from Tarjan to 2-SAT

难 度 : from Green to Purple

为什么讲Tarjan?

因为她很重要，并且感觉很多人都有一些很深的 Tarjan情结，再加上本人一直立下 flag想要搞懂 Tarjan，所以就讲 Tarjan了。

当然，CSP-S2022考了两道最短路，这也是讲 Tarjan的原因之一。

~~大胆猜测今年的所有考试中一定会有一道题要用 Tarjan缩点求强连通分量的 QwQ~~

什么是Tarjan?

众所周知，Tarjan是图论中非常常见的一种方法，它的用处包括但不仅限于缩点、求强连通分量（scc）。注意这里的强连通分量指的是有向图中的。

为什么我们需要 Tarjan?

利用 Tarjan，我们可以把有向图中的强连通分量缩成一个点，使原图成为一个 DAG（有向无环图）。

前置芝士

前置芝士

- 强连通：对于有向图中两点i，j，若存在i到j和j到i的路径，则称i，j强连通。
- 强连通：对于有向图中两点i，j，若存在i到j或j到i的路径，则称i，j弱连通。
- 强连通图：任意两点均强连通的图。
- 弱连通图：任意两点均弱连通的图。
- 强连通分量（scc）：有向图的极大强连通子图，即再向子图中加点构不成强连通图。
- DFS树：对图进行DFS之后删去所有未经过的边产生的树。
    - 树边：在DFS树上的边
    - 返祖边：对于u到v的边，如果v出栈的时候u还没有出栈，那么从v到u的边是一条返祖边
    - 横叉边
- DFS序（dfn）：点被 DFS 的顺序。

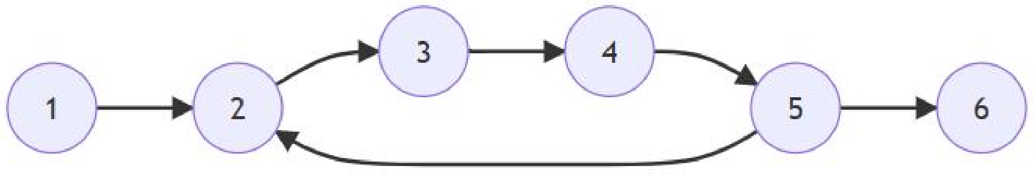
如何实现 Tarjan?

~~其实前置芝士并不重要~~

首先要理解 Tarjan 的思想，我们可以模拟一下这个过程。首先对于一个图，我们把它 DFS 一遍，记录每一个点被 DFS 时的编号（类似于时间戳的思想）。为什么要使用 DFS 捏？主要是因为在搜索的过程中，我们把每一个强连通分量都当成搜索树的一个子树，这样完整的流程结束之后我们得到的就是一棵完整的搜索树。

除此之外，我们还要维护一个 `low` 数组（写 Markdown 写习惯了），表示的是以当前节点为根节点，能够访问到的所有节点中时间戳最小的值，说的通俗易懂就是找它父亲的时间戳。对于每次找到的新点，我们令 `low[i]=dfn[i]`。

在搜索的过程中，我们把当前节点为根节点组成的搜索树中未处理的节点加入一个栈，DFS之后回溯时我们可以判断栈顶到栈中节点是否能构成强连通分量。



关于`low`数组

关于 Tarjan，自己认为比较难理解的可能就是这个`low`数组。对于`low[i]`“以当前节点为根节点，能够访问到的所有节点”这个定义，满足以下条件中的任意一个即可：

是以 i 为根节点的搜索树的子树上的所有节点。

或者可以通过一条非树边即返祖边到达以 i 为根节点的搜索树上的一个点。

举个栗子：

对于左下图来说，我们从 1 开始 DFS，如果我们要求`low[3]`的值，我们可以走到的子树节点是{4,5}，而 5→2这一条返祖边连接的 2 也属于可以到达的点，由于这里1的时间戳最小，所以`low[2]`的值为2。

那么，为什么我们需要`low`数组，其实`low`相等的点就在同一个强连通分量中啦！为什么`low`要选的是最小时间戳，对于左上图来说，我们可以把{2,3,4,5}缩成 2，这就是为什么我们要令这些的`low`都等于 2,，因为我们要维护 scc 的父节点。

关于`low`数组

由`low`数组的定义我们可以知道，对于边(u,v)来说：

如果为树边（即没有被访问过），那么`low[u]=min(low[u],low[v])`（尝试用相邻节点的`low`更新最小时间戳）

如果为返祖边或者横叉边，那么`low[u]=min(low[u],dfn[v])`。为什么？可不可以写成`low[u]=min(low[u],low[v])`捏？在运行的时候看似对答案没有影响，但却破坏了`low`数组的定义，因为`low`会被不断修正而不能保持稳定。实际上，`low`数组不需要保证一定是最小`dfn`，只需要保证这个子树中的`low`都小于最小`dfn`即可。之所以要令`low`为最小DFS序，就是为了方便找出scc内的节点。

