栈
    if(x==root && head[x]==0){//判断孤立点
        dcc[++cnt].push_back(x);
        return;
    }
    int flag=0,z;
    for(int i=head[i];i;i=Next[i]){
        int y=ver[i];//儿子节点
        if(!dfn[y]){
            tarjan(y);
            low[x]=min(low[x],low[y]);
            if(low[y]>=dfn[x]){
                flag++,cnt++;
                if(x!=root || flag>1) cut[x]=true;
                while(1){//x 是该双连通分量中深度最小的点
                    z=stack[top--];
                    dcc[cnt].push_back(z);
                    if(z==y) break;
                }
                dcc[cnt].push_back(x);//x 可能出现在多个 v-DCC 中
            }
        }
        else low[x]=min(low[x],dfn[y]);
    }
}
```

点双连通分量

v-DCC 的缩点比 e-DCC 要复杂一些——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，我们建立一张包含 $p+t$ 个节点的信徒，把每个 v-DCC 和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。容易发现，这棵新图其实是一棵树（或森林）。

```
num=cnt;//cnt 是 v-DCC 数量
for(int i=1;i<=n;i++)//割点新建一个点
    if(cut[i]) new_id[i]=++num;
for(int i=1;i<=cnt;i++)
    for(int j=0;j<dcc[i].size();j++){
        int x=dcc[i][j];
        if(cut[x]){//割点往 v-DCC 连边
            add_c(i,new_id[x]);
            add_c(new_id[x],i);
        }
        else c[x]=i;//除割点外，其他点仅属于一个 v-DCC
    }
```

例题

NetWork

给定一张 N 个点 M 条边的无向连通图，然后执行 Q 次操作，每次向图中添加一条边，并且询问当前无向图中“桥”的数量。
$N \leq 10^5, M \leq 2 * 10^5, Q \leq 1000$。

例题

NetWork

给定一张 N 个点 M 条边的无向连通图，然后执行 Q 次操作，每次向图中添加一条边，并且询问当前无向图中“桥”的数量。
$N \leq 10^5, M \leq 2 * 10^5, Q \leq 1000$。

先用 Tarjan 算法求出无向图中所有的“边双连通分量”(e-DCC)，并对每个 e-DCC 缩点，得到一棵树，树上的每个节点都是原图中的一个 e-DCC。设 $c[x]$ 表示节点 x 所在的 e-DCC 的编号。最初，“桥”的总数就是缩点之后树的边数。
一次考虑每个添加边 (x, y) 的操作。若 $c[x] = c[y]$，则说明二者属于同一个 e-DCC，加入这条边后对“桥”的数量没有影响；否则在缩点得到的树上 $c[x]$ 与 $c[y]$ 的路径上，每条边都不再是“桥”，因为它们已经处于一个环内了。
因此我们可以求出 $p = lca(c[x], c[y])$，把 $c[x], c[y]$ 到 p 节点上的所有边都打上标记。若有 cnt 条边新获得了标记，“桥”的总数就会减少 cnt。
为了保证复杂度，我们可以使用并查集，每次对于任意一个节点 x，快速找到从 x 到根节点的路径上第一条未被标记的边在哪。

例题

Knights of the Round Table

有 n 个骑士经常举行圆桌会议，商讨大事。每次圆桌会议至少有 3 个骑士参加，且相互憎恨的骑士不能坐在圆桌的相邻位置。如果发生意见分歧，则需要举手表决，因此参加会议的骑士数目必须是大于 1 的奇数，以防止赞同和反对票一样多。知道骑士之间相互憎恨的关系后，请你帮忙统计有多少骑士参加不了任意一个会议。

$1 \leq n \leq 10^3, 1 \leq m \leq 10^6$。

例题

Knights of the Round Table

有 n 个骑士经常举行圆桌会议，商讨大事。每次圆桌会议至少有 3 个骑士参加，且相互憎恨的骑士不能坐在圆桌的相邻位置。如果发生意见分歧，则需要举手表决，因此参加会议的骑士数目必须是大于 1 的奇数，以防止赞同和反对票一样多。知道骑士之间相互憎恨的关系后，请你帮忙统计有多少骑士参加不了任意一个会议。
