LUT-ICPC Day 2 树形结构、图论基础

syh.hs

兰州理工大学

August 5, 2022

Contents

不严谨的定义

图论由点集和边集构成，其中点集非空

边分为有向边和无向边

一个好用的作图工具

度数表示与该节点关联的边数
在有向图里对应的有入读 和出度

不严谨的定义

- 自环
- 重边
- 补图，针对无向图
- 反图，针对有向图
- 菊花图，常用来卡解法
- 简单路径
- 连通 | 连通图 | 连通分量

不严谨的定义

有根树 | 无根树
父亲 | 祖先 | 子节点
节点深度 | 树的高度
完全二叉树 | 满二叉树
前序 | 中序 | 后序遍历

存图

直接存边

```
struct Edge {
  int u, v, w;
  bool operator<(const Edge &a) const {
    return w<a.w;
  }
};
```

存图

直接存边

```
struct Edge {
  int u, v, w;
  bool operator<(const Edge &a) const {
    return w<a.w;
  }
};
```

邻接矩阵

不太用
能直接排除重边的影响，别的没啥

存图

邻接表

```
int n, m;
cin >> n >> m;
std::vector<std::vector<int>>adj(n + 1);
std::bitset<100010>vis; vis.reset();
for (int i=1; i<=m; i++) {
  int u, v; std::cin>>u>>v;
  adj[u].push_back(v);
}
function<void(int)> dfs=[&](int u)->void {
  vis[u]==1;
  for (auto it:adj[u]) if (vis[it]==0) dfs(dfs, it);
};
```

存图

链式前向星

```
const int N=1e5+5;
const int M=1e6+5;
int tot, nxt[M], to[M], head[N];
void add(int u, int v) {
  to[++tot]=v, nxt[tot]=head[u], head[u]=tot;
}
int main() {
  int u;
  for (int i=head[u]; i; i=nxt[i]) {
    int y=to[i];
    /* code */
  }
  return 0;
}
```

存图

根据对应的情况来存图，例如用 Kruskal 找 MST，则边要按权值排序，直接存边

还有具体对应做法，对于树只存每个节点的父节点等等

遍历一个图

深度优先搜索 (DFS):
深搜树的时候先访问更深的节点，然后再访问同深度的节点，以此递归，对于图也是如此，从源点开始，有访问的节点之后继续递归，然后回溯时访问同一级节点

广搜 (BFS)
每个访问的点入队，出队的时候将所有能到的点入队

<- .... <-

---

dfs 序

正常 dfs(深度优先搜索) 遍历整个图
每次访问一个新的节点，标记顺序
以邻接表为例

```
vector<vector<int>> adj(n + 1);
vector<int> dfn(n + 1), vis(n + 1, 0);
function<void(int)> dfs = [&](int u) {
  static int dfc = 0;
  vis[u] = 1, dfn[u] = ++dfc;
  for (int v : adj[u])
    if (vis[v] == 0)
      dfs(v);
};
```

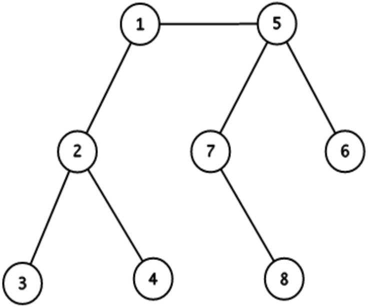
欧拉序

有两种形式

形式 1：在深搜的过程中，出入各统计一次

```
vector<vector<int>> adj(n + 1);
vector<int> euler;
function<void(int,int)> dfs = [&](int u, int fa) {
  euler.push_back(u);
  for (int v : adj[u]) {
    if (fa == v) continue;
    dfs(v, u);
  }
  euler.push_back(u);
};
```

欧拉序列



欧拉序列

形式 1:

从 1 开始 dfs 的欧拉序就是 1 2 3 3 4 4 2 5 6 6 7 8 8 7 5 1

入加出减，前缀和维护，快速求子树的点权和

欧拉序列

形式 1:

