

# Tarjan

强连通分量、点/边双连通分量、割点/边

Epsilon

北京理工大学

2023 年 8 月 14 日



## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强连通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量

- 强连通：在有向图  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果从  $u$  到  $v$  可以互相到达，那么称在该图中  $u$  和  $v$  强连通的。
- 强连通图：在有向图  $G = (V, E)$  中，任意两个结点都强连通，则称该图为强连通图。
- 强连通分量：有向图的极大强连通子图称为强连通分量（Strongly Connected Components, SCC）。

- 强连通：在有向图  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果从  $u$  到  $v$  可以互相到达，那么称在该图中  $u$  和  $v$  强连通的。
- 强连通图：在有向图  $G = (V, E)$  中，任意两个结点都强连通，则称该图为强连通图。
- 强连通分量：有向图的极大强连通子图称为强连通分量 (Strongly Connected Components, SCC)。

- 强连通：在有向图  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果从  $u$  到  $v$  可以互相到达，那么称在该图中  $u$  和  $v$  强连通的。
- 强连通图：在有向图  $G = (V, E)$  中，任意两个结点都强连通，则称该图为强连通图。
- 强连通分量：有向图的极大强连通子图称为强连通分量（Strongly Connected Components, SCC）。

- 割点：对于一个**无向图**，如果把一个点删除后这个图的极大连通分量数增加了，那么这个点就是这个图的割点。
- 割边（桥）：对于一个**无向图**，如果删掉一条边后图中的连通分量数增加了，则称这条边为桥或者割边。

- 割点：对于一个**无向图**，如果把一个点删除后这个图的极大连通分量数增加了，那么这个点就是这个图的割点。
- 割边（桥）：对于一个**无向图**，如果删掉一条边后图中的连通分量数增加了，则称这条边为桥或者割边。

- **连通**：在**无向图**  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果存在一条路径  $v_0, v_1, \dots, v_k$ ，其中  $v_0 = u, v_k = v$ ，那么称在该图中  $u$  和  $v$  是连通的。
- **连通图**：在**无向图**  $G = (V, E)$  中，任意两个结点都连通，则称该图是连通图。
- **连通分量**：**无向图**的极大连通子图称为连通分量。
- **点双联通**：在一个连通的**无向图**中，对于两个点  $u$  和  $v$ ，如果无论删去哪一个点（只能删去一条，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。
- **点双联通图**：在一个连通的**无向图**中，如果删除任意一个点都不改变图的连通性，那么称该图为点双连通图。
- **点双联通分量**：**无向图**中的极大点双联通子图称为点双联通分量。



- 连通：在**无向图**  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果存在一条路径  $v_0, v_1, \dots, v_k$ ，其中  $v_0 = u, v_k = v$ ，那么称在该图中  $u$  和  $v$  是连通的。
- 连通图：在**无向图**  $G = (V, E)$  中，任意两个结点都连通，则称该图是连通图。
- 连通分量：无向图的极大连通子图称为连通分量。
- 点双联通：在一个连通的**无向图**中，对于两个点  $u$  和  $v$ ，如果无论删去哪一个点（只能删去一条，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。
- 点双联通图：在一个连通的**无向图**中，如果删除任意一个点都不改变图的连通性，那么称该图为点双连通图。
- 点双联通分量：无向图中的极大点双联通子图称为点双联通分量。

- 连通：在无向图  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果存在一条路径  $v_0, v_1, \dots, v_k$ ，其中  $v_0 = u, v_k = v$ ，那么称在该图中  $u$  和  $v$  是连通的。
- 连通图：在无向图  $G = (V, E)$  中，任意两个结点都连通，则称该图是连通图。
- 连通分量：无向图的极大连通子图称为连通分量。
- 点双联通：在一个连通的无向图中，对于两个点  $u$  和  $v$ ，如果无论删去哪一个点（只能删去一条，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。
- 点双联通图：在一个连通的无向图中，如果删除任意一个点都不改变图的连通性，那么称该图为点双连通图。
- 点双联通分量：无向图中的极大点双联通子图称为点双联通分量。

- 连通：在**无向图**  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果存在一条路径  $v_0, v_1, \dots, v_k$ ，其中  $v_0 = u, v_k = v$ ，那么称在该图中  $u$  和  $v$  是连通的。
- 连通图：在**无向图**  $G = (V, E)$  中，任意两个结点都连通，则称该图是连通图。
- 连通分量：**无向图**的极大连通子图称为连通分量。
- 点双联通：在一个连通的**无向图**中，对于两个点  $u$  和  $v$ ，如果无论删去哪一个点（只能删去一条，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。
- 点双联通图：在一个连通的**无向图**中，如果删除任意一个点都不改变图的连通性，那么称该图为点双连通图。
- 点双联通分量：**无向图**中的极大点双联通子图称为点双联通分量。

- 连通：在**无向图**  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果存在一条路径  $v_0, v_1, \dots, v_k$ ，其中  $v_0 = u, v_k = v$ ，那么称在该图中  $u$  和  $v$  是连通的。
- 连通图：在**无向图**  $G = (V, E)$  中，任意两个结点都连通，则称该图是连通图。
- 连通分量：**无向图**的极大连通子图称为连通分量。
- 点双联通：在一个连通的**无向图**中，对于两个点  $u$  和  $v$ ，如果无论删去哪一个点（只能删去一条，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。
- 点双联通图：在一个连通的**无向图**中，如果删除任意一个点都不改变图的连通性，那么称该图为点双联通图。
- 点双联通分量：**无向图**中的极大点双联通子图称为点双联通分量。

