ACM算法与程序设计
(七)
图论基础算法

杜育根

Ygdu@sei.ecnu.edu.cn

7. 图论基础算法

- 这节课介绍了图论中除了最短路以外的几个常见算法，包括最小生成树、拓扑排序、欧拉路、强连通 Tarjan 算法。

7.1. Kruskal 最小生成树算法

- 生成树

  已知连通图 G，图上有 n个顶点。生成树 是指图 G的一个极小（边最少）连通子图，生成树上有 n个顶点、n−1 条边，且任意两点之间都是连通的。

- 最小生成树

  已知带权连通图 G，图中有 n个顶点，每条边都有权值。我们要从图中抽出一棵生成树，使得树上所有边权之和最小，这棵生成树就叫做 最小生成树（Minimum Spanning Tree，MST）。常见的求解最小生成树的算法有 Prim 算法 和 Kruskal 算法。

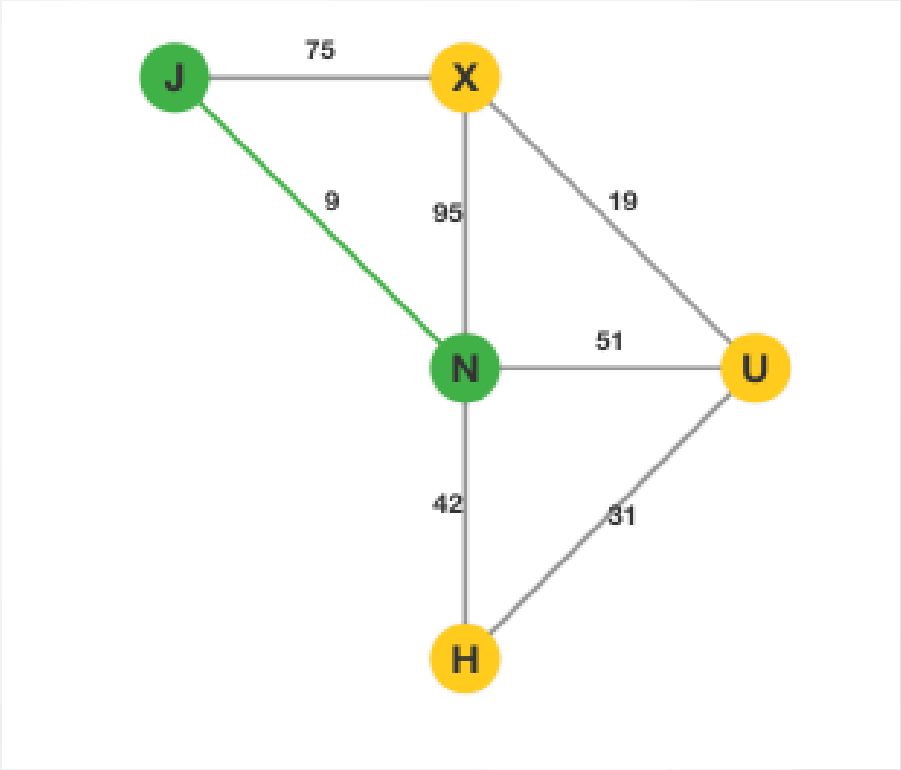
Kruskal 算法

- Kruskal 算法是最小生成树算法的一种。算法过程如下：
- 我们定义带权无向图 G的边集合为 E，接着我们再定义最小生成树的边集为T，初始集合 T=∅。
- 执行以下操作：
- 1. 首先，我们把图 G看成一个有 n 棵树的森林，图上每个顶点对应一棵树；
- 2. 接着，我们将边集 E 的每条边，按权值从小到大进行排序；
- 3. 按边权从小到大的顺序遍历每条边 e=(u,v)，我们记顶点u所在的树为Tu，顶点 v 所在的树为 Tv，如果 Tu 和 Tv 不是同一棵树，则我们将边 e 加入集合T，并将两棵树和 Tv 进行合并；否则不进行任何操作。
- 算法执行完毕后，如果集合 T 包含 n−1 条边，则 T 就代表最小生成树中的所有边。