所以对于这个柿子，含义是u的`low`值由通过返祖边到达的`dfn`或横叉边走到更新过的点的`dfn`更新而来。

算法流程

重点来啦！

对于首次搜索到的点u，记录一下 `dfn[u]`。

首先进行堆栈操作，把u压入栈顶，更新 `low[u]` 为 `dfn[u]`。

对于u能到达的点v，如果v不在栈中，那么证明是树边；如果u已经被弹出，那么证明是返祖边。所以说我们可以更新一个定义，对于u到v的边，如果v出栈的时候u还没有出栈，那么从v到u的边是一条返祖边。

如果遍历完全部u的子树后 `low[u]==dfn[u]`，可以想一下之前的图，我们把u和在u之后压入的元素全部出栈（注意栈是先进后出），出栈的所有元素组成一个强连通分量。为什么？大佬曾说，`low`数组说白了就是找爹，如果它能找着的最大爹是它自己，那么它一定是强连通分量的根，所以把它的子树全弹掉。

继续 DFS，直到所有点被遍历。

例题

讲 Tarjan 不讲例题，就像四大名著不看红楼梦，说明这个人文学造诣和自我修养不足，他理解不了这种内在的阳春白雪的高雅艺术，他只能看到外表的辞藻堆砌，参不透其中深奥的精神内核，他整个人的层次就卡在这里了，只能度过一个相对失败的人生。(逃

在这里奉上一道 Tarjan 缩点的题（附代码讲解），供大家食用 QwQ

经典永流传——P2341

我们先找出强连通分量，发现强连通分量里的牛牛一定互相喜欢，所以用 Tarjan 缩点形成 DAG。因为明星牛牛要被所有牛牛喜欢，于是乎明星牛牛不能被除了它所在的 scc 之外的牛牛喜欢。于是乎只需要统计一下是否存在出度为0的点（其实这里的点是 scc）。但是如果存在2个或2个以上的出度为0的点，那么说明存在分裂开的子图，即有一部分牛牛连接不到明星牛牛的 scc 中，输出0。

代码实现

```cpp
void tarjan(int u)
{
    low[u]=dfn[u]=++dfncnt;
    sta[++top]=u;vis[u]=1;
    for(int i=head[u];i;i=nxt[i])
    {
        int v=to[i];
        if(!dfn[v]) tarjan(v)/*继续往下搜*/,low[u]=min(low[u],low[v]);
        else if(vis[v]) low[u]=min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u])
    {
        ++sccnum;
        while(sta[top]!=u)//把它搜索树里的都弹完
            scc[sta[top]]=sccnum,vis[sta[top]]=0,siz[sccnum]++,top--;//数组模拟弹栈
        //处理它自己
        scc[sta[top]]=sccnum;
        vis[sta[top]]=0;
        siz[sccnum]++;
        top--;
    }
}

int main()
{
    int N,M,a,b;cin>>N>>M;
    for(int i=1;i<=M;i++)
        cin>>a>>b,add(a,b);
    for(int i=1;i<=N;i++) if(!dfn[i]) tarjan(i);
    for(int i=1;i<=N;i++)
        for(int j=head[i];j;j=nxt[j])
            if(scc[to[j]]!=scc[i])
                du[scc[i]]++;
    int ans=0,cntt=0;
    for(int i=1;i<=sccnum;i++)
    {
        if(!du[i])
        {
            cntt++;
            if(cntt>1) {ans=0;break;}
            ans+=siz[i];
        }
    }
    cout<<ans;
```

~~exTarjan~~
2-SAT

在学会了用 Tarjan 求强连通分量之后，我们可以尝试解决一道紫模板——2-SAT 问题。

关于 2-SAT，要把它拆开解释。所谓 SAT，就是 satisfiability（可满足性）的简称。而“2”指的就是对于多种可能只有两个条件，举个栗子，在写代码的时候，每个人都要求满足自己码风的一个：

Steven24：大括号换行（a），不 `#define int long long`（¬b）

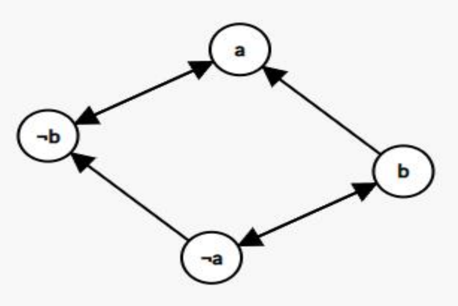
fchwpo：大括号换行（a），`#define int long long`（b）