从 1 开始 dfs 的欧拉序就是 1 2 3 3 4 4 2 5 6 6 7 8 8 7 5 1

入加出减，前缀和维护，快速求子树的点权和

还有很多变形，自己可以操♂作

欧拉序列

形式 2:
每经过一个点，就加入到序列中，回溯的过程也是

```
vector<int> dep(n + 1, 0), pre(n + 1, 0), suf(n + 1, 0);
vector<int> euler;

auto dfs = [&](auto self, int u, int fa, int _dep) {
  euler.push_back(u);
  pre[u] = euler.size() - 1;
  dep[u] = _dep;
  for (int v : adj[u]) {
    if (v == fa) continue;
    self(self, v, u, _dep + 1);
    euler.push_back(u); // 回来的时候还要经过
  }
  suf[u] = euler.size() - 1; // 返回时记录靠后的位置
};
```

欧拉序列

按照上图的例子就是
1 2 3 2 4 2 1 5 6 5 7 8 5 1

共经过 2n-1 个点，因为经过了 2(n-1) 次边

欧拉序列

按照上图的例子就是
1 2 3 2 4 2 1 5 6 5 7 8 5 1

共经过 2n-1 个点，因为经过了 2(n-1) 次边

之后会用到

拓扑排序

定义

拓扑排序要求对给定的一张图进行节点顺序排序，保证对于所有有向边 (u,v)，都有 u 的顺序在 v 的顺序之前

拓扑排序

定义

拓扑排序要求对给定的一张图进行节点顺序排序，保证对于所有有向边 (u,v)，都有 u 的顺序在 v 的顺序之前

solution

动态维护一个入度为 0 的节点集合，这些就是当前能够排到顺序里的候选项

拓扑排序

定义

拓扑排序要求对给定的一张图进行节点顺序排序，保证对于所有有向边 (u,v)，都有 u 的顺序在 v 的顺序之前

solution

动态维护一个入度为 0 的节点集合，这些就是当前能够排到顺序里的候选项

至于字典序最大、最小那就是用什么维护的问题了

题目

洛谷 P1347 排序
2015-2016 NEERC, Northern Subregional Contest G-Graph

Contents

最近公共祖先 - LCA

模板题

给定一颗 n 个点的树，以及 q 个询问

每个询问给定两个点 (u,v)，要求出深度最大的点，既是 u 的祖先，也是 v 的祖先，记为 lca(u,v)

solution-倍增上跳

$\mathtt{dp[i][j]}$:i 的 2^j 祖先是谁

$\mathtt{dp[i][j+1]} = \mathtt{dp[dp[i][j]][j]}$

利用倍增性质在 ($\log_2$ 高度) 的时间内处理完

求 LCA 就一起跳到同一个深度，然后再往上跳即可

复杂度?

solution-欧拉序 +RMQ

有欧拉序列之后，我们可以把树上问题转化成区间问题

对于两个节点 u,v，我们从 $\mathtt{min}\{\mathtt{pos}[u], \mathtt{pos}[v]\}$ 走到 $\mathtt{max}\{\mathtt{pos}[u], \mathtt{pos}[v]\}$ 的过程，不会经过深度比 LCA(u,v) 更小的节点

solution-欧拉序 +RMQ

有欧拉序列之后，我们可以把树上问题转化成区间问题

对于两个节点 u,v，我们从 $\mathtt{min}\{\mathtt{pos}[u], \mathtt{pos}[v]\}$ 走到 $\mathtt{max}\{\mathtt{pos}[u], \mathtt{pos}[v]\}$ 的过程，不会经过深度比 LCA(u,v) 更小的节点

那么就变成了区间求最小深度，把序列对应的深度数组维护出来，做 RMQ，ST 表可以解决这类静态问题

代码
复杂度?

还有四毛子，加减 1 RMQ 等特殊方法来做

Tarjan 离线算法

离线？给出所有询问之后再输出答案也不迟

我们存储了所有的询问，那么从底向上的考虑问题，假设当前在一个最小的子树，我们发现有一组询问 $\mathtt{lca}(u, v)$，那么答案就是当前子树的根，处理完当前节点的所有子树之后，再往上走，依次处理涉及询问的点，答案始终是当前所在子树的根

代码
复杂度?

对应的就有强制在线

CF191C

给定一颗树，有 m 次操作，每次操作给定 (u,v)，路径上的长度都加 1

最后输出边权

Solution

简单的树上差分 +LCA

把边权标在深度更大的点上，然后两端 +1，lca-2，dfs 从底向上做一遍前缀和即可

题目

模板

可以用树剖、LCT 牛刀杀鸡

[ZJOI2012] 灾难

CF805 G2

LCA 的题还有很多，一般是结合别的操作，例如树上差分、DP

别的

其实还有很多内容,，但是我怕讲不完了

树的直径，直径数量，重心

难一点、复杂一点像树剖、LCT 这种，还是要靠自己去看..