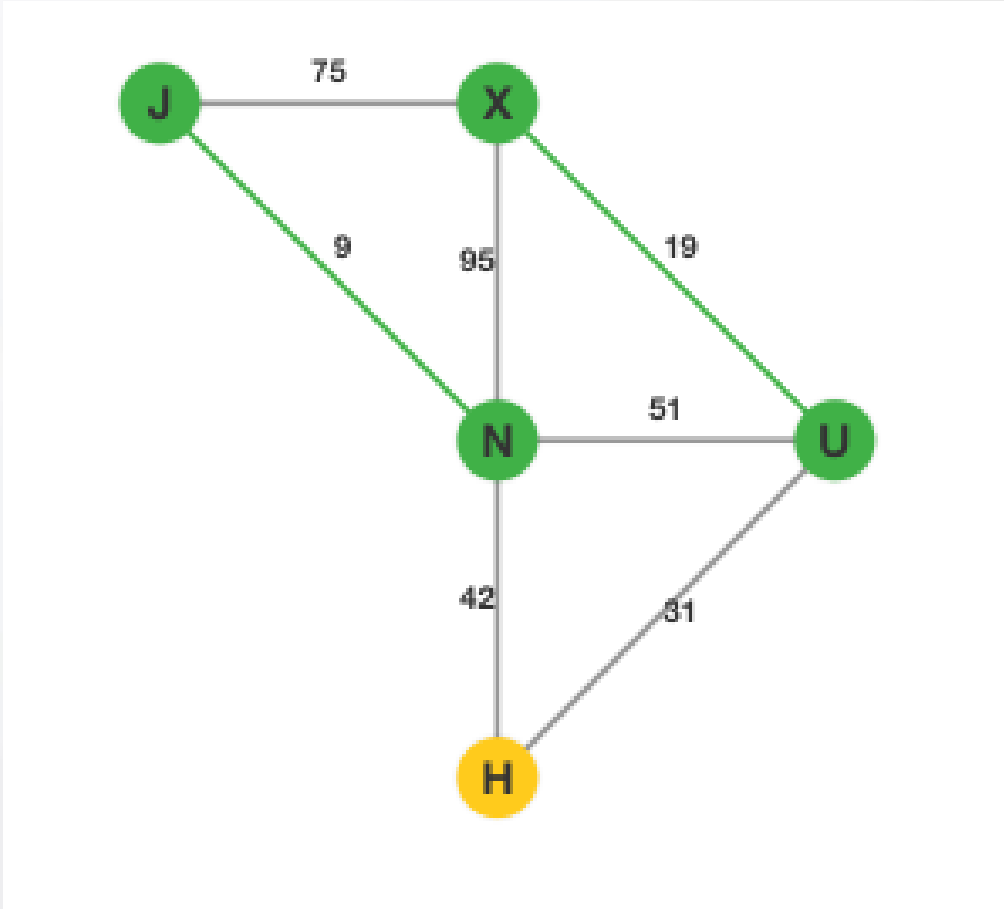
- *第3步中需要对集合进行合并操作，因此要用并查集来维护。*

举例描述

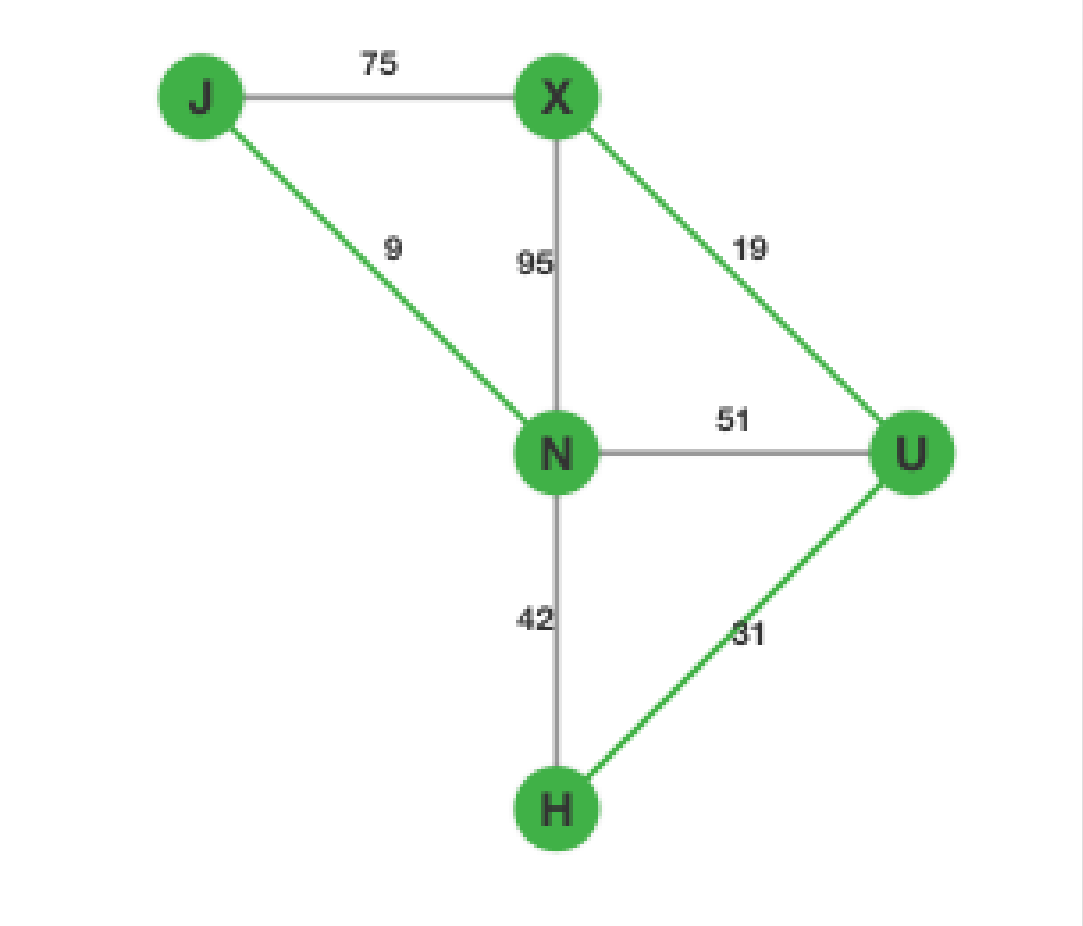
- 有这样一张图，图上有 5个顶点和 7条边。
- 第一步：将所有的边按边权从小到大排序。排序完成后，我们选择权值最小的边 JN。这样我们的图就变成了：



