宫水三叶的刷题日记

拓扑排序

Author : 宫水三叶
Date : 2021/10/07
QQ Group: 703311589
WeChat : oaoaya

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

噔噔噔噔，这是公众号「宫水三叶的刷题日记」的原创专题「图论：拓扑排序」合集。

本合集更新时间为 2021-10-07，大概每 2-4 周会集中更新一次。关注公众号，后台回复「图论：拓扑排序」即可获取最新下载链接。

💡**下面介绍使用本合集的最佳使用实践：**

学习算法：

1. 打开在线目录（Github 版 & Gitee 版）；
2. 从侧边栏的类别目录找到「图论：拓扑排序」；
3. 按照「推荐指数」从大到小进行刷题，「推荐指数」相同，则按照「难度」从易到难进行刷题‘
4. 拿到题号之后，回到本合集进行检索。

维持熟练度：

1. 按照本合集「从上往下」进行刷题。

学习过程中遇到任何困难，欢迎加入「每日一题打卡 QQ 群：703311589」进行交流 🍭🍭🍭

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

题目描述

这是 LeetCode 上的 **802. 找到最终的安全状态** ，难度为 **中等**。

Tag：「图」、「拓扑排序」

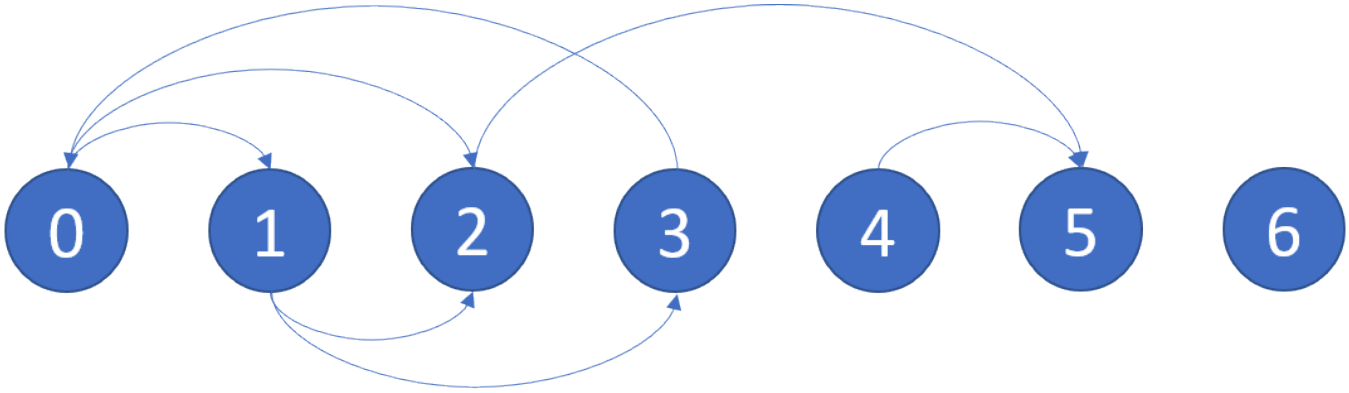
在有向图中，以某个节点为起始节点，从该点出发，每一步沿着图中的一条有向边行走。如果到达的节点是终点（即它没有连出的有向边），则停止。

对于一个起始节点，如果从该节点出发，无论每一步选择沿哪条有向边行走，最后必然在有限步内到达终点，则将该起始节点称作是 安全 的。

返回一个由图中所有安全的起始节点组成的数组作为答案。答案数组中的元素应当按 升序 排列。

该有向图有 n 个节点，按 0 到 n - 1 编号，其中 n 是 graph 的节点数。图以下述形式给出：graph[i] 是编号 j 节点的一个列表，满足 (i, j) 是图的一条有向边。

示例 1：




```
输入：graph = [[1,2],[2,3],[5],[0],[5],[],[]]

输出：[2,4,5,6]

解释：示意图如上。
```

示例 2：

```
输入：graph = [[1,2,3,4],[1,2],[3,4],[0,4],[]]

输出：[4]
```

提示：

- n == graph.length
- 1 <= n <= 10^4
- 0 <= graph[i].length <= n
- graph[i] 按严格递增顺序排列。
- 图中可能包含自环。
- 图中边的数目在范围 [1, 4 * 10^4] 内。

基本分析 & 拓扑排序

为了方便，我们令点数为 n，边数为 m。

在图论中，一个有向无环图必然存在至少一个拓扑序与之对应，反之亦然。

如果对拓扑排序不熟悉的小伙伴，可以看看 拓扑排序。

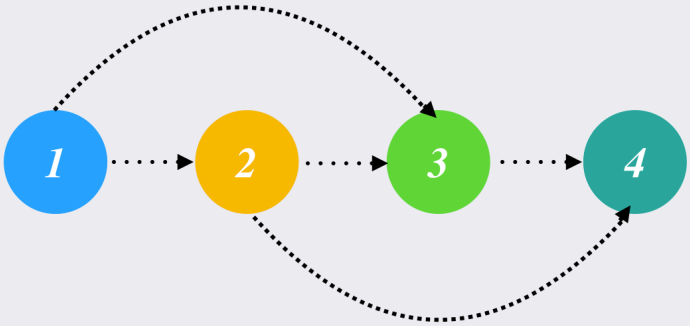
简单来说，就是将图中的所有节点展开成一维序列，对于序列中任意的节点 (u, v)，如果在序列中 u 在 v 的前面，则说明在图中存在从 u 出发达到 v 的通路，即 u 排在 v 的前面。反之亦然。