Contents

定义

P3371 【模板】单源最短路径（弱化版）
给定一个有向图，求出从某一点出发到所有点的最短路长度

Bellman-Ford

$\texttt{dis}[\texttt{i}]$ 表示源点 s 到点 i 的最短距离，初值统一设置为 ∞

$\texttt{dis}[\texttt{s}] = \texttt{0}$

松弛操作：对于存在的边 (u, v, w) 有
$\texttt{dis}[\texttt{v}] = \texttt{min}(\texttt{dis}[\texttt{v}], \texttt{dis}[\texttt{u}] + \texttt{w})$
对于所有的边，松弛一遍的效果是什么？

Bellman-Ford

$\mathtt{dis}[i]$ 表示源点 s 到点 i 的最短距离，初值统一设置为 ∞

$\mathtt{dis}[s] = \mathtt{0}$

松弛操作：对于存在的边 (u, v, w) 有
$\mathtt{dis}[v] = \mathtt{min}(\mathtt{dis}[v], \mathtt{dis}[u] + w)$
对于所有的边，松弛一遍的效果是什么?

从源点出发的所有最短路，边数 $+1$，即新扩展了一个点

Bellman-Ford

$\texttt{dis}[\texttt{i}]$ 表示源点 s 到点 i 的最短距离，初值统一设置为 ∞

$\texttt{dis}[\texttt{s}] = \texttt{0}$

松弛操作：对于存在的边 (u, v, w) 有
$\texttt{dis}[\texttt{v}] = \texttt{min}(\texttt{dis}[\texttt{v}], \texttt{dis}[\texttt{u}] + \texttt{w})$
对于所有的边，松弛一遍的效果是什么?

从源点出发的所有最短路，边数 $+1$，即新扩展了一个点

一条最短路最多有多少个点 (最多松弛几轮)

负环

模板题

一条边权之和为负数的回路

最多松弛 n-1 轮，如果第 n 轮还能松弛下去，说明图中存在一个负环

代码

SPFA

注意到只有一处的松弛成功之后，被松弛的点才会引起下一次松弛

那么在松弛成功后，我们可以用队列维护被松弛的点，作为下一轮的候选项
这就是队列优化的 Bellman-Ford，还有别的很多优化方式..

至于 SPFA，早期国内选手习惯这么叫，对其中历史感兴趣的可以看这一篇 博客

SPFA

注意到这样一来复杂度变得玄学，但是出题人很快就有了各种卡 SPFA 以及各类优化版本的手段 使其轻松达到 $O(nm)$

卡掉一个普通 SPFA 并不难，只需要一个美丽的菊花图

当然费用流的部分问题上 SPFA 还是好用的，只不过别在正权图求最短路用就是了

Dijkstra

节点分为两个集合
一个是确定了最短路径的S 和未确定的T
初始化 $\mathtt{dis}(\mathtt{s}) = 0$ 其余的 dis 均为 ∞
然后重复以下作直到 T 为空:
1. 从T 中，选取最距离源点最近的结点，移到S 中
2. 对刚刚加入S 的结点的出边进行松弛

照这个思路暴力.. $\mathsf{O}(\mathsf{N}^2+\mathsf{M}) = \mathsf{O}(\mathsf{N}^2)$

正确性可以自己尝试用反证法证明一下

Dijkstra

Dijkstra 算法可以很好地解决无负权图的最短路径问题，但如果出现了负权边，Dijkstra 算法就会失效，例如图 10-39 中设置 A 为源点时，首先会将点 B 和点 C 的 dist 值变为–1 和 1，接着由于点 B 的 dist 值最小，因此用点 B 去更新其未访问的邻接点（虽然并没有）。在这之后点 B 标记为已访问，于是将无法被从点 C 出发的边 CB 更新，因此最后 dist[B]就是–1，但显然 A 到 B 的最短路

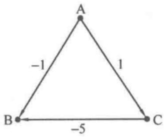


图 10-39 负权图示意图

391

算 法 笔 记

径长度应当是 A→C→B 的–4。

优化

P4779 【模板】单源最短路径（标准版）
慢的原因在第一步，怎么找到最近的节点

用二叉堆优化，维护 $\{$堆顶节点，边权$\}$ 的最小值，n 次删除堆顶操作，m 次插入 (修改) 操作，共 $O((n+m)\texttt{log}n) = O(M\texttt{log}N)$ 的复杂度