- 连通：在**无向图**  $G = (V, E)$  中，对于两个结点  $u, v \in V$ ，如果存在一条路径  $v_0, v_1, \dots, v_k$ ，其中  $v_0 = u, v_k = v$ ，那么称在该图中  $u$  和  $v$  是连通的。
- 连通图：在**无向图**  $G = (V, E)$  中，任意两个结点都连通，则称该图是连通图。
- 连通分量：**无向图**的极大连通子图称为连通分量。
- 点双联通：在一个连通的**无向图**中，对于两个点  $u$  和  $v$ ，如果无论删去哪一个点（只能删去一条，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。
- 点双联通图：在一个连通的**无向图**中，如果删除任意一个点都不改变图的连通性，那么称该图为点双联通图。
- 点双联通分量：**无向图**中的极大点双联通子图称为点双联通分量。

- 边双连通：在一个连通的**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。
- 边双联通图：在一个连通的**无向图**中，如果删除任意一条边都不改变图的连通性，那么称该图为点双连通图。
- 边双联通分量：**无向图**中的极大边双联通子图称为边双联通分量。

- 边双连通：在一个连通的**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。
- 边双联通图：在一个连通的**无向图**中，如果删除任意一条边都不改变图的连通性，那么称该图为点双连通图。
- 边双联通分量：无向图中的极大边双联通子图称为边双联通分量。

- 边双连通：在一个连通的**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。
- 边双联通图：在一个连通的**无向图**中，如果删除任意一条边都不改变图的连通性，那么称该图为点双连通图。
- 边双联通分量：**无向图**中的极大边双联通子图称为边双联通分量。



## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强连通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量

## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强连通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量

首先我们先来一段基本的 DFS 代码。

```
bool flag[MAXN];  
void dfs(int x) {  
    flag[x] = true;  
    for(int y : son[x]) {  
        if(!flag[y]) dfs(y);  
    }  
}
```

在基础的 DFS 代码上，为了方便，我们添加一个  $dfn$  数组用来记录每一个节点第一次被访问的时间（时间戳）。

```
int cnt, dfn[MAXN];  
void dfs(int x) {  
    dfn[x] = ++cnt;  
    for(int y : son[x]) {  
        if(!dfn[y]) dfs(y);  
    }  
}
```

$dfn$  数组单调递增，范围为 1 到  $n$ 。

在基础的 DFS 代码上，为了方便，我们添加一个  $dfn$  数组用来记录每一个节点第一次被访问的时间（时间戳）。

```
int cnt, dfn[MAXN];  
void dfs(int x) {  
    dfn[x] = ++cnt;  
    for(int y : son[x]) {  
        if(!dfn[y]) dfs(y);  
    }  
}
```

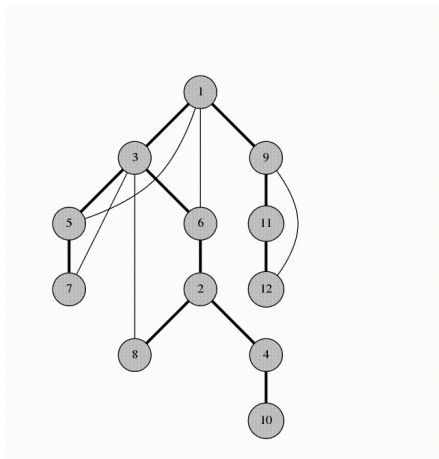
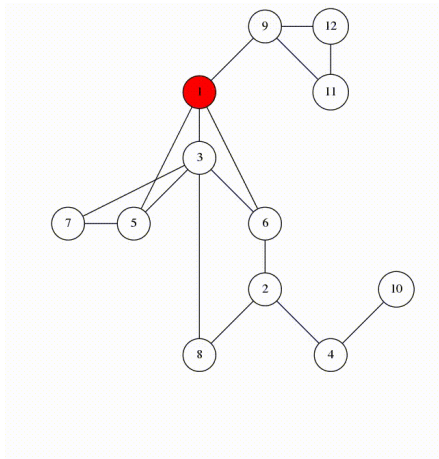
$dfn$  数组单调递增，范围为 1 到  $n$ 。

在基础的 DFS 代码上，为了方便，我们添加一个  $dfn$  数组用来记录每一个节点第一次被访问的时间（时间戳）。

```
int cnt, dfn[MAXN];  
void dfs(int x) {  
    dfn[x] = ++cnt;  
    for(int y : son[x]) {  
        if(!dfn[y]) dfs(y);  
    }  
}
```

$dfn$  数组单调递增，范围为 1 到  $n$ 。

对一个图运行上面的 DFS 算法，得到了一棵树，这棵树叫做 DFS 树，举一个例子（以 1 号节点为起点；在与  $u$  相邻的点中，先枚举编号小的点）：



## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强联通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量



在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。

在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。

在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。

在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

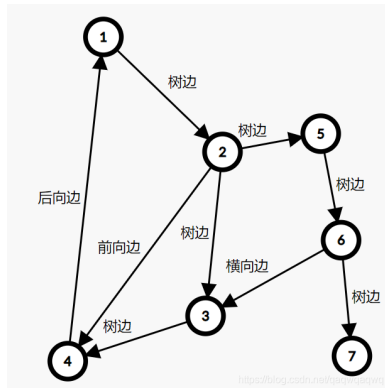
1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。