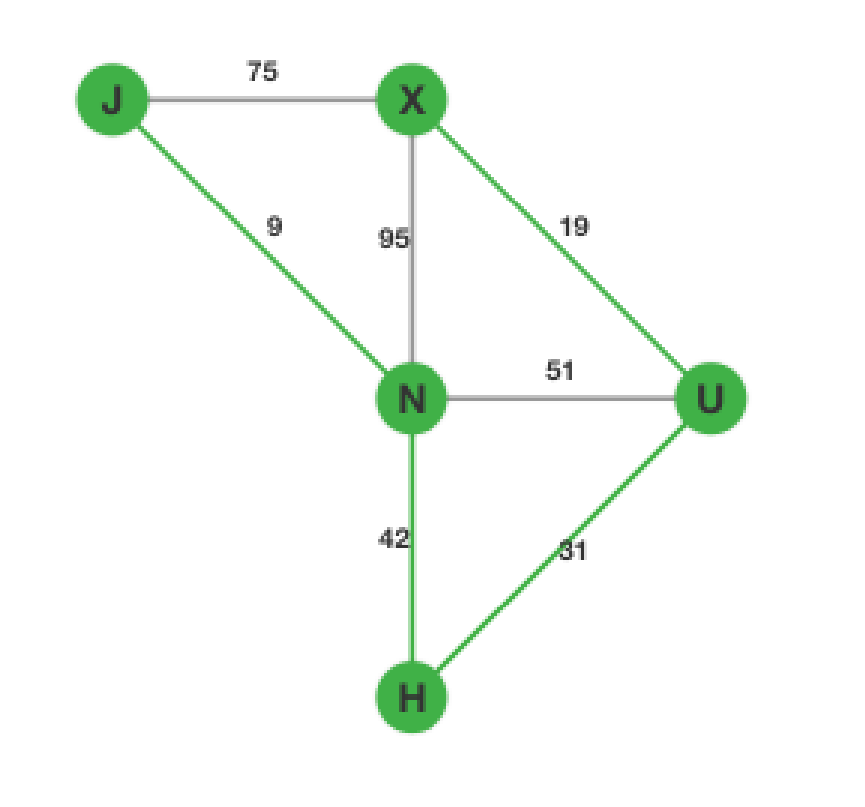
- 第二步，在剩下的边中寻找权值最小的边 UX：



- 依次类推我们找到 HU，图变成：



- 最后我们只需要再选择 HN：



- 至此所有的点都已经连通，一个最小生成树构建完成。
- Kruskal 算法的时间复杂度由排序算法决定，若采用快排则时间复杂度为 O(ElogE)。
- 总的时间复杂度为 O(ElogE+Vα(V))，其中 α(V) 可以近似地被当做常数。

C++ 示例代码如下

```cpp
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int MAX_N = 100000;  // 最大顶点数
const int MAX_M = 100000;  // 最大边数
struct edge {
    int u, v, w;
}e[MAX_M];
int fa[MAX_N], n, m;  // fa 数组记录了并查集中结点的父亲
bool cmp(edge a,edge b) {
    return a.w < b.w;
}
// 并查集相关代码
int ancestor(int x) {  // 在并查集森林中找到 x 的祖先，也是所在连
                       //通块的标识
    if(fa[x] == x) return fa[x];
    else return fa[x] = ancestor(fa[x]);
}
int same(int x, int y) {//判断两个点是否在一个连通块（集合）内
    return ancestor(x) == ancestor(y);
}
void merge(int x, int y) {  // 合并两个连通块（集合）
    int fax = ancestor(x), fay = ancestor(y);
    fa[fax] = fay;
}
int main() {
 scanf("%d%d", &n, &m);  // n 为点数，m 为边数
 for (int i = 1; i <= m; i++) {
    scanf("%d%d%d", &e[i].u, &e[i].v, &e[i].w);  // 用边集数组存
                                                 //放边，方便排序和调用
 }
 sort(e + 1, e + m + 1, cmp);  // 对边按边权进行升序排序
 for (int i = 1; i <= n; i++) {
    fa[i] = i;
 }
 int rst = n, ans = 0;  // rst 表示还剩多少个集合，ans 保存最小生
                        //成树上的总边权
 for (int i = 1; i <= m && rst > 1; i++) {
    int x = e[i].u, y = e[i].v;
    if (same(x, y)) {
        continue;  // same函数是查询两个点是否在同一集合中
    } else {
    merge(x, y);//merge 函数用来将两个点合并到同一集合中
    rst--;//每次将两个不同集合中的点合并，都将使 rst 值减 1
    ans += e[i].w;  // 这条边是最小生成树中的边，将答案加上边权
    }
 }
 printf("%d\n", ans);
 return 0;
}
```

最小生成树的应用

例题 1：最大边权最小的生成树

- 给定一个无向连通图，求出它所有生成树中最大边权最小的一棵。
- 解法：很多同学会下意识地想到二分，枚举生成树的边权上界之后判定是否存在边权不超过限制的生成树，最终得到答案。当然这么做是可行的，但实际上，你只需要使用 Kruskal 或 Prim 求出图的最小生成树就可以了。因为最小生成树中的最大边一定是所有生成树中最小的。

例题 2：次小生成树

- 给定一个无向连通图，求出它的边权总和第二小的生成树。
- 解法：枚举最小生成树上的 n条边，对于其中某条边，从图中删除它以后计算剩余图的最小生成树，一共 n−1 棵。从这 n−1 棵生成树中找出最小的一棵，就是整个图的次小生成树。

例题 3：边权极差最小生成树

- 给定一个无向连通图，求出它所有生成树中，最大边权和最小边权之差最小的生成树。
- 解法：利用例题 1 中给出的性质，按边权顺序枚举每条边，计算以该边为边权最小的最小生成树，用当前最小生成树的最大边减去最小边来更新最终答案。

习题：布设光纤

- S国有 n座基站，现在小明想给基站之间布设光纤，使得任意两座基站都是连通的，光纤传输具有传递性，即如果基站 A和基站 B之间有光纤，基站 B和基站 C之间有光纤，则基站 A和基站 C也是连通的，可以通过中间基站 B来完成传输。
- 不同的基站之间布设光纤的费用是不同的，现在小明知道了任意两座基站之间布设光纤的费用，求问如何布设，可以使得任意两座基站都是连通的，且总费用最小。
- 输入格式：第一行输入一个整数 n(2≤n≤100)，表示基站总数。
- 接下来输入 n×n的矩阵。第i行第j列的整数表示第 i座基站和第 j座基站之间布设光纤的费用 wij（0≤wij≤10,000）。
- 输出格式：输出一个整数，表示布设光纤的最小总费用，且使任意两座基站都是连通的。
- 样例输入
- 4
- 0 1 5 1
- 1 0 6 3
- 5 6 0 2
- 1 3 2 0
- 样例输出
- 4

习题：连线问题

- 小明和花椰菜君经常出难题考对方。一天，花椰菜君给小明出了这样一道难题：花椰菜君在坐标系上随机画了 N个点，然后让小明给点之间连线，要求任意两点之间都是连通的，且所连的线段长度之和最小。聪明的你快来帮小明解决一下吧。
- 输入样例：第一行输入一个整数 N(1≤N≤100)，表示花椰菜君一共画了 N个点。然后输入 N行，每行输入两个整数 x,y（0≤x,y≤1,000），代表一个点的坐标。
- 输出样例：输出一行，输出一个实数，表示所连线段长度之和的最小值（结果四舍五入保留两位小数）。
- 样例输入
- 4
- 1 1
- 2 2
- 3 3
- 3 1
- 样例输出
- 4.24