STL 的 priority_queue 因为不能修改堆中的元素，也不能删除，规模在 m，单次就是 $\texttt{log}M$，总体就是 $O(M\texttt{log}M)$

线段树的维护与手写堆类似，也是 $O(M\texttt{log}N)$

Floyd

用来求任意两个节点之间的最短路

令 $f[k][i][j]$ 表示只经过 $1 \cdots k$ 中的点，从 i 到 j 的最短路径

显然 $f[n][i][j]$ 就是所求的答案数组，我们令 $f[0][i][j]$ 为邻接矩阵，显然有

$$f[k][i][j] = \min\{f[k][i][j], f[k-1][i][k] + f[k-1][k][j]\}$$

容易发现第一维是没有用的，每一层 k 都由 k-1 推导来，因此可以省略

```
for (k = 1; k <= n; k++)
  for (x = 1; x <= n; x++)
    for (y = 1; y <= n; y++)
      f[x][y] = min(f[x][y], f[x][k] + f[k][y]);
```

Floyd

找最小环

考虑环上最大的节点编号 u
枚举 u,floyd 的时候更新 $\{f[u-1][x][y]+v(u,x)+v(u,y)\}$ 最小值即可

Floyd

找最小环

考虑环上最大的节点编号 u
枚举 u,floyd 的时候更新 $\{f[u-1][x][y]+v(u,x)+v(u,y)\}$ 最小值即可

传递闭包

求解一个图的任意两点是否连通

把边权变成 0/1，更新的时候改成或操作即可
$f[i][j] | = f[i][k] \& f[k][j]$

Floyd

找最小环

考虑环上最大的节点编号 u
枚举 u,floyd 的时候更新 $\{f[u-1][x][y]+v(u,x)+v(u,y)\}$ 最小值即可

传递闭包

求解一个图的任意两点是否连通

把边权变成 $0/1$，更新的时候改成或操作即可
$f[i][j] \mid = f[i][k] \& f[k][j]$

如果出现 $f[i][i] < 0$，那就说明出现了负环

Contents

定义

一个连通无向图的生成子图，同时要求是树。
即在图的边集中选择 n-1 条，将所有顶点连通。

最小生成树 - MST 为边权和最小的生成树

Kruskal 求最小生成树

将给定的边按照边权排序

从小到大依次考虑每条边，如果边连接的两点不在一个集合内，那么加入这条边然后合并节点所在集合，直到插入 n-1 条边为止 (或者集合大小为 n)，判断方式有很多..

可以自己用数学归纳和反证法证明，考虑当前边的集合为 F，下一条边是 e，目标 MST 是 T

Kruskal

反过来，一个正权图怎么找最大生成树?

Prim 求最小生成树

每次选择一个最近的新的节点，收录到点集里，并用相关的边更新到别的点的距离
证明同 Kruskal

拓展

还有次小生成树，严格次小生成树等等

这个可以用 LCA、倍增等方式解决，自己看 OI-wiki 就好

Kruskal 重构树

例题 货车运输

给定一个图，要求任意两点间所有路径最小权值的最大值

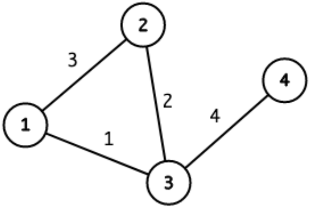
Solution - 最大生成树 + LCA

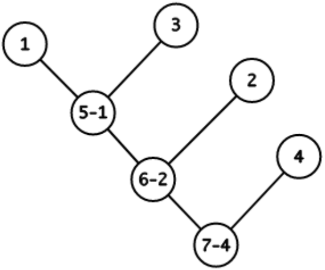
显然保存一个最大生成树，维护出来的最小值肯定最大，然后查询就好

代码

Solution - kruskal 重构树

直接看图讲构造方式吧..