在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。

在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

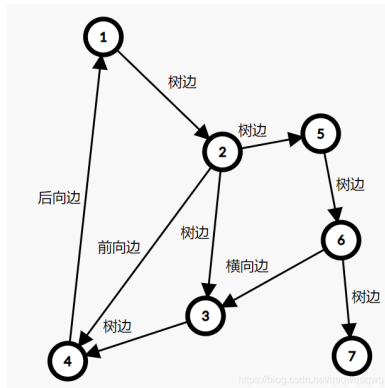
1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。



在介绍 Tarjan 之前，我们首先要利用 DFS 对所有的边分类。这些边可以分成四类：

1. 树边：在 DFS 树上的边称为树边。
2. 后向边（返祖边）：指结点  $u$  连向其在 DFS 树上的祖先结点  $v$  的边，其中， $u$  的自环也属于后向边。
3. 前向边：指结点  $u$  连向其在 DFS 树上的后代结点  $v$  的非树边。
4. 横向边：所有的其他边叫做横向边，即边  $(u, v)$  中  $u$  和  $v$  没有祖先关系。

如果该图是无向图，可以无向边拆成两条有向边，也适用这种分法。



## 性质 1

无向图没有横向边。

对于每一对有边相连的点  $(u, v)$ :

- 设  $u$  先于  $v$  被访问。
- 在枚举到  $u$  时, 设与  $u$  相连的边按照顺序为  $[t_0, t_1, \dots, t_k]$ ,  $t_i = v$ .
  - 若在访问完  $t_0$  到  $t_{i-1}$  之后,  $v$  还未被访问, 之后就会访问  $v$ ,  $(u, v)$  为树边。
  - 若在访问完  $t_0$  到  $t_{i-1}$  之后,  $v$  已经被访问, 那么  $v$  必然是被  $u$  的子树上的点访问的, 所以  $u$  是  $v$  的祖先。
- 综上,  $(u, v)$  不是横向边。

在无向图中, 只存在树边和非树边 (也就是有向图中的前/后向边)。



## 性质 1

无向图没有横向边。

对于每一对有边相连的点  $(u, v)$ :

- 设  $u$  先于  $v$  被访问。
- 在枚举到  $u$  时, 设与  $u$  相连的边按照顺序为  $[t_0, t_1, \dots, t_k]$ ,  $t_i = v$ 。
  - 若在访问完  $t_0$  到  $t_{i-1}$  之后,  $v$  还未被访问, 之后就会访问  $v$ ,  $(u, v)$  为树边。
  - 若在访问完  $t_0$  到  $t_{i-1}$  之后,  $v$  已经被访问, 那么  $v$  必然是被  $u$  的子树上的点访问的, 所以  $u$  是  $v$  的祖先。
- 综上,  $(u, v)$  不是横向边。

在无向图中, 只存在树边和非树边 (也就是有向图中的前/后向边)。

## 性质 2

在有向图中，如果  $(u, v)$  是后向边或横向边，那么  $v$  必然先于  $u$  被访问。

后向边指向祖先，祖先肯定先访问，没问题。看横向边：

- 如果  $u$  和  $v$  同时被访问，则为后向边（因为我们认为自环也属于后向边）。
- 如果  $u$  先被访问，那么类似性质 1，我们发现  $(u, v)$  不会是横向边。

综上，如果  $(u, v)$  是横向边，那么  $v$  必然先于  $u$  被访问。

## 性质 3

在有向图中，如果  $(u, v)$  是树边或前向边，那么  $u$  必然先于  $v$  被访问。

根据 DFS 的性质可得，在 DFS 树中，如果  $v$  在  $u$  的子树中，那么访问  $v$  之前必然先访问  $u$ 。

## 性质 2

在有向图中，如果  $(u, v)$  是后向边或横向边，那么  $v$  必然先于  $u$  被访问。

后向边指向祖先，祖先肯定先访问，没问题。看横向边：

- 如果  $u$  和  $v$  同时被访问，则为后向边（因为我们认为自环也属于后向边）。
- 如果  $u$  先被访问，那么类似性质 1，我们发现  $(u, v)$  不会是横向边。

综上，如果  $(u, v)$  是横向边，那么  $v$  必然先于  $u$  被访问。

## 性质 3

在有向图中，如果  $(u, v)$  是树边或前向边，那么  $u$  必然先于  $v$  被访问。

根据 DFS 的性质可得，在 DFS 树中，如果  $v$  在  $u$  的子树中，那么访问  $v$  之前必然先访问  $u$ 。

## 性质 2

在有向图中，如果  $(u, v)$  是后向边或横向边，那么  $v$  必然先于  $u$  被访问。

后向边指向祖先，祖先肯定先访问，没问题。看横向边：

- 如果  $u$  和  $v$  同时被访问，则为后向边（因为我们认为自环也属于后向边）。
- 如果  $u$  先被访问，那么类似性质 1，我们发现  $(u, v)$  不会是横向边。

