强联通分量（SCC）—— Tarjan

定义

1. 强连通： 在一个有向图 $G = (V, E)$ 中，如果对于图中的任意两个顶点 u 和 v（u 和 v 可以是同一个顶点），都存在一条从 u 到 v 的路径，同时也存在一条 从 v 到 u 的路径，那么我们就称这个有向图 G 是强连通图。
2. 强连通分量： 一个有向图的强连通分量是其最大的强连通子图。
3. “最大”意味着：你无法再向这个子图中添加任何该图中的其他顶点，同时还能保持子图的强连通性。

一个顶点自身（没有自环边）也构成一个强连通分量（因为从自己到自己的路径可以认为是空路径或长度为0的路径）。

一个有向图**不是强连通图**时，它可以被分解成若干个 *极大强连通子图* ，这些子图就是该图的强连通分量。

应用

常被用来简化图的结构，使用 SCC 缩点来创造 DAG（有向无环图），为其他各种依赖于 DAG 的算法实现创造条件

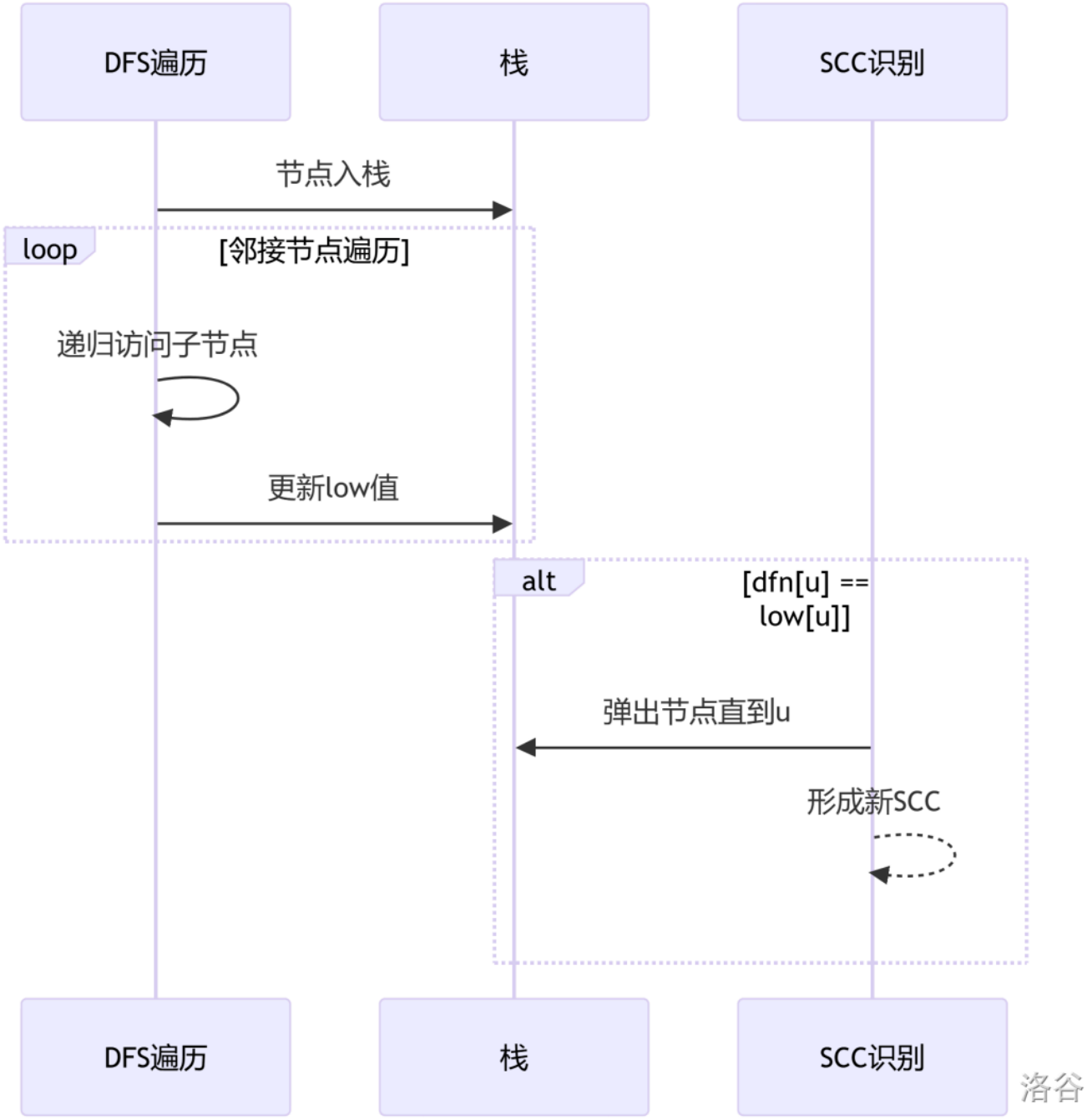
算法流程（Tarjan）

`dfn[u]`：DFS 访问节点 u 的时间戳（发现顺序）

`low[u]`：节点 u 通过 DFS 树边和后向边能回溯到的最早祖先节点的 dfn 值

栈结构：存储当前 DFS 路径上的节点

流程示意图：



核心代码

```
void Tarjan(int u) {
    dfn[u] = low[u] = ++timestamp;
    s.push(u);
    ins[u] = true;
    for (auto v : g[u]) {
        if (!dfn[v]) {
            Tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (ins[v]) {// 子树内查询
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        ++cnt_col;
```

```
        int t;
        do {
            t = s.top();
            s.pop();
            ins[t] = false;
            col[t] = cnt_col;  // 染色代替并查集
        } while (t != u);
    }
}
```