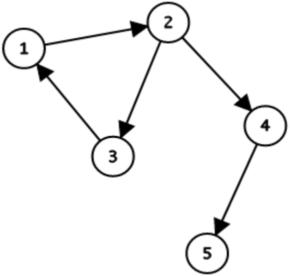


Contents

强连通

强连通分量 SCC - 极大强连通子图

对于有向图 G 强连通，有任意两点都连通
例如下图，SCC 有三组 $\{1,2,3\},\{4\},\{5\}$

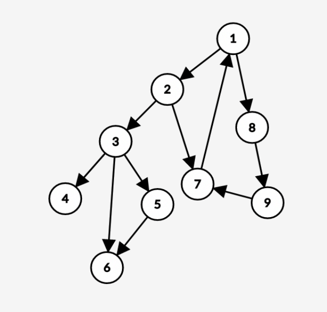


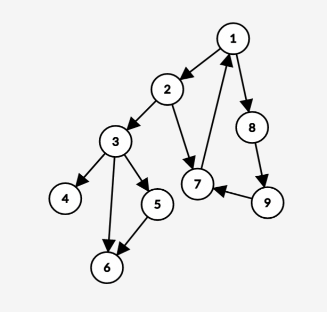
Tarjan 求 SCC

DFS 生成树

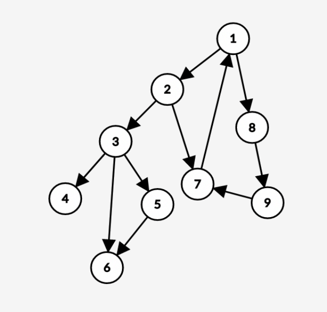
搜索过程中将图的边分为四种

1. 树边
2. 返祖边从 dfn 大的点指向 dfn 小的点
3. 横叉边指向一个 dfn 更小的点，但是不是祖先
4. 前向边遇到一个被访问过的子节点


1
2
3
4
5
6
7
8
9



如果 u 是某个强连通分量在搜索树中遇到的第一个节点，那么该 SCC 剩余节点都在以 u 为根结点的子树中



如果 u 是某个强连通分量在搜索树中遇到的第一个节点，那么该 SCC 剩余节点都在以 u 为根结点的子树中

反证法证明?

Tarjan 求 SCC

维护两个东西
$\mathtt{dfn}_u$ 这个很好理解
$\mathtt{low}_u$：设 u 为根的子树为 $\mathtt{subtree}_u$，$\mathtt{low}_u$ 是以下节点的 dfn 最小值，$\mathtt{subtree}_u$ 中的节点，从 $\mathtt{subtree}_u$ 中节点通过一条非树边能到达的节点

Tarjan 求 SCC

维护两个东西
`dfn`$_u$ 这个很好理解
`low`$_u$: 设 u 为根的子树为 `subtree`$_u$，`low`$_u$ 是以下节点的 dfn 最小值，`subtree`$_u$ 中的节点，从 `subtree`$_u$ 中节点通过一条非树边能到达的节点

感性理解: 一个 DAG(有向无环图) 从任意节点都能往更深的地方走，如果这时候给它一条回到 dfn 更小的点的边，那么这个环里面的所有点构成一个 SCC

Tarjan 求 SCC

维护两个东西
$\mathtt{dfn}_u$ 这个很好理解
$\mathtt{low}_u$: 设 u 为根的子树为 $\mathtt{subtree}_u$，$\mathtt{low}_u$ 是以下节点的 dfn 最小值，$\mathtt{subtree}_u$ 中的节点，从 $\mathtt{subtree}_u$ 中节点通过一条非树边能到达的节点

感性理解: 一个 DAG(有向无环图) 从任意节点都能往更深的地方走，如果这时候给它一条回到 dfn 更小的点的边，那么这个环里面的所有点构成一个 SCC

一定构成吗？一定有环产生？

如果有多条返回的边，自然是选择 dfn 最小的

Tarjan 求 SCC

dfs 的时候通过一个栈，将节点都保存下来，在符合判定条件的时候，栈中节点构成一个 SCC
显然在 SCC 中只有一个点的 $\mathtt{dfn}_u = \mathtt{low}_u$

对于节点 u 和它所有相邻的节点 $\{v\}$
1. v 没有被访问，(u,v) 是一条树边，那么就继续深搜，然后用 $\mathtt{low}_v$ 更新 $\mathtt{low}_u$
2. v 被访问过，并且在栈里面，那么就用 $\mathtt{dfn}_v$ 更新 $\mathtt{low}_u$
3. v 被访问过，并且不在栈里面，那么 v 所在的 SCC 已经被处理掉了，并且 v 也不能走到 u，不然在 1 的时候就会继续深搜，一并处理掉