同时，我们需要知晓「入度」和「出度」的概念：

- 入度：有多少条边直接指向该节点；
- 出度：由该节点指出边的有多少条。

因此，对于有向图的拓扑排序，我们可以使用如下思路输出拓扑序（`BFS` 方式）：

1. 起始时，将所有入度为 0 的节点进行入队（入度为 0，说明没有边指向这些节点，将它们放到拓扑排序的首部，不会违反拓扑序定义）；
2. **从队列中进行节点出队操作，出队序列就是对应我们输出的拓扑序。**
   对于当前弹出的节点 x，遍历 x 的所有出度，即遍历所有由 x 直接指向的节点 y，对 y 做入度减一操作（因为 x 节点已经从队列中弹出，被添加到拓扑序中，等价于从 x 节点从有向图中被移除，相应的由 x 发出的边也应当被删除，带来的影响是与 x 相连的节点 y 的入度减一）；
3. 对 y 进行入度减一之后，检查 y 的入度是否为 0，如果为 0 则将 y 入队（当 y 的入度为 0，说明有向图中在 y 前面的所有的节点均被添加到拓扑序中，此时 y 可以作为拓扑序的某个片段的首部被添加，而不是违反拓扑序的定义）；
4. 循环流程 2、3 直到队列为空。



宫水三叶

对应的拓扑排序为 [1,2,3,4]

证明

上述 `BFS` 方法能够求得「某个有向无环图的拓扑序」的前提是：**我们必然能够找到（至少）一个「入度为 0 的点」，在起始时将其入队。**

这可以使用反证法进行证明：假设有向无环图的拓扑序不存在入度为 0 的点。

那么从图中的任意节点 x 进行出发，沿着边进行反向检索，由于不存在入度为 0 的节点，因此每个点都能够找到上一个节点。

当我们找到一条长度为 $n+1$ 的反向路径时，由于我们图中只有 n 个节点，因此必然有至少一个节点在该路径中重复出现，即该反向路径中存在环，与我们「有向无环图」的起始条件冲突。

得证「有向无环图的拓扑序」必然存在（至少）一个「入度为 0 的点」。

即按照上述的 `BFS` 方法，我们能够按照流程迭代下去，直到将有向无环图的所有节点从队列中弹出。

反之，如果一个图不是「有向无环图」的话，我们是无法将所有节点入队的，因此能够通过入队节点数量是否为 n 来判断是否为有向无环图。

反向图 + 拓扑排序

回到本题，根据题目对「安全节点」的定义，我们知道如果一个节点无法进入「环」的话则是安全的，否则是不安全的。

另外我们发现，**如果想要判断某个节点数 x 是否安全，起始时将 x 进行入队，并跑一遍拓扑排序是不足够的。**

因为我们无法事先确保 x 满足入度为 0 的要求，所以当我们处理到与 x 相连的节点 y 时，可能会存在 y 节点入度无法减到 0 的情况，即我们无法输出真实拓扑序中，从 x 节点开始到结尾的完整部分。

但是根据我们「证明」部分的启发，我们可以将所有边进行反向，这时候「入度」和「出度」翻转了。

对于那些反向图中「入度」为 0 的点集 x，其实就是原图中「出度」为 0 的节点，它们「出度」为 0，根本没指向任何节点，必然无法进入环，是安全的；同时由它们在反向图中指向的节点（在原图中**只指向**它们的节点），必然也是无法进入环的，对应到反向图中，就是那些减去 x 对应的入度之后，入度为 0 的节点。

因此整个过程就是将图进行反向，再跑一遍拓扑排序，如果某个节点出现在拓扑序列，说明其进入过队列，说明其入度为 0，其是安全的，其余节点则是在环内非安全节点。

> 另外，这里的存图方式还是使用前几天一直使用的「链式前向星」，关于几个数组的定义以及其他的存图方式，如果还是有不熟悉的小伙伴可以在 这里 查阅，本次不再赘述。

代码：

```Java
class Solution {
    int N = (int)1e4+10, M = 4 * N;
    int idx;
    int[] he = new int[N], e = new int[M], ne = new int[M];
    int[] cnts = new int[N];
    void add(int a, int b) {
        e[idx] = b;
        ne[idx] = he[a];
        he[a] = idx++;
    }
    public List<Integer> eventualSafeNodes(int[][] g) {
        int n = g.length;
        // 存反向图，并统计入度
        Arrays.fill(he, -1);
        for (int i = 0; i < n; i++) {
            for (int j : g[i]) {
                add(j, i);
                cnts[i]++;
            }
        }
        // BFS 求反向图拓扑排序
        Deque<Integer> d = new ArrayDeque<>();
        for (int i = 0; i < n; i++) {
            if (cnts[i] == 0) d.addLast(i);
        }
        while (!d.isEmpty()) {
            int poll = d.pollFirst();
            for (int i = he[poll]; i != -1; i = ne[i]) {
                int j = e[i];
                if (--cnts[j] == 0) d.addLast(j);
            }
        }
        // 遍历答案：如果某个节点出现在拓扑序列，说明其进入过队列，说明其入度为 0
        List<Integer> ans = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            if (cnts[i] == 0) ans.add(i);
        }
        return ans;
    }
}
```

- 时间复杂度：$O(n + m)$
- 空间复杂度：$O(n + m)$

> **🍭 更多精彩内容，欢迎关注：公众号 / Github / LeetCode / 知乎 **

💡**更新 Tips：本专题更新时间为 2021-10-07，大概每 2-4 周 集中更新一次。**

最新专题合集资料下载，可关注公众号「宫水三叶的刷题日记」，回台回复「图论：拓扑排序」获取下载链接。

觉得专题不错，可以请作者吃糖 🍭🍭🍭 ：