综上，如果  $(u, v)$  是横向边，那么  $v$  必然先于  $u$  被访问。

## 性质 3

在有向图中，如果  $(u, v)$  是树边或前向边，那么  $u$  必然先于  $v$  被访问。

根据 DFS 的性质可得，在 DFS 树中，如果  $v$  在  $u$  的子树中，那么访问  $v$  之前必然先访问  $u$ 。

## 性质 2

在有向图中，如果  $(u, v)$  是后向边或横向边，那么  $v$  必然先于  $u$  被访问。

后向边指向祖先，祖先肯定先访问，没问题。看横向边：

- 如果  $u$  和  $v$  同时被访问，则为后向边（因为我们认为自环也属于后向边）。
- 如果  $u$  先被访问，那么类似性质 1，我们发现  $(u, v)$  不会是横向边。

综上，如果  $(u, v)$  是横向边，那么  $v$  必然先于  $u$  被访问。

## 性质 3

在有向图中，如果  $(u, v)$  是树边或前向边，那么  $u$  必然先于  $v$  被访问。

根据 DFS 的性质可得，在 DFS 树中，如果  $v$  在  $u$  的子树中，那么访问  $v$  之前必然先访问  $u$ 。

在 DFS 树上的边称为树边。在之前的 DFS 中，我们有这样的一句话：

```
if(!dfn[y]) dfs(y);
```

如果满足这个条件，那么  $(x, y)$  这条边就在 DFS 树上，就是树边。

连向祖先结点（包括连向自己）的边称为后向边。

- 首先，我们应该如何知道这个结点的祖先有哪些呢？用栈！
- 在 DFS 搜索到结点  $x$  时，我们将  $x$  入栈，在访问  $x$  完成回溯时，将  $x$  出栈。这样在栈里的结点就是  $x$  的祖先（包括自己）。

我们用一个数组  $instk_x$  来表示  $x$  是否在栈中。

在枚举与  $x$  相邻的点  $y$  时，如果  $y$  已经被访问过，且在栈中，那么  $(x, y)$  就是一条后向边。

```
if(!dfn[y]) {  
    //  $x \rightarrow y$  是树边  
    dfs(y);  
} else if(instk[y]) { //  $y$  被访问过  
    //  $x \rightarrow y$  是后向边  
}
```



- 连向后代结点的非树边称为前向边。
- 如果边  $(u, v)$  的两个节点  $u$  和  $v$  没有祖先关系，则称为该边为横向边。

在枚举与  $x$  相邻的点  $y$  时，如果  $y$  已经被访问过，且不在栈中，那么边  $(x, y)$  要么是前向边，要么是横向边。

- 由性质 2，如果  $dfn_y > dfn_x$ ，说明  $x$  先于  $y$  被访问，这样的边不是横向边，那么它就是前向边。
- 由性质 3，如果  $dfn_y < dfn_x$ ，说明  $y$  先于  $x$  被访问，这样的边不是前向边，那么它就是横向边。
- 不会出现  $dfn_y = dfn_x$ ，两个节点  $x$  和  $y$  的  $dfn$  值相等，那么肯定  $x = y$ ，这是后向边，在前面已经被考虑过了。

根据前面的讨论，我们可以更进一步完善 DFS 的代码：

```
int cnt, top, dfn[MAXN], stk[MAXN];
bool instk[MAXN];
void dfs(int x) {
    dfn[x] = ++cnt;
    stk[++top] = x, instk[x] = true;
    for(int y : son[x]) {
        if(!dfn[y]) {
            // x->y 是树边
            dfs(y);
        } else if(instk[y]) { // y 被访问过
            // x->y 是后向边
        } else if(dfn[y] > dfn[x]) { // x 先于 y 被访问
            // x->y 是前向边
        } else { // y 先于 x 被访问
            // x->y 是横向边
        }
    }
    --top, instk[x] = false;
}
```

## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强联通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量

强连通图是指任意两个结点都能相互到达的图，而强连通分量指的是有向图的极大强连通子图，英文缩写叫 SCC。

## 定义 1

设有向图  $G = (V, E)$ ，对于  $\forall u, v \in V$ ，都存在一条路径从  $u$  出发到  $v$ ，那么称  $G$  是强连通图。

## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强联通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量

## 推论 1 (SCC 判定)

设有向图  $G = (V, E)$ , 如果  $\exists u \in V$ ,  $u$  可以到达其他所有结点, 且其他所有结点都可以到达  $u$ , 那么  $G$  是强连通图。

对于  $\forall v_1, v_2 \in V, v_1, v_2 \neq u$ , 存在一条路径从  $v_1$  到  $u$ , 也存在一条路径从  $u$  到  $v_2$ , 所以  $v_1$  可以到达  $v_2$ 。根据定义可得推论 1 正确。

## 推论 2

设有向图  $G = (V, E)$ ,  $u, v \in V$ ,  $SCC_u$  表示包含  $u$  的强联通分量的点集。如果  $u$  可以到  $v$ , 且  $v$  可以到  $u$ , 那么  $v \in SCC_u$ 。

加入点  $v$  之后显然仍满足推论 1 的条件。

## 推论 1 (SCC 判定)

设有向图  $G = (V, E)$ , 如果  $\exists u \in V$ ,  $u$  可以到达其他所有结点, 且其他所有结点都可以到达  $u$ , 那么  $G$  是强连通图。

对于  $\forall v_1, v_2 \in V, v_1, v_2 \neq u$ , 存在一条路径从  $v_1$  到  $u$ , 也存在一条路径从  $u$  到  $v_2$ , 所以  $v_1$  可以到达  $v_2$ 。根据定义可得推论 1 正确。