Kazdale：大括号不换行（¬a），不 `#define int long long`（¬b）

所谓 2-SAT 问题，就是给定多个关系，每次关系涉及两个 `bool` 变量，对这些 `bool` 变量进行赋值使满足关系。

对于上面的问题，我们需要赋值使满足：
$(a\lor\neg b)\land(a\lor b)\land(\neg a\lor\neg b)$

如何解决 2-SAT 问题？



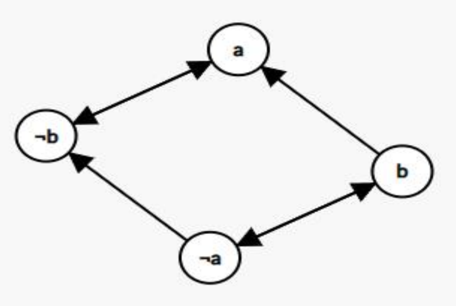
我们可以用强连通分量解决，虽然它们看似毫无关系，但是我大为震撼。

对于每一个要赋值的 `bool` 型变量，我们把它拆成两个点，`i` 和 `¬i`，我们可以这样建图：

对于一个同学的要求 $a∨b$，可以转化成两条边，因为如果有一个是假另一个一定是真。

那么，为什么不能建从 `b` 到 `¬a` 的边捏？因为如果 `b`成立 `a` 的真假无法确定。

式子	建图
$a \vee b$	$\neg a \to b \wedge \neg b \to a$
$\neg a \vee b$	$a \to b \wedge \neg b \to \neg a$
$a \vee \neg b$	$\neg a \to \neg b \wedge b \to a$
$\neg a \vee \neg b$	$a \to \neg b \wedge b \to \neg a$



如何解决 2-SAT 问题？

对于左上图，我们可以发现，`a` 和 `¬b` 在同一强连通分量内，同时 `¬a` 和 `b` 也在同一强连通分量内。这意味着什么捏？意味着我们只要知道一个点的 `bool` 值，就可以推知整个强连通分量内部的其他 `bool` 值，因为它们一定都是相等的（因为建边的前提就是可以推出）。~~这不爽死了么~~

对于找到的强连通分量，我们进行缩点处理。那么缩完之后的这个点 `bool` 是多少捏？

~~习惯上，~~我们给缩点之后的图进行拓补排序，如果 `a` 所在的强连通分量缩点后拓补序小于 `¬a` 的，那么令 `a` 为真（由于 Tarjan 是 DFS 实现的，所以相当于它已经帮你把 scc 排好序了 QwQ。

（拓补序：如果 x 能到 y，那么 x 的拓补序小于 y。）

什么时候会存在无解情况捏？如果 `a` 和 `¬a` 在同一个强连通分量中，那么 `a` 和 `¬a` 一定不相等，所以无解。

板子题
P4782

题目描述

复制Markdown 展开

有 n 个布尔变量 x_1~x_n，另有 m 个需要满足的条件，每个条件的形式都是「x_i 为 `true` / `false` 或 x_j 为 `true` / `false`」。比如「x_1 为真或 x_3 为假」、「x_7 为假或 x_2 为假」。

2-SAT 问题的目标是给每个变量赋值使得所有条件得到满足。

输入格式

第一行两个整数 n 和 m，意义如题面所述。

接下来 m 行每行 4 个整数 i, a, j, b，表示「x_i 为 a 或 x_j 为 b」$(a, b \in \{0, 1\})$

输出格式

如无解，输出 `IMPOSSIBLE`；否则输出 `POSSIBLE`。

下一行 n 个整数 $x_1 \sim x_n$（$x_i \in \{0, 1\}$），表示构造出的解。

代码实现

Tarjan求强连通分量缩点：

```cpp
void tarjan(int u)
{
    low[u]=dfn[u]=++dfncnt;
    sta[++top]=u;vis[u]=1;
    for(int i=head[u];i;i=nxt[i])
    {
        int v=to[i];
        if(!dfn[v]) tarjan(v)/*继续往下搜*/,low[u]=min(low[u],low[v]);
        else if(vis[v]) low[u]=min(low[u],dfn[v]);
    }
    if(dfn[u]==low[u])
    {
        ++sccnum;
        while(sta[top]!=u)//把它搜索树里的都弹完
            scc[sta[top]]=sccnum,vis[sta[top]]=0,top--;//数组模拟弹栈
        //处理它自己
        scc[sta[top]]=sccnum;
        vis[sta[top]]=0;
        top--;
    }
}
```

代码实现

建图+跑Tarjan+判断+A了这道题

```
int main()
{
    int n,m;cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int ii,a,jj,b;cin>>ii>>a>>jj>>b;
        //实现建两个点，对于一个点k，k+n是true点，k是false点
        add(ii+!a*n,jj+b*n);
        add(jj+!b*n,ii+a*n);
    }
    for(int i=1;i<=2*n;i++) if(!dfn[i]) tarjan(i);
    for(int i=1;i<=n;i++)
        if(scc[i]==scc[i+n]) puts("IMPOSSIBLE"),exit(0);
    puts("POSSIBLE");
    for(int i=1;i<=n;i++)
    {
        if(scc[i]<scc[i+n]) cout<<0<<' ';
        else cout<<1<<' ';
    }
    return 0;
}
```