习题：穿越雷区

- 小明最近迷上了一款 RPG 游戏，这次他要去森林里的 n个宝藏点收集宝藏，编号从 1到 n。森林里有 m条道路连接宝藏点，每条道路上都有数量不等的地雷，小明想从中找出若干条道路，使得任意两个宝藏点都是连通的，这样小明都能访问到每个宝藏点了。另外，由于遇到一个地雷，小明会减少一定的血量。
- 现在小明知道了这 m条道路上的地雷数，小明希望挑选若干条道路，使得挑选出的道路，地雷数量之和尽可能小。
- 输入格式：第一行输入两个整数 n,m（1≤n≤30,000，1≤m≤50,000），表示森林里有 n个宝藏点，m条道路。两个宝藏点之间可能会有多条道路。接下来输入 m行，每行输入三个整数 x，y，z（1≤x,y≤n，0≤z≤1,000），表示 x 和 y之间有 z个地雷。
- 输出格式：输出一行，输出一个整数。表示小明挑出了若干条道路，使得任意两个宝藏点都是连通的，输出道路的地雷数量之和最小值。
- 样例输入
- 3 4
- 1 2 121
- 1 3 202
- 2 1 497
- 2 3 135
- 样例输出
- 256

习题：高速公路

- 小明所在的国家有 n个城市，现在需要在城市之间修高速公路，有 m条修路的方案，每个方案表示a，b城市之间修一条限速为 c的高速公路。小明希望从这 m个方案中选出若干方法试行，能够让 n座城市联通，并且希望所有高速公路中最高限速和最低限速的差值最小。
- 输入格式：第一行输入两个整数 n,m（2≤n≤100，1≤m≤n(n−1)/2），表示有 n个城市，m条修路方案。两个城市之间可能会有多条修路方案。
- 接下来输入 m行，每行输入三个整数 a，b，c（1≤a,b≤n，0≤c≤1,0000）。
- 输出格式：如果修路方案不能让 n个城市之间联通，输出 -1，否者输出最小的差值。
- 样例输入
- 4 4
- 1 2 2
- 2 3 4
- 1 4 1
- 3 4 2
- 样例输出
- 1

7.2. LCA 问题及其倍增解法

- 概述

LCA（Least Common Ancestors），即最近公共祖先，是这样一个问题：在有根树中，找出某两个结点 u和 v最近的公共祖先（或者说，离树根最远的公共祖先）。

- 倍增解法

倍增解法的核心是分治思想。当已知两个点在树中的深度时，先让较深的结点向上走，直到两个结点深度一样；再二分找出离他们最近的公共祖先。我们记一个结点的父结点为它的 $2^0=1$ 倍祖先，它的父结点的父结点为它的 2 倍祖先，以此类推。

倍增算法的具体流程第一步：

- 一、在树上预处理每个结点的深度和 1倍祖先，也就是每个结点的父结点。用d数组来表示每个结点的深度，fa[v][h]表示结点 v的 h倍祖先的结点编号。d数组中的元素初始为 −1。初始化完成后，fa[a][0]保存的是a的第 2^0=1 倍祖先结点，即它的父结点。时间复杂度 O(V)。
- const int MAX_N = 100000; //最大的点数
- const int MAX_M = 1000000; //最大的边数
- struct edge{
- int v,next;//边的终点，同起点的上一条边的序号
- }e[MAX_N];
- int p[MAX_N],vst[MAX_N],d[MAX_N],fa[MAX_N][20];
- //存储边的序号的头指针,标记数组,存储深度,fa[i][j]为i往上走(2^j)层的序号
- void init() {
- memset(d, -1, sizeof(d));
- }
- void dfs(int node) {
- for (int i = p[node]; i != -1; i = e[i].next) {
- if (d[e[i].v] == -1) {
- d[e[i].v] = d[node] + 1;
- fa[e[i].v][0] = node;
- dfs(e[i].v);
- }
- }
- }

倍增算法的具体流程第二步：

- 二、倍增计算各个点的 2^j祖先是谁，其中，1倍祖先就是父亲，2倍祖先是父亲的父亲，以此类推。该点2^j 祖先也就是该点 2^{j-1}祖先的 2^{j-1} 祖先。
- for (int level = 1; (1 << level) <= n; level++) {
- for (int i = 1; i <= n; i++) {
- fa[i][level] = fa[fa[i][level - 1]][level - 1]; // i 的第 2^j 祖先就是 i 的 //第 2^(j-1) 祖先的第 2^(j-1) 祖先
- }
- }

- 在这一步中，2^{level}表示的是祖先的倍数。很显然，i 的 2^j倍祖先就是i 的 2^{j-1}倍祖先的 2^{j-1}倍祖先。时间复杂度为O(VlogV)。

倍增算法的具体流程第三步：

三（查询）：首先，深度大的点往上爬，直到与深度小的点在同一层。若此时两结点相同直接返回此结点，即最近公共祖先 LCA。这意味着其中一个结点是另一个结点的祖先结点。
否则，利用倍增思想，同时让 x和 y二分地向上找，直到找到深度相等且最小的 x和 y的祖先 x′,y′，满足 x′≠y′。此时他们的父结点 fa[x′][0] 即为 x 和 y 的最近公共祖先 LCA。代码的时间复杂度为 O(logV)。

```
1.int lca(int x, int y) {
2.    int i, j;
3.      // 确保 x 的深度比 y 大
4.    if (d[x] < d[y]) {
5.        swap(x, y);
6.    }
   // 接下来，让 x 向上找，直到 x 的深度和 y 一致
7.    for (i = 0; (1 << i) <= d[x]; i++);  // 2^i > d[x]
          // 找出的 i 是最大的可能需要从 d[x] 里减去
          // 2^i，也就是，i 是满足 2^i<=d[x] 的最大值
8.    i--;
9.      for (j = i; j >= 0; j--) {
10.       if (d[x] - (1 << j) >= d[y]) {
11.           x = fa[x][j];
12.       }
13.   }
    // 若此时 x 和 y 相等，则当前的 x 就是他们的 LCA
14.   if (x == y) {
15.       return x;
16.   }
    // 否则，继续二分，找到深度最小的 x' 和 y'
17.   for (j = i; j >= 0; j--) {
18.       if (fa[x][j] != fa[y][j]) {
19.           x = fa[x][j];
20.           y = fa[y][j];
21.       }
22.   }
23.   // fa[x'][0] 就是 LCA
24.   return fa[x][0];
25. }
```