## 推论 2

设有向图  $G = (V, E)$ ,  $u, v \in V$ ,  $SCC_u$  表示包含  $u$  的强联通分量的点集。如果  $u$  可以到  $v$ , 且  $v$  可以到  $u$ , 那么  $v \in SCC_u$ 。

加入点  $v$  之后显然仍满足推论 1 的条件。

## 推论 1 (SCC 判定)

设有向图  $G = (V, E)$ , 如果  $\exists u \in V$ ,  $u$  可以到达其他所有结点, 且其他所有结点都可以到达  $u$ , 那么  $G$  是强连通图。

对于  $\forall v_1, v_2 \in V, v_1, v_2 \neq u$ , 存在一条路径从  $v_1$  到  $u$ , 也存在一条路径从  $u$  到  $v_2$ , 所以  $v_1$  可以到达  $v_2$ 。根据定义可得推论 1 正确。

## 推论 2

设有向图  $G = (V, E)$ ,  $u, v \in V$ ,  $SCC_u$  表示包含  $u$  的强联通分量的点集。如果  $u$  可以到  $v$ , 且  $v$  可以到  $u$ , 那么  $v \in SCC_u$ 。

加入点  $v$  之后显然仍满足推论 1 的条件。



## 推论 1 (SCC 判定)

设有向图  $G = (V, E)$ , 如果  $\exists u \in V$ ,  $u$  可以到达其他所有结点, 且其他所有结点都可以到达  $u$ , 那么  $G$  是强连通图。

对于  $\forall v_1, v_2 \in V, v_1, v_2 \neq u$ , 存在一条路径从  $v_1$  到  $u$ , 也存在一条路径从  $u$  到  $v_2$ , 所以  $v_1$  可以到达  $v_2$ 。根据定义可得推论 1 正确。

## 推论 2

设有向图  $G = (V, E)$ ,  $u, v \in V$ ,  $SCC_u$  表示包含  $u$  的强联通分量的点集。如果  $u$  可以到  $v$ , 且  $v$  可以到  $u$ , 那么  $v \in SCC_u$ 。

加入点  $v$  之后显然仍满足推论 1 的条件。

设有向图  $G = (V, E)$ , 它的 DFS 树为  $T$ ,  $SCC_u$  表示包含  $u$  的强联通分量的点集,  $PATH_{u,v}$  表示  $T$  中  $u$  到  $v$  路径上的点集。

## 推论 3 (SCC 的连续性)

对于  $u, v \in V$ , 如果  $u$  可以到达  $v$ , 且  $v \in SCC_u$ , 那么对于  $\forall v' \in PATH_{u,v}$ ,  $v' \in SCC_u$ 。

- $v \in SCC_u$ , 那么  $v$  可以到达  $u$ 。
- $PATH_{u,v}$  中的所有点都可以到达  $v$ ,  $v$  可以到达  $u$ , 那么  $PATH_{u,v}$  中的所有点都可以到达  $u$ 。
- $u$  可以到达  $PATH_{u,v}$  中的所有点。
- 根据推论 2 可得该推论正确。

进一步可得 SCC 的在树上的连续性。

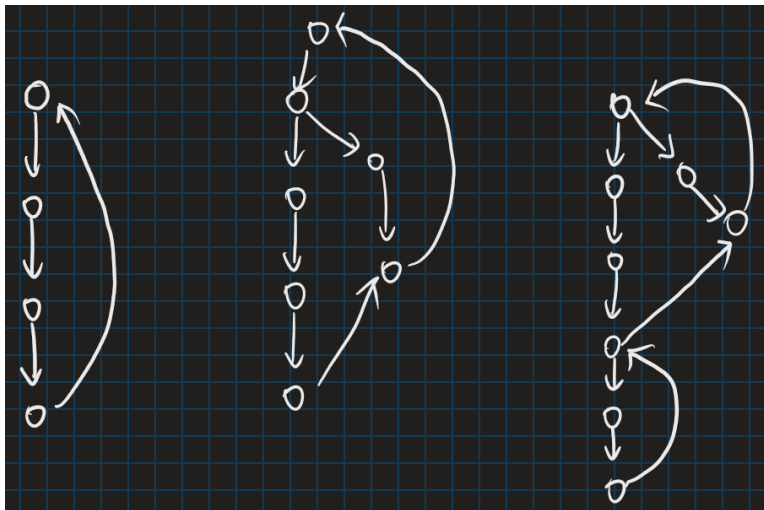
设有向图  $G = (V, E)$ ，它的 DFS 树为  $T$ ， $SCC_u$  表示包含  $u$  的强联通分量的点集， $PATH_{u,v}$  表示  $T$  中  $u$  到  $v$  路径上的点集。

## 推论 3 (SCC 的连续性)

对于  $u, v \in V$ ，如果  $u$  可以到达  $v$ ，且  $v \in SCC_u$ ，那么对于  $\forall v' \in PATH_{u,v}$ ， $v' \in SCC_u$ 。

- $v \in SCC_u$ ，那么  $v$  可以到达  $u$ 。
- $PATH_{u,v}$  中的所有点都可以到达  $v$ ， $v$  可以到达  $u$ ，那么  $PATH_{u,v}$  中的所有点都可以到达  $u$ 。
- $u$  可以到达  $PATH_{u,v}$  中的所有点。
- 根据推论 2 可得该推论正确。

