图【深度优先+三色法】【拓扑排序】

题目描述：

802. 找到最终的安全状态

难度 中等 204

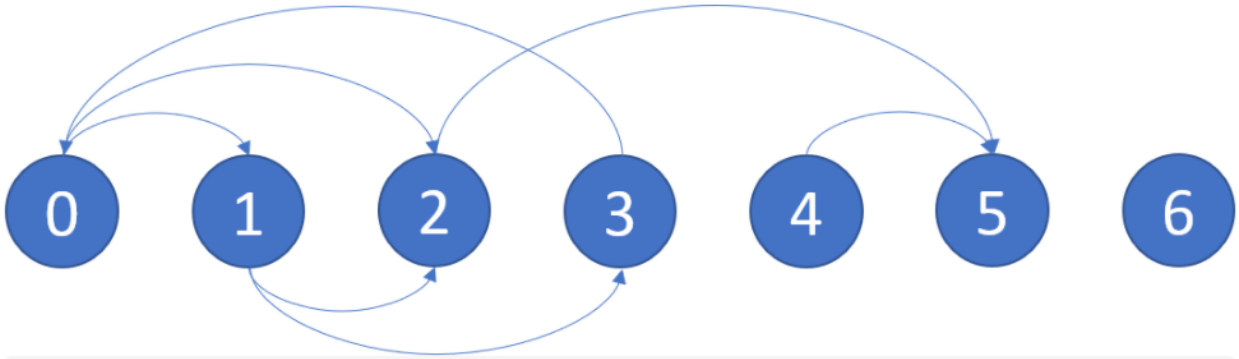
在有向图中，以某个节点为起始节点，从该点出发，每一步沿着图中的一条有向边行走。如果到达的节点是终点（即它没有连出的有向边），则停止。

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]
```

深度优先超时

```
class Solution {
private:
    vector<bool>visited;
    vector<bool>ans;
    bool dfs(vector<vector<int>>& graph,int x){
        if(!ans[x])return true;
        //存在闭环
        if(visited[x]){
            ans[x]=false;
            return true;
        }
        visited[x]=true;
        for(int i=0;i<graph[x].size();i++){
            if(dfs(graph,graph[x][i])){
                ans[x]=false;
                return true;
            }
        }
        visited[x]=false;
        return false;

    }
```

```
public:
    vector<int> eventualSafeNodes(vector<vector<int>>& graph) {
        //深度优先，超时了
        int n=graph.size();
        visited.resize(n,false);
        ans.resize(n,true);
        for(int i=0;i<n;i++){
            if(visited[i]||!ans[i])continue;
            dfs(graph,i);
        }
        vector<int>res;
        for(int i=0;i<n;i++){
            if(ans[i])res.push_back(i);
        }
        return res;
    }
};
```

深度优先+三色法

我们可以使用深度优先搜索来找环，并在深度优先搜索时，用三种颜色对节点进行标记，标记的规则如下：

- 白色（用 0 表示）：该节点尚未被访问；
- 灰色（用 1 表示）：该节点位于递归栈中，或者在某个环上；
- 黑色（用 2 表示）：该节点搜索完毕，是一个安全节点。

1. 当我们首次访问一个节点时，将其标记为灰色，并继续搜索与其相连的节点。
2. 如果在搜索过程中遇到了一个灰色节点，则说明找到了一个环，此时退出搜索，栈中的节点仍保持为灰色，这一做法可以将「找到了环」这一信息传递到栈中的所有节点上。
3. 如果搜索过程中没有遇到灰色节点，则说明没有遇到环，那么递归返回前，我们将其标记由灰色改为黑色，即表示它是一个安全的节点。

```
class Solution {
private:
    vector<int>visited;
    bool dfs(vector<vector<int>>& graph,int x){
        //存在闭环
        if(visited[x]==1){
            return false;
        }else if(visited[x]==2){
            return true;
        }
        if(visited[x]==0){
            visited[x]=1;
        }
        for(int i=0;i<graph[x].size();i++){
            if(!dfs(graph,graph[x][i])){
                return false;
            }
        }
        visited[x]=2;
        return true;

    }
public:
    vector<int> eventualSafeNodes(vector<vector<int>>& graph) {
        //深度优先，超时了
        //修改一下——深度优先+三色法
        int n=graph.size();
        //0表示未访问，1表示在栈中或在环中，2表示结点安全
        visited.resize(n,0);

        for(int i=0;i<n;i++){
            if(visited[i]==1||visited[i]==2)continue;
            dfs(graph,i);
        }
        vector<int>res;
        for(int i=0;i<n;i++){
            if(visited[i]==2)res.push_back(i);
        }
        return res;
    }
};
```

拓扑排序：

1. 将图进行一个反向
2. 将所有入度为**0**的点入队
3. 从入度为**0**的点搜索，并将搜索过程中入度降为**0**的点入队，直到队列为空

```
```

```cpp
class Solution {
public:
    vector<int> eventualSafeNodes(vector<vector<int>>& graph) {
        //玩一手拓扑排序
        //将边反向
        int n=graph.size();
        vector<vector<int>>edges(n);
        for(int i=0;i<n;i++){
            int n1=graph[i].size();
            for(int j=0;j<n1;j++){
                edges[graph[i][j]].emplace_back(i);
            }
        }
        //统计各个顶点的入度
        vector<int>indegree(n,0);
        for(int i=0;i<n;i++){
            int n1=edges[i].size();
            for(int j=0;j<n1;j++){
                ++indegree[edges[i][j]];
            }
        }
        //将入度为0的点入队列
        queue<int>que;
        vector<bool>ans(n,false);
        for(int i=0;i<n;i++){
            if(!indegree[i]){
                que.push(i);
                ans[i]=true;
            }
        }
        //开始搞事情
        while(!que.empty()){
            int d=que.front();
            que.pop();
            for(int i=0;i<edges[d].size();i++){
                int temp=edges[d][i];
                --indegree[temp];
                if(!indegree[temp]){
                    que.push(temp);
                    ans[temp]=true;
                }
            }
        }
        //寻找答案
        vector<int>res;
        for(int i=0;i<n;i++){
            if(ans[i])res.push_back(i);
        }
        return res;
```

```
        }
    };
```