小结

- 倍增解法是 LCA 问题的在线解法，整体时间复杂度是 O(VlogV+QlogV)，其中 Q 是总查询次数。

习题：节点的最近公共祖先

- 树是一种很常见的数据结构。现在小明面临一个问题，在一个有 n个节点的树上，节点编号分别是1...n。小明想知道一些节点之间的最近公共祖先是那些节点。
- 输入格式：第一行输入一个整数 n（2≤n≤10,000），表示树上有 n个节点。
- 接下来的 n−1 行，每行输入俩个整数 a，b（1≤a,b≤n）代表节点 a，b 之间有一条 a 到 b 边，a 是 b 的父亲。
- 接下来输入一个整数 q，代表小明的 q 次提问。（1≤q≤1,000）
- 接下来的 q 行，每行输入俩个整数 c，d（1≤c,d≤n）代表询问 c，d 俩个节点的最近公共祖先。
- 输出格式：对于每次询问，输出公共祖先的节点编号，占一行。
- 样例输入
- 5
- 1 2
- 2 3
- 1 4
- 2 5
- 2
- 3 4
- 3 5
- 样例输出
- 1
- 2

习题：小明运送宝藏

- S国有 N 座城市，编号依次从 1 到 N，城市之间有 M 条双向的道路。每一条道路对每辆车都有一个最大载重量。小明意外发现了一批宝藏，精心策划了下，他计划用 Q 辆货车分头秘密的运送这批宝藏。他想知道每辆车在不超过道路最大载重量的情况下，最多能运送宝藏的重量（此处忽略货车的重量，只考虑宝藏的重量）。
- 输入格式
- 输入第一行输入两个正整数N,M(0<N<10,000,0<M<50,000)，之间用一个空格隔开，表示S国的 N 座城市和 M 条道路。
- 接下来输入 M 行，每行输入三个正整数 x、y、z，每两个整数之间用一个空格隔开，表示编号为 x 的城市和编号为 y 的城市之间有一条最大载重量为 z的道路。x不等于 y，两座城市之间可能有多条道路。
- 接下来输入一行，输入一个正整数Q(0<Q<30,000)，表示有 Q 辆货车在运送宝藏。
- 接下来输入 Q 行，每行输入两个整数 a、b，之间用一个空格隔开，表示一辆货车需要从 a 城市运输宝藏到 b 城市，注意：a 不等于 b。
- 输出格式：输出一共有 Q行，每行输出一个整数，表示对于每一辆货车，在不超过道路最大载重量的情况下，最多能运送宝藏的重量。如果货车不能到达目的地，输出 -1。

样例输入

```
5 4
1 2 4
2 3 3
3 4 5
3 1 1
2
1 3
2 4
```

样例输出

```
3
3
```

7.3. 拓扑排序

- 概述
- 对一个 有向无环图（Directed Acyclic Graph，DAG） G 进行拓扑排序，是将 G 中的所有顶点排成一个线性序列，使得图中任意一对顶点 u和v，若边 <u,v>∈E(G)，则 u在线性序列中出现在v 之前。通常，这样的线性序列称为满足 拓扑次序（Topological Order） 的序列，简称 拓扑序列。简单的说，由某个集合上的一个偏序得到该集合上的一个全序，这个操作称之为 拓扑排序。

例：先修课程的顺序

例如，对右表的一个专业方案列表，你需要根据专业方案学习列表上的必修课程和选修课程，而学习每门课程的先决条件是学习完它的全部先修课程。如学习《数据结构》课程就必须安排在它的先修课程《离散结构》之后。而学习《C 语言程序设计》课程可以随时安排，因为它是基础课程，没有先修课。一个符合要求的学习课程的顺序就是对这些课程进行拓扑排序。