进一步可得 SCC 的在树上的连续性。



## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强连通分量缩点

## 4. 割点与点双连通分量

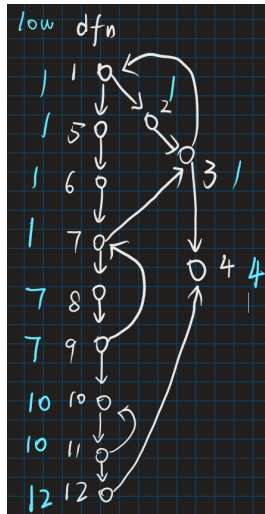
## 5. 割边与边双连通分量

在 Tarjan 算法中，引入了一个概念，叫做“回溯值”，一般用  $low$  数组来表示。

## 定义 2

$low_x$  定义为以下结点的时间戳的最小值：

- 在  $x$  的子树中的点。
- 从  $x$  的子树中的任意点经过一条非树边所能到达的，且从该点也可以回到  $x$  的点。

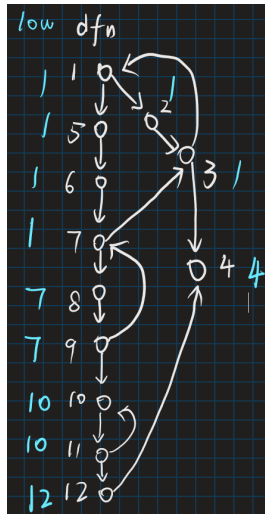


在 Tarjan 算法中，引入了一个概念，叫做“回溯值”，一般用  $low$  数组来表示。

## 定义 2

$low_x$  定义为以下结点的时间戳的最小值：

- 在  $x$  的子树中的点。
- 从  $x$  的子树中的任意点经过一条非树边所能到达的，且从该点也可以回到  $x$  的点。



## 定义 3

定义强联通分量的根节点为该强联通分量中时间戳最小的点。

在 Tarjan 中，单独的  $low$  没有意义， $low$  与  $dfn$  的比较才有意义：

## 推论 4

- $low_x < dfn_x$  等价于  $SCC_x$  的根节点的时间戳小于  $x$ 。
- $low_x = dfn_x$  等价于  $x$  为  $SCC_x$  的根节点。
- 不存在  $low_x > dfn_x$  的情况。



## 定义 3

定义强联通分量的根节点为该强联通分量中时间戳最小的点。

在 Tarjan 中，单独的  $low$  没有意义， $low$  与  $dfn$  的比较才有意义：

## 推论 4

- $low_x < dfn_x$  等价于  $SCC_x$  的根节点的时间戳小于  $x$ 。
- $low_x = dfn_x$  等价于  $x$  为  $SCC_x$  的根节点。
- 不存在  $low_x > dfn_x$  的情况。

## 推论 5

在 DFS 树上, 设  $v$  为  $u$  子树中的一点, 那么:

$low_u = dfn_u$ , 且对于  $\forall x \in PATH_{u,v}, x \neq u$ , 都有  $low_x < dfn_x$

等价于

对于  $\forall x \in PATH_{u,v}$ , 都有  $x \in SCC_u$ , 且  $u$  为  $SCC_u$  的根节点

- 右推左:  $u$  是根节点, 所以  $low_u = dfn_u$ , 根据推论 4, 可得成立。
- 左推右:  $low_u = dfn_u$ , 所以  $u$  是根节点, 根据推论 4 利用反证法, 可得成立。

通俗来讲, 就是在树的一条链上, 两个  $low = dfn$  的节点之间的所有  $low < dfn$  的点都在深度较浅的那个  $low = dfn$  的点的强连通分量中。

## 推论 5

在 DFS 树上, 设  $v$  为  $u$  子树中的一点, 那么:

$low_u = dfn_u$ , 且对于  $\forall x \in PATH_{u,v}, x \neq u$ , 都有  $low_x < dfn_x$

等价于

对于  $\forall x \in PATH_{u,v}$ , 都有  $x \in SCC_u$ , 且  $u$  为  $SCC_u$  的根节点

- 右推左:  $u$  是根节点, 所以  $low_u = dfn_u$ , 根据推论 4, 可得成立。
- 左推右:  $low_u = dfn_u$ , 所以  $u$  是根节点, 根据推论 4 利用反证法, 可得成立。

通俗来讲, 就是在树的一条链上, 两个  $low = dfn$  的节点之间的所有  $low < dfn$  的点都在深度较浅的那个  $low = dfn$  的点的强连通分量中。

接下来就是如何正确求出  $low$  数组了。重点在横向边：

其实“从  $x$  的子树中的任意点经过一条非树边所能到达的，且从该点也可以回到  $x$  的点。”这句话就说明  $x$  和该点在一个强连通分量中。

## 推论 6

对于横向边  $(u, v)$ ，如果  $\exists x \in PATH_{v, lca(u, v)}, x \neq lca(u, v)$ ，使得  $low_x = dfn_x$ ，那么  $u$  和  $v$  就不在一个强连通分量中。

根据推论 5 可得。

接下来就是如何正确求出  $low$  数组了。重点在横向边：

其实“从  $x$  的子树中的任意点经过一条非树边所能到达的，且从该点也可以回到  $x$  的点。”这句话就说明  $x$  和该点在一个强连通分量中。

## 推论 6

对于横向边  $(u, v)$ ，如果  $\exists x \in PATH_{v, lca(u, v)}, x \neq lca(u, v)$ ，使得  $low_x = dfn_x$ ，那么  $u$  和  $v$  就不在一个强连通分量中。