例题1

P3387 【模板】缩点

如题，这是一道板子题，不做赘述。

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int maxn = 1e5+5,maxm = 3e5+5;
int n,m,u,v,ta,tb,timestamp,ans;
int dfn[maxn],low[maxn],a[maxn],in[maxn],dis[maxn];
vector<int>e[maxn],e2[maxn];
stack<int>s;
bool ins[maxn];
void Tarjan(int u) {
    dfn[u] = low[u] = ++timestamp;
    s.push(u);
    ins[u] = true;
    for (auto v : g[u]) {
        if (!dfn[v]) {
            Tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (ins[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        cnt_col++;
        int t;
        do {
            t = s.top();
            s.pop();
            ins[t] = false;
            col[t] = cnt_col;  // 染色代替并查集
        } while (t != u);
    }
}
void topo(){
        queue<int> q;
        for(int i=1;i<=n;i++){
                if(!in[i]){
                        q.push(i);
```

```
                        dis[i] = a[i];
                }
        }
        while(!q.empty()){
                u = q.front();q.pop();
                while(!e2[u].empty()){
                        int v = e2[u].back();e2[u].pop_back();
                        dis[v] = max(dis[v],dis[u] + a[v]);
                        if(--in[v] == 0)q.push(v);
                }
        }
}



signed main(){
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++)scanf("%d",&a[i]);
        for(int i=1;i<=m;i++){
                scanf("%d%d",&u,&v);
                e[u].push_back(v);
        }
        for(int i=1;i<=n;i++)
                if(!dfn[i])tarjan(i);//未被访问
        for(int i=1;i<=n;i++){
                ta = col[i];
                for(auto p:e[i]){
                        tb = col[p];
                        if(ta != tb){
                                e2[ta].push_back(tb);
                                in[tb]++;
                        }
                }

        }
        topo();
        for(int i = 1;i <= n;i++)ans = max(ans,dis[i]);
        cout<<ans;
        return 0;
}
```

例题2

P4819 [中山市选] 杀人游戏

注意：这里可以使用排除法优化一次（有且仅有一次）的查询次数

具体来说：

基本思路

在缩点后的 DAG 中，通常需要查证所有入度为 0 的强连通分量（SCC）。

查证次数 $=$ 入度为 0 的SCC数量（设为 k）。

安全概率 $= 1 - k/n$。

为什么需要特殊检查

存在一种特殊情况：可以通过排除法确定最后一个SCC的身份，从而减少一次查证

这需要满足特定条件，使得某个入度为0的SCC不需要实际查证

下面我们来讨论这个条件

特殊分量的条件

一个入度为 0 的SCC满足以下条件时可优化：

大小为 1：该SCC只包含单个节点

所有后继的入度≥2：该SCC指向的所有节点在DAG中的入度≥2

即每个后继节点至少有两条独立的路径可以到达

为什么可以优化

当满足上述条件时：

警察先查证其他所有入度为 0 的SCC

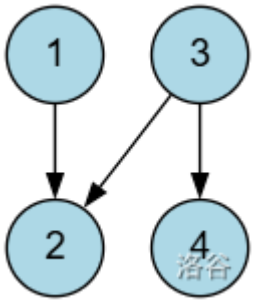
如果这些查证的都是平民，就能推断出整个图中除该SCC外的所有身份

根据"只有一个杀手"的条件，剩下的唯一未查证SCC必然是杀手

因此不需要实际查证该SCC，避免可能的危险

举个例子

考虑下面这张图：



我们只需要查询 3 号节点。

原因如下：

1. 查询 3 号节点 $\Rightarrow$ 知道 $2, 3, 4$ 号节点的身份。
2. 若 $2, 3, 4$ 号节点的身份均为平民，则 1 号节点一定为杀手。

Code

```
#include <bits/stdc++.h>
#define int long long
using namespace std;
const int maxn = 1e5+5,maxm = 3e5+5;
int n,m,u,v,ta,tb,timestamp,cnt;
int dfn[maxn],low[maxn],a[maxn],in[maxn],dis[maxn];
vector<int>e[maxn],e2[maxn];
stack<int>s;
bool ins[maxn];
void Tarjan(int u) {
    dfn[u] = low[u] = ++timestamp;
    s.push(u);
    ins[u] = true;
    for (auto v : g[u]) {
        if (!dfn[v]) {
            Tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (ins[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        cnt_col++;
        int t;
        do {
            t = s.top();
            s.pop();
            ins[t] = false;
            col[t] = cnt_col;  // 染色代替并查集
            if(t!=u)a[col[s.top]]++;
        } while (t != u);
    }
}
signed main(){
        scanf("%d%d",&n,&m);
        for(int i=1;i<=n;i++){
                a[i]=1;//记录scc的大小
        }
        for(int i=1;i<=m;i++){
                scanf("%d%d",&u,&v);
                e[u].push_back(v);
        }
        for(int i=1;i<=n;i++)
                if(!dfn[i])tarjan(i);
        for(int i=1;i<=n;i++){
                ta = dsu::find(i);
                for(auto p:e[i]){
                        tb = dsu::find(p);
                        if(ta != tb){
                                e2[ta].push_back(tb);
                                in[tb]++;
                        }
```

```
                }
        }
        bool flag=false;
        for(int i=1;i<=n;i++){
                if(dsu::find(i) == i && in[i]==0){//找到一个scc
                        cnt++;
                        if(a[i]==1 && !flag){
                                flag = true;
                                for(auto p:e[i]){
                                        if(in[p]<2)flag=false;
                                }
                        }
                }
        }
        double Ex = 1-((double)cnt-(flag?1:0))/n;
        printf("%.6f",Ex);
        return 0;
}
```