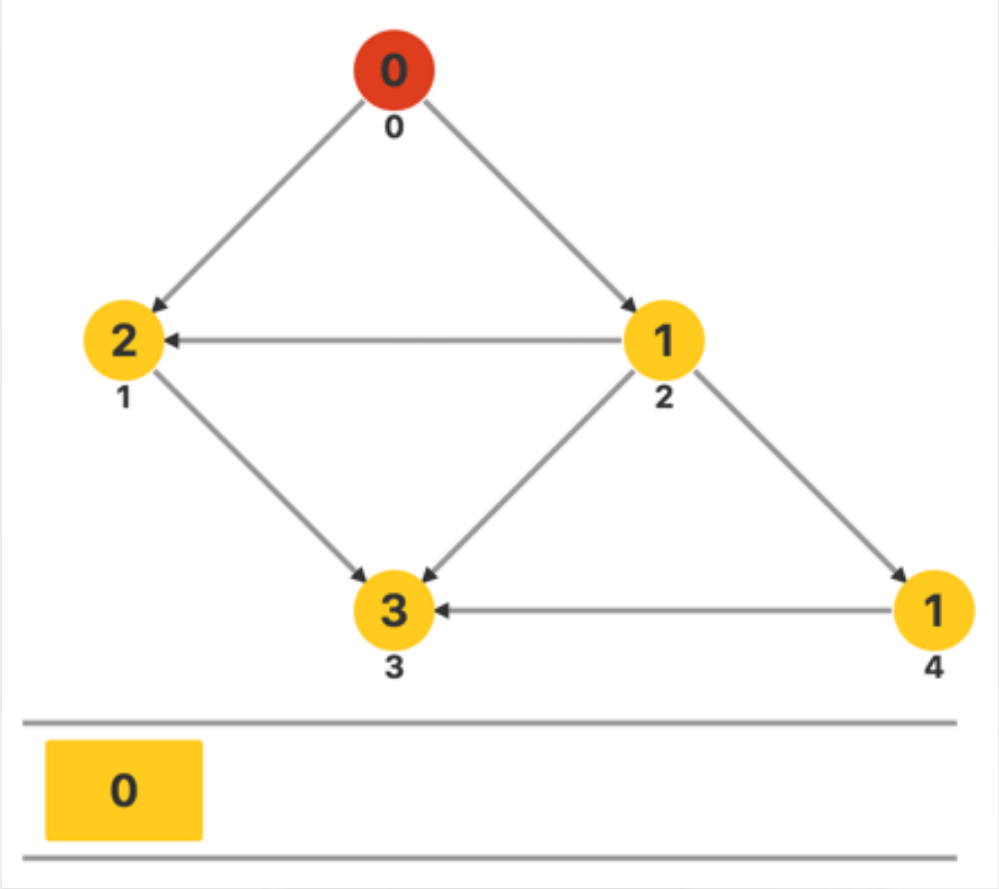
课程列表	课号	课程名称	学分	建议先修	最新版开课时间	说明
专业入门	CS 101	计算机科学导论	2		2017 春季学期	
	CS 111	C 语言程序设计	3	无	2016 冬季学期	这三门课为一个 CS 11A 系列 至少选择其中一门学习
	CS 112	C++ 程序设计	3		2016 冬季学期	
	CS 116	Java 程序设计	3		2016 冬季学期	
	CS 113	Python 程序设计	2	CS 11A	2016 冬季学期	这三门课为一个 CS 11B 系列 至少选择其中一门学习
	CS 114	Ruby 程序设计	2		2016 冬季学期	
	CS 115	JavaScript 程序设计	2		2016 冬季学期	
	CS 141	数据科学导论	2	CS 11B	2016 冬季学期	
	CS 161	离散结构	3		2017 春季学期	
专业基础	CS 200	工程实践基础	3	CS 11A	2016 冬季学期	
	CS 212	面向对象的程序设计（C++）	4		2016 冬季学期	这两门课为一个 CS 21A 系列 至少选择其中一门学习
	CS 216	面向对象的程序设计（Java）	4		2017 秋季学期	
	CS 221	操作系统入门与系统编程	4	CS 11A, CS 261	2017 春季学期	
	CS 241	数据库引论与关系数据库	4	CS 141	2017 春季学期	
	CS 251	计算机组成原理	4	CS 101	2017 秋季学期	
	CS 261CC	数据结构（C++ 版）	4	CS 11A	2016 冬季学期	不与 CS 262/263 学分同时获得
	CS 261CL	数据结构（C 语言版）	4		2017 冬季学期	

算法流程

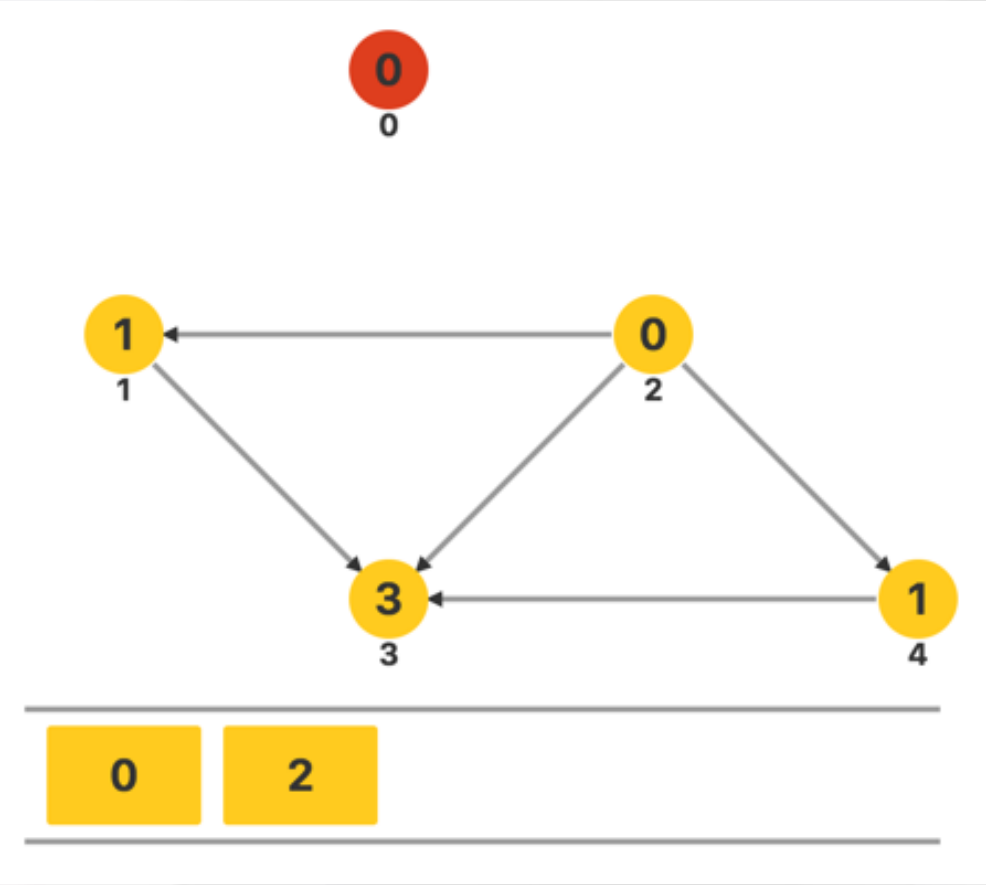
- 拓扑排序算法主要由以下两步循环执行，直到不存在入度为 0 的顶点为止。
- 1. 选择一个入度为 0 的顶点并将它输出；
- 2. 删除图中从顶点连出的所有边。
- 循环结束，若输出的顶点数小于图中的顶点数，则表示该图中存在回路，也就是无法进行拓扑排序；否则输出的顶点序列就是一个拓扑序列。
- 接下来，我们用一个例子来说明这个算法过程。