根据推论 5 可得。

接下来就是如何正确求出  $low$  数组了。重点在横向边：

其实“从  $x$  的子树中的任意点经过一条非树边所能到达的，且从该点也可以回到  $x$  的点。”这句话就说明  $x$  和该点在一个强连通分量中。

## 推论 6

对于横向边  $(u, v)$ ，如果  $\exists x \in PATH_{v, lca(u, v)}, x \neq lca(u, v)$ ，使得  $low_x = dfn_x$ ，那么  $u$  和  $v$  就不在一个强连通分量中。

根据推论 5 可得。

根据上面的推论，我们就能求出强联通分量了！我们先写一个朴素的代码：

```
int cnt, top, scnt, dfn[MAXN], low[MAXN], stk[MAXN];
vector<int> scc[MAXN];
bool instk[MAXN], inscc[MAXN];
void findscc(int x, const vector<int> &vec) {
    if(inscc[x]) return ;
    vec.push_back(x), inscc[x] = true;
    for(int y : son[x]) findscc(y);
}
void faketarjan(int x) {
    dfn[x] = low[x] = ++cnt, stk[++top] = x, instk[x] = true;
    for(int y : son[x]) {
        if(!dfn[y]) faketarjan(y), low[x] = min(low[x], low[y]);
        else if(instk[y]) low[x] = min(low[x], dfn[y]);
        else if(dfn[y] < dfn[x] && !inscc[y]) low[x] = min(low[x], dfn[y]);
    }
    instk[x] = false, --top;
    if(low[x] == dfn[x]) findscc(x, scc[++scnt]);
}
```

在 Tarjan 中，维护了一个栈，与前面用到的栈不同，这个栈是这样维护的：

- 遍历到  $x$  时将  $x$  入栈。
- 在遍历完与  $x$  相连的所有点之后，如果  $dfn_x = low_x$ ，那么不断弹栈，直到将  $x$  弹出。
- 弹出的所有点就在一个强联通分量中，搜索到  $x$  时在栈中的点就都能回到  $x$ 。



在 Tarjan 中，维护了一个栈，与前面用到的栈不同，这个栈是这样维护的：

- 遍历到  $x$  时将  $x$  入栈。
- 在遍历完与  $x$  相连的所有点之后，如果  $dfn_x = low_x$ ，那么不断弹栈，直到将  $x$  弹出。
- 弹出的所有点就在一个强联通分量中，搜索到  $x$  时在栈中的点就都能回到  $x$ 。

在 Tarjan 中，维护了一个栈，与前面用到的栈不同，这个栈是这样维护的：

- 遍历到  $x$  时将  $x$  入栈。
- 在遍历完与  $x$  相连的所有点之后，如果  $dfn_x = low_x$ ，那么不断弹栈，直到将  $x$  弹出。
- 弹出的所有点就在一个强联通分量中，搜索到  $x$  时在栈中的点就都能回到  $x$ 。

在 Tarjan 中，维护了一个栈，与前面用到的栈不同，这个栈是这样维护的：

- 遍历到  $x$  时将  $x$  入栈。
- 在遍历完与  $x$  相连的所有点之后，如果  $dfn_x = low_x$ ，那么不断弹栈，直到将  $x$  弹出。
- 弹出的所有点就在一个强连通分量中，搜索到  $x$  时在栈中的点就都能回到  $x$ 。

在 Tarjan 中，维护了一个栈，与前面用到的栈不同，这个栈是这样维护的：

- 遍历到  $x$  时将  $x$  入栈。
- 在遍历完与  $x$  相连的所有点之后，如果  $dfn_x = low_x$ ，那么不断弹栈，直到将  $x$  弹出。
- 弹出的所有点就在一个强联通分量中，搜索到  $x$  时在栈中的点就都能回到  $x$ 。

除 Tarjan 以外，还有一些其他的算法可以求强联通分量，如 Kosaraju 算法和 Garbow 算法，详情见OIWiKi 强联通分量板块

```
int cnt, top, scnt, dfn[MAXN], low[MAXN], stk[MAXN];
vector<int> scc[MAXN];
bool instk[MAXN];
void tarjan(int x) {
    dfn[x] = low[x] = ++cnt, stk[++top] = x, instk[x] = true;
    for(int y : son[x]) {
        if(!dfn[y]) tarjan(y), low[x] = min(low[x], low[y]);
        else if(instk[y]) low[x] = min(low[x], dfn[y]);
    }
    if(dfn[x] == low[x]) { // 弹栈
        ++scnt;
        while(stk[top] != x) scc[scnt].push_back(stk[top]), instk[stk[top--]] = false;
        scc[scnt].push_back(x), instk[x] = false, --top;
    }
}
```