Tarjan 求 SCC

```
void tarjan(int u) {
  static int dfc = 0;
  low[u] = dfn[u] = ++dfc;
  sta.push(u);
  in[u] = 1;
  for (int &v : adj[u]) {
    if (!dfn[v]) {
      tarjan(v);
      low[u] = min(low[u], low[v]);
    } else if (in[v])
      low[u] = min(low[u], dfn[v]);
  }
  if (dfn[u] == low[u]) {
    // 栈顶所有元素弹出，直到u为止
  }
  // 弹出u，对应操作
}
```

Kosaruju 求 SCC

对于一个图，如果正常的顺序有 $A \to B$，建立反图，也有 $A \to B$，不难发现原图上 AB 及其连边构成一个环

后序遍历一个图，即左右根的方式，拿到序列之后反过来看，发现这个顺序满足拓扑序，称其为 逆后序

Kosaruju 求 SCC

对于一个图，如果正常的顺序有 $A \to B$，建立反图，也有 $A \to B$，不难发现原图上 AB 及其连边构成一个环

后序遍历一个图，即左右根的方式，拿到序列之后反过来看，发现这个顺序满足拓扑序，称其为 逆后序

顺序考虑逆后序的点，它和能到达的点构成一个 SCC

Kosaruju 求 SCC

对于一个图，如果正常的顺序有 $A \to B$，建立反图，也有 $A \to B$，不难发现原图上 AB 及其连边构成一个环

后序遍历一个图，即左右根的方式，拿到序列之后反过来看，发现这个顺序满足拓扑序，称其为 逆后序

顺序考虑逆后序的点，它和能到达的点构成一个 SCC

思考一下横叉边，两端的点不在一个 SCC 内的情况，逆后序的顺序有影响吗?
代码

割点

定义内容源自李煜东的 WC 课件

在无向图 G 上定义：删掉某点 P 之后，G 分裂为两个及两个以上的子图，则 P 为 G 的割点

割点集合：无向连通图 G 中，如果有一个顶点集合，删除其中的点和关联的边之后，分裂为两个及以上的子图，则称该点为 G 的割点集合

点连通度：最小的割点集合的大小-K，即删除任意 K-1 个点，图都连通

割边 | BCC

定义类比割点

双连通：点/边双连通图:(点/边) 连通度大于 1 的图

无向图的极大双连通子图称为双连通分量

求割点

Tarjan 三件套的核心是什么?
dfn 与 low 数组，找到能回到的 dfn 最小的祖先

求割点

Tarjan 三件套的核心是什么?
dfn 与 low 数组，找到能回到的 dfn 最小的祖先

显然对于点 u，tarjan 过程中如果存在点 v，通过边 (u,v) 以及递归得到 $\mathtt{low}_v \geq \mathtt{dfn}_u$，那么 u 就是一个割点

求割点

Tarjan 三件套的核心是什么?
dfn 与 low 数组，找到能回到的 dfn 最小的祖先

显然对于点 u，tarjan 过程中如果存在点 v，通过边 (u,v) 以及递归得到 $\mathtt{low}_v \geq \mathtt{dfn}_u$，那么 u 就是一个割点

对于源点要特判..

求割点

Tarjan 三件套的核心是什么?
dfn 与 low 数组，找到能回到的 dfn 最小的祖先

显然对于点 u，tarjan 过程中如果存在点 v，通过边 (u,v) 以及递归得到 $\mathtt{low}_v \geq \mathtt{dfn}_u$，那么 u 就是一个割点

对于源点要特判..

对于点双连通 V-BCC，只要 (u,v) 满足了 $\mathtt{low}_v \geq \mathtt{dfn}_u$ 那么就依次取出栈中的点，和 u 构成一个点双连通分量

当然割点 u 可能属于多个 V-BCC，因此留在栈中，结束之后还剩一个 V-BCC

找割边 | 边双连通

$\mathtt{tarjan}$ 的过程记录一个父节点 fa，只要 $\mathtt{low}[v] > \mathtt{dfn}[u]$ 并且 $\mathtt{low}[u] = \mathtt{min}(\mathtt{low}[u], \mathtt{dfn}[v])$ 的时候 $u! = \mathtt{fa}[v]$ 即可

关键是验证是否只能通过树边回去

Contents

推荐阅读

洛谷日报 - 图论的小技巧以及扩展

参考资料

OI-wiki

图连通性若干拓展问题探讨-李煜东.pptx

知乎问题