例子

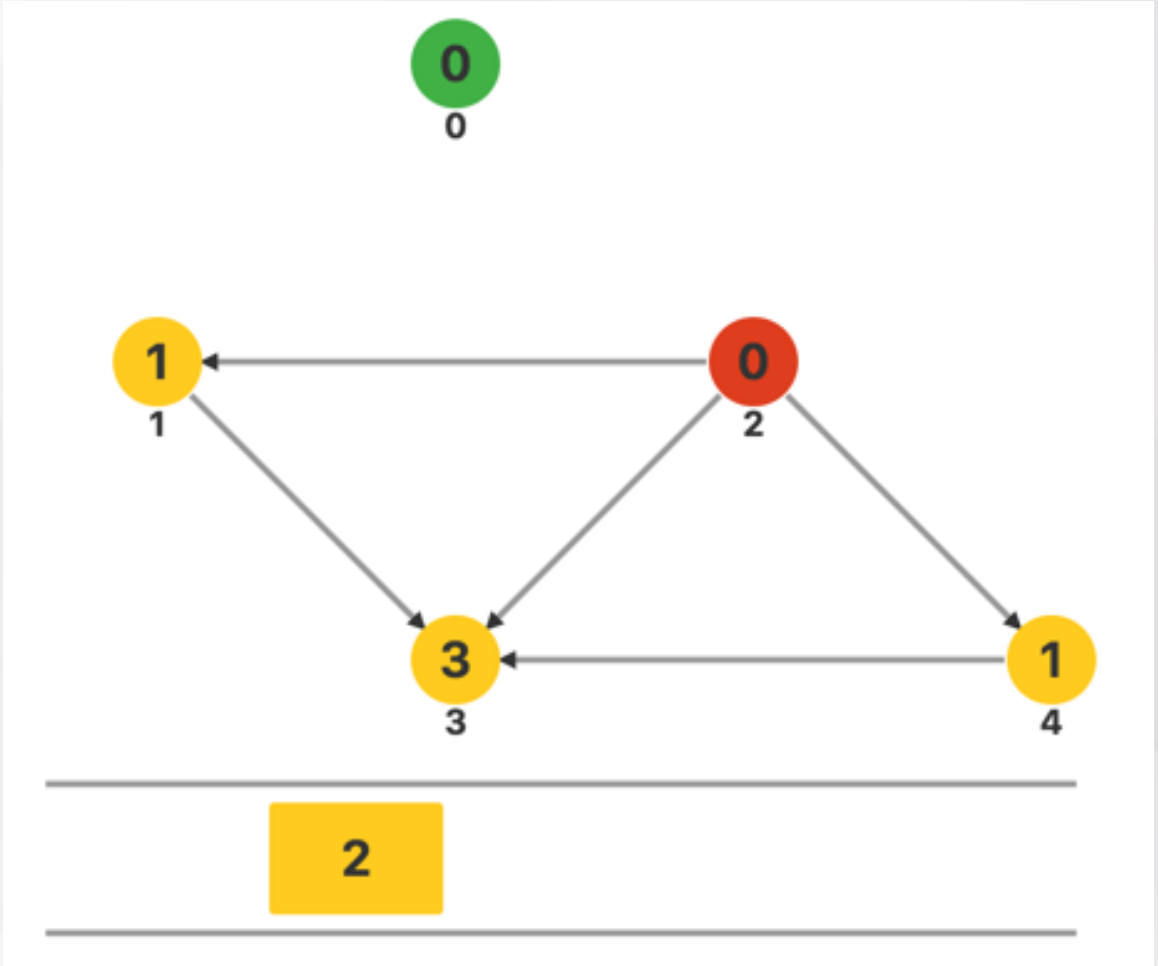
- 对于如下的图，图中圆圈内的数字表示顶点的入度，圆圈下方的数字表示顶点编号，直线表示边，直线一端的箭头表示边的方向。图的下方是一个队列，用来在拓扑排序时储存所有未处理的入度为零的顶点。
- 一、我们首先算出所有顶点的入度，并找出其中所有入度为零的顶点，发现只有 0，于是我们将 0 插入队列中。