## 1. 基本概念

## 2. 前置内容

### 2.1. DFS 与 DFS 树

### 2.2. 边的分类

## 3. 强连通分量

### 3.1. 前置推论

### 3.2. Tarjan 算法

### 3.3. 强联通分量缩点

## 4. 割点与点双连通分量

## 5. 割边与边双连通分量

## 缩点

在某些题中，一个强联通分量可以看做一个超级点，将一个强联通分量缩成一个点后，整张图就变成了一张 DAG，在这张 DAG 上可以再跑一些其他东西。

## 例题（洛谷 P3387）

给定一个  $n$  个点  $m$  条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。

## 缩点

在某些题中，一个强联通分量可以看做一个超级点，将一个强联通分量缩成一个点后，整张图就变成了一张 DAG，在这张 DAG 上可以再跑一些其他东西。

## 例题（洛谷 P3387）

给定一个  $n$  个点  $m$  条边有向图，每个点有一个权值，求一条路径，使路径经过的点权值之和最大。你只需要求出这个权值和。



```
#include <bits/stdc++.h>
#define fu(i, l, r) for(int i = (l); i <= (r); ++i)
#define fd(i, r, l) for(int i = (r); i >= (l); --i)
#define fv(i, vec) for(auto i : vec)
using namespace std;
const int MAXN = 1e5 + 50;
int n, m, a[MAXN], b[MAXN];
int cnt, top, scnt, dfn[MAXN], low[MAXN], stk[MAXN];
int bel[MAXN], d[MAXN], f[MAXN];
vector<int> son[MAXN], son2[MAXN];
bool instk[MAXN], flag[MAXN];

void tarjan(int x) {
    dfn[x] = low[x] = ++cnt, stk[++top] = x, instk[x] = true;
    fv(y, son[x]) {
        if(!dfn[y]) tarjan(y), low[x] = min(low[x], low[y]);
        else if(instk[y]) low[x] = min(low[x], dfn[y]);
    }
    if(dfn[x] == low[x]) { // 弹栈
        ++scnt;
        while(stk[top] != x) {
            bel[stk[top]] = scnt, instk[stk[top]] = false;
            b[scnt] += a[stk[top--]];
        }
        bel[x] = scnt, instk[x] = false, b[scnt] += a[x], --top;
    }
}
```

```
int topsort() {
    queue<int> q; int ans = 0;
    fu(i, 1, scnt) if(!d[i]) q.push(i), f[i] = b[i];
    while(!q.empty()) {
        int x = q.front(); q.pop();
        ans = max(ans, f[x]);
        fv(y, son2[x]) {
            --d[y], f[y] = max(f[y], f[x] + b[y]);
            if(!d[y]) q.push(y);
        }
    }
    return ans;
}

int main() {
    ios::sync_with_stdio(false), cin.tie(nullptr), cout.tie(
        nullptr);
    cin >> n >> m;
    fu(i, 1, n) cin >> a[i];
    int u, v; fu(i, 1, m) cin >> u >> v, son[u].push_back(v);
    fu(i, 1, n) if(!dfn[i]) tarjan(i);
    fu(i, 1, n) fv(j, son[i]) if(bel[i] != bel[j]) son2[bel[i]].
        push_back(bel[j]), ++d[bel[j]];
    cout << topsort() << '\n';
    return 0;
}
```

1. 基本概念
2. 前置内容
  - 2.1. DFS 与 DFS 树
  - 2.2. 边的分类
3. 强连通分量
  - 3.1. 前置推论
  - 3.2. Tarjan 算法
  - 3.3. 强连通分量缩点
4. 割点与点双连通分量
5. 割边与边双连通分量

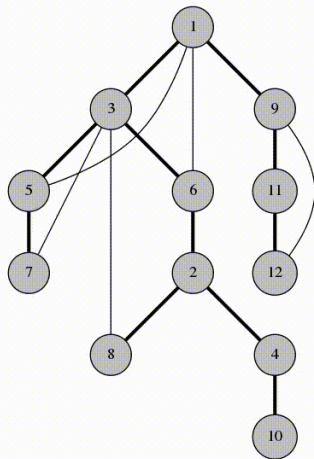
首先明确，强连通分量是有向图中的概念，割点和割边是无向图中的概念。  
对于一个无向图，如果把一个点删除后这个图的极大连通分量数增加了，那么这个点就是这个图的割点。

对于一个点  $u$ ，如果  $u$  是整个搜索树的根节点，且他不止一个儿子节点，那么他是一个割点。  
如果  $u$  不是搜索树的根节点，那么当  $low[v] \geq dfn[u]$  时，这是一个割点

点双联通分量的定义：没有割点的极大联通子图为点双连通分量。

注意，一个割点一定属于多个点双连通分量

点双联通分量的求解：当我们发现节点  $u$  为割点时，将所有在  $u$  子树内，并且在栈中的点弹栈，注意  $u$  不弹栈。



1. 基本概念
2. 前置内容
  - 2.1. DFS 与 DFS 树
  - 2.2. 边的分类
3. 强连通分量
  - 3.1. 前置推论
  - 3.2. Tarjan 算法
  - 3.3. 强联通分量缩点
4. 割点与点双连通分量
5. 割边与边双连通分量

对于一个无向图，如果把一个边删除后这个图的极大连通分量数增加了，那么这个点就是这个图的割边。  
割边也被称作桥



对于一条边，在搜索树上的顺序是从  $u$  走到  $v$ ，如果  $low[v] > dfn[u]$ ，那么这条边是割边。



边双联通分量的定义：没有割边的极大联通子图为边双连通分量。有两种求解方案



先将图中所有的桥求出。  
对于某个点开始深搜。  
深搜时不能走桥，所能搜到的所有点就是一个边双。



与求强连通分量完全一样。

当  $low[v] = dfn[u]$  时开始弹栈，直到弹出  $u$  为止。