$1 \leq n \leq 10^3, 1 \leq m \leq 10^6$。

我们可以建出原图的补图——n 个节点代表 n 个骑士，若两名骑士没有憎恨关系，则在二者之间连一条无向边。
根据题意，若干名骑士可以召开圆桌会议的条件是：他们对应的节点组成一个长度为奇数的简单环。因此，本题就是求有多少个点不被任何奇环包含。

Knights of the Round Table

首先我们可以发现，若两个骑士属于两个不同的“点双连通分量”，则他们不可能一起出席会议。因为如果这两个节点在同一个奇环上，那么对于这两个“点双联通分量”上的每一个点，删除它都不会导致两个骑士对应的节点不连通，所以两个骑士应该属于同一个“点双连通分量”，与假设矛盾。
其次，若某个“点双连通分量”中存在奇环，则这个“点双连通分量”中的所有点都能被至少一个奇环包含。因为奇环上任意两个不同的点都可以把奇环分割为两段为长度分别为奇数、偶数的路径，总能与另外一段不过这个环的路径拼出奇环。
综上所述，我们用 Tarjan 算法求出“补图”中所有的 v-DCC，并判定每个 v-DCC 中是否存在奇环即可。若存在奇环，则该 v-DCC 中的所有骑士都能参加会议。
奇环可以用深度优先遍历的染色法进行判定。一张无向图没有奇环，等价于它是一张二分图，我们尝试给每个点一个颜色，使得任意一条边两段的节点颜色不同。如果使用两种颜色就可以完成染色，则说明是一张二分图，不存在奇环。否则说明存在奇环。

有向图上的 Tarjan

给定有向图 $G=(V,E)$，若存在 $r\in V$，满足从 r 出发能够到达 V 中所有的点，则称 G 是一个“流图”，即为 (G,r)，其中 r 称为流图的源点。
与无向图类似，我们也可以定义“流图”的搜索树和时间戳的概念。
在一个流图 (G,r) 上从 r 出发进行深度优先遍历，每个节点只访问一次。所有发生递归的边 (x,y)（从 x 到 y 是对 y 的第一次访问）构成一棵以 r 为根的树，我们把它称为流图 (G,r) 的搜索树。
同时，在深度优先遍历的过程中，按照每个节点第一次被访问的时间顺序，依次给予流图中 N 个节点 $1\sim N$ 的证书标记，该标记被称为时间戳，记为 $dfn[x]$。
流图中的每条有向边必然是一下四种之一：

- 树枝边，指搜索树中的边，即 x 是 y 的父节点。
- 前向边，指搜索树中 x 是 y 的祖先节点。
- 后向边，指搜索树中 y 是 x 的祖先节点。
- 横叉边，指除了以上情况之外的边，它一定满足 $dfn[y]<dfn[x]$。

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，既存在从 x 到 y 的路径，也存在从 y 到 x 的路径，则称该有向图是“强连通图”。
有向图的极大强连通子图被称为“强连通分量”，简记为 SCC。此处“极大”的含义与之前类似。
Tarjan 算法基于有向图的深度优先遍历，能够在线性时间内找出一张有向图的各个强连通分量。
一个“环”一定是强连通图。如果既存在从 x 到 y 的路径，也存在从 y 到 x 的路径，那么 x, y 显然在一个环中。因此，Tarjan 算法的基本思路就是对于每个点，尽量找到与它一起能构成环的所有节点。
容易发现，前向边 (x, y) 没有什么用处，因为搜索书上本就存在 x 到 y 的路径。后向边 (x, y) 非常有用，它可以和搜索树上从 y 到 x 的路径一起构成环。横叉边 (x, y) 视情况而定，如果从 y 出发能找到一条路径回到 x 的祖先节点，那么 (x, y) 就是有用的。

有向图的强连通分量

为了找到通过“后向边”和“横叉边”构成的环，Tarjan 算法在深度优先遍历的同时维护了一个栈。当访问到节点 x 时，栈中需要保存一下两类节点：

- 搜索树上 x 的祖先节点，记为集合 $anc(x)$。
  设 $y \in anc(x)$。若存在后向边 (x, y)，则 (x, y) 与 y 到 x 的路径一起形成环。
- 已经访问过，并且存在一条路径到达 $anc(x)$ 的节点。
  设 z 是一个这样的点，从 z 出发存在一条路径到达 $y \in anc(x)$。若存在横叉边 (x, z)，则 (x, z)、z 到 y 的路径、y 到 x 的路径形成一个环。

综上所述，栈中的节点就是能与从 x 出发的后向边、横叉边形成环的节点。进而可以引入“追溯值”的概念。

有向图的强连通分量

追溯值

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

- 该点在栈中。
- 存在一条从 $subtree(x)$ 出发的有向边，以该点为终点。

根据定义，Tarjan 算法按照以下步骤计算“追溯值”：

- 当节点 x 第一次被访问时，把 x 入栈，初始化 $low[x] = dfn[x]$。
- 扫描从 x 出发的每条边 (x, y)：
  - 若 y 没被访问过，则说明 (x, y) 是树枝边，递归访问 y，从 y 回溯之后，令 $low[x] = \min(low[x], low[y])$。
  - 若 y 被访问过并且 y 在栈中，则令 $low[x] = \min(low[x], dfn[y])$。
- 从 x 回溯之前，判断是否有 $low[x] = dfn[x]$。若成立，则不断从栈中弹出节点，直至 x 出栈。

有向图的强连通分量

```
void tarjan(int x){
    dfs[x]=low[x]=++num;
    stack[++top]=x,ins[x]=1;//节点入栈并标记在栈中
    for(int i=head[x];i;i=Next[i])
        if(!dfs[ver[i]]){
            tarjan(ver[i]);
            low[x]=min(low[x],low[ver[i]]);
        }
        else if(ins[ver[i]])//注意判断 y 是否在栈中
            low[x]=min(low[x],dfn[ver[i]);
    if(dfn[x]==low[x]){//x 是一个强连通分量中深度最小的节点
        cnt++;int y;
        while(1){
            y=stack[top--],ins[y]=0;
            c[y]=cnt,scc[cnt].push_back(y);
            if(y==x) break;//弹出 x 到栈顶的节点构成强连通分量
        }
    }
}
```

例题

Network of Schools

一些学校连入一个电脑网络。那些学校已订立了协议：每个学校都会给其它的一些学校分发软件（称作“接受学校”）。注意即使 B 在 A 学校的分发列表中，A 也不一定在 B 学校的列表中。
你要写一个程序计算，根据协议，为了让网络中所有的学校都用上新软件，必须接受新软件副本的最少学校数目（子任务 1）。更进一步，我们想要确定通过给任意一个学校发送新软件，这个软件就会分发到网络中的所有学校。为了完成这个任务，我们可能必须扩展接收学校列表，使其加入新成员。计算最少需要增加几个扩展，使得不论我们给哪个学校发送新软件，它都会到达其余所有的学校（子任务 2）。一个扩展就是在一个学校的接收学校列表中引入一个新成员。
$2 \leq n \leq 100$。

Network of Schools

把学校看作节点，若学校 A 能支援学校 B，则从 A 到 B 连一条有向边，得到一张有向图。在有向图的一个强连通分量内，任意两个点都属相互可达的。因此，只要其中任何一个学校获得新软件，该强连通分量内其他学校都可以通过网络获得这个新软件。

可以求出所有强连通分量，并执行“缩点”过程，得到一张有向无环图。首先，“零入度点”无法被其它学校支援。其次，若同时向所有“零入读点”提供新软件，则新软件从这些点出发沿着网络显然能遍历到整个有向无环图。这样我们就得到第第一问的答案。

第二问就是问最少添加多少条有向边，可以把这张有向图变成强连通图。设缩点之后的有向无环图中有 p 个零入读点，q 个零出度点，那么答案就是 $\max(p, q)$。因为当 $\min(p, q) = 1$ 时，答案显然为 $\max(p, q)$；当 $\min(p, q) > 1$ 时，我们总是可以通过从一个零出度的点往一个零入度的点连边，把 p, q 都减小 1。特别地，如果整张图就是一个强连通分量，答案为 0。

例题

银河

银河中的恒星浩如烟海，但是我们只关注那些最亮的恒星。
我们用一个正整数来表示恒星的亮度，数值越大则恒星就越亮，恒星的亮度最暗是 1。
现在对于 N 颗我们关注的恒星，有 M 对亮度之间的相对关系已经判明。
你的任务就是求出这 N 颗恒星的亮度值总和至少有多大。
一对恒星 (A, B) 之间的亮度关系总共有五种，分别为相等、$A > B$、$A \geq B$、$A < B$、$A \leq B$。
$1 \leq N, M \leq 100000$。

银河

我们设 $d[A]$ 表示 A 的亮度，那么亮度关系可以写成 $d[A]-d[B]\geq 0/1$ 或 $d[B]-d[A]\geq 0/1$ 的形式。另外，由于恒星的亮度最暗是 1，我们可以设 $d[0]=0$，然后把这个条件写作 $\forall A, d[A]-d[0]\geq 1$。

我们当然可以根据差分约束系统相关的知识，连边后跑最长路。然而 n 有 10^5 级别，这样的时间复杂度是不对的。

仔细观察，我们根据差分约束系统建立的有向图中，边权只有 $0,1$ 两种。如果图中存在一个环，那么环上的边长度必须都是 0，否则一定无解。因此，我们可以用 Tarjan 算法求出图中所有的强连通分量，只要强连通分量内部存在长度为 1 的边，就可以直接判定无解。

如果有解，那么每个强连通分量内部各个恒星的亮度是相等的，我们可以通过缩点操作得到一张边权为 0 或 1 的有向无环图。从节点 0 所在的 SCC 出发，按照拓扑序执行动态规划，即可在 $O(N+M)$ 的时间内求出到达每个 SCC 的最长路，即该 SCC 内所有恒星的最小亮度。