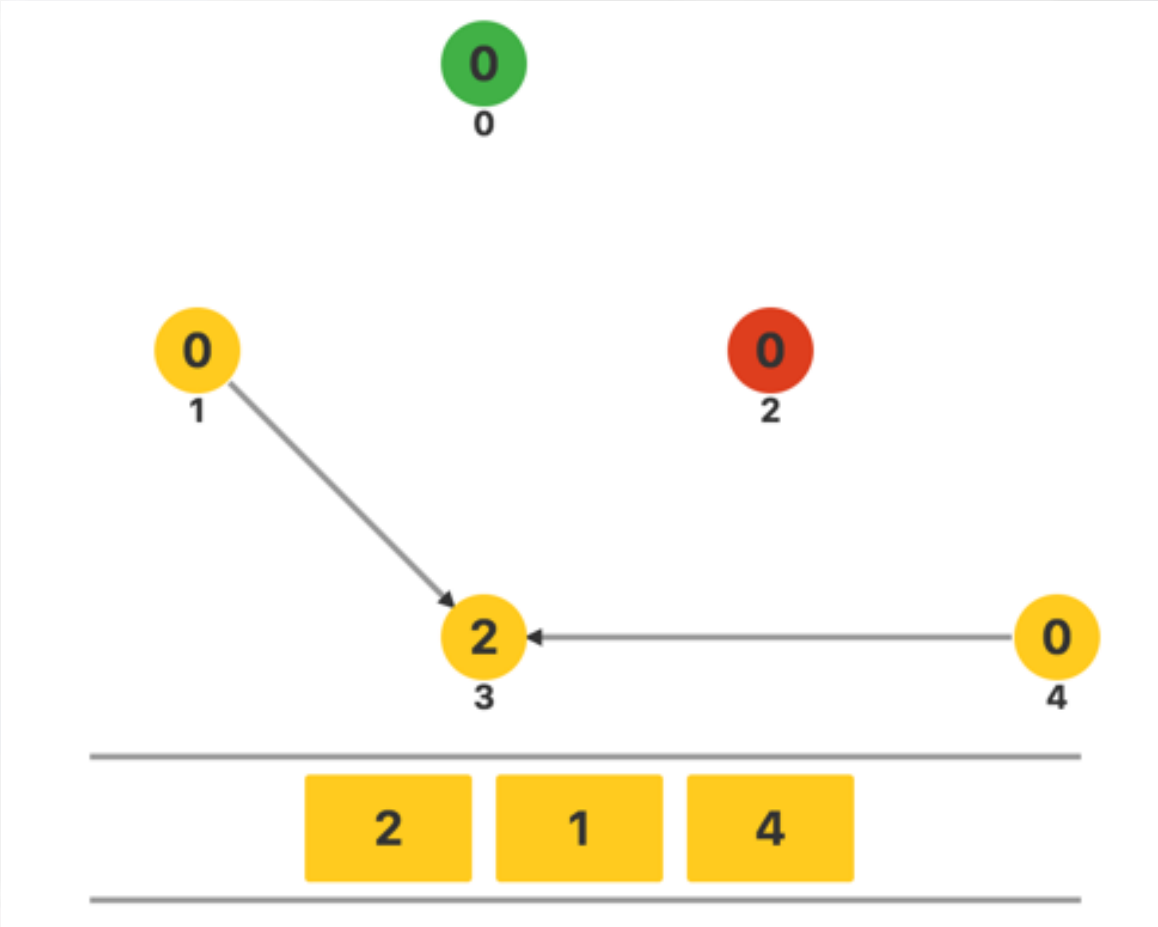
- 二、枚举 0连出的所有边，对于每条边，将另一端的顶点的入度减一。发现有新的入度为零的点 2，则将它们都插入到队列中。



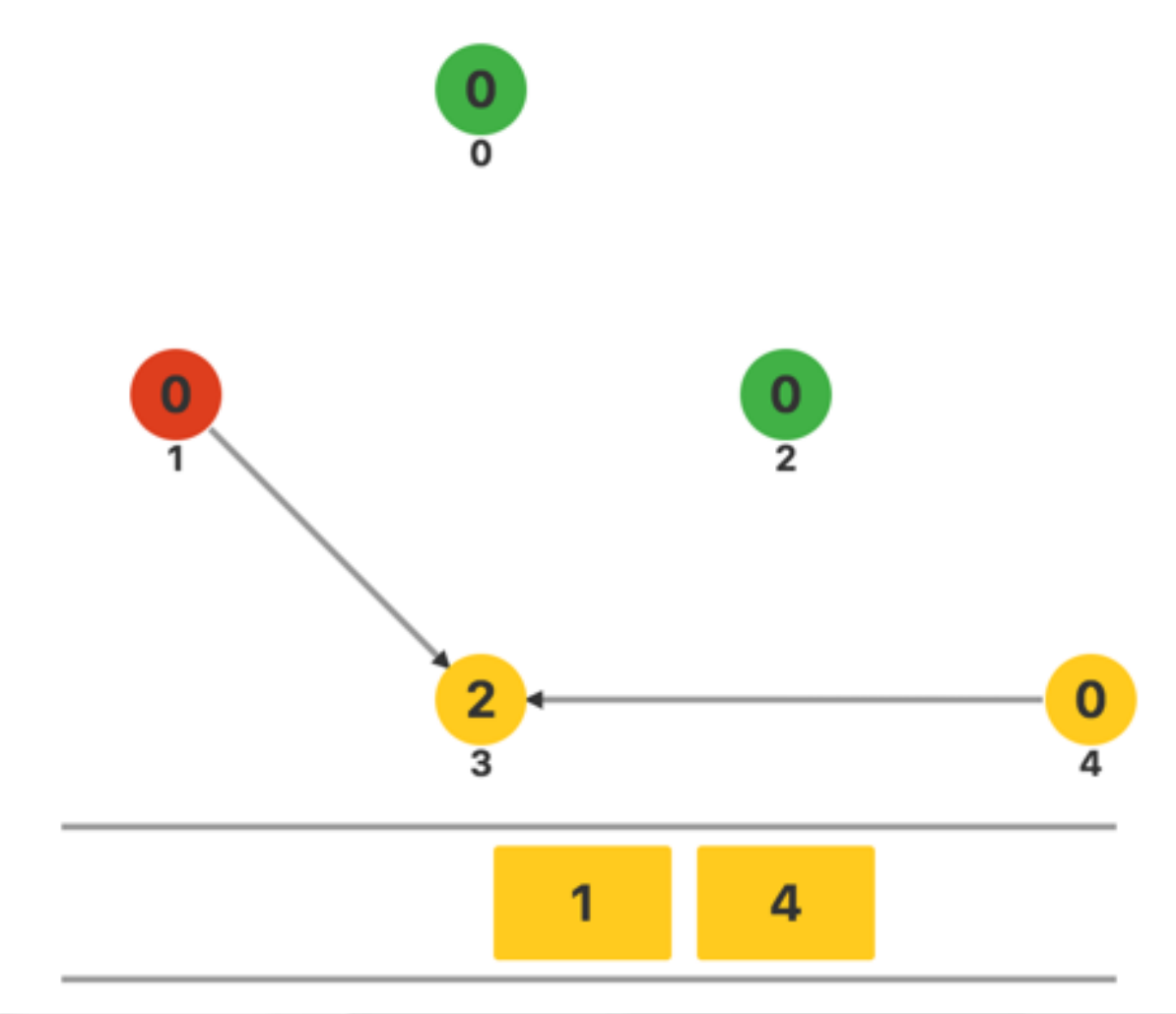
- 三、处理完成后将队首元素出队，继续访问当前的队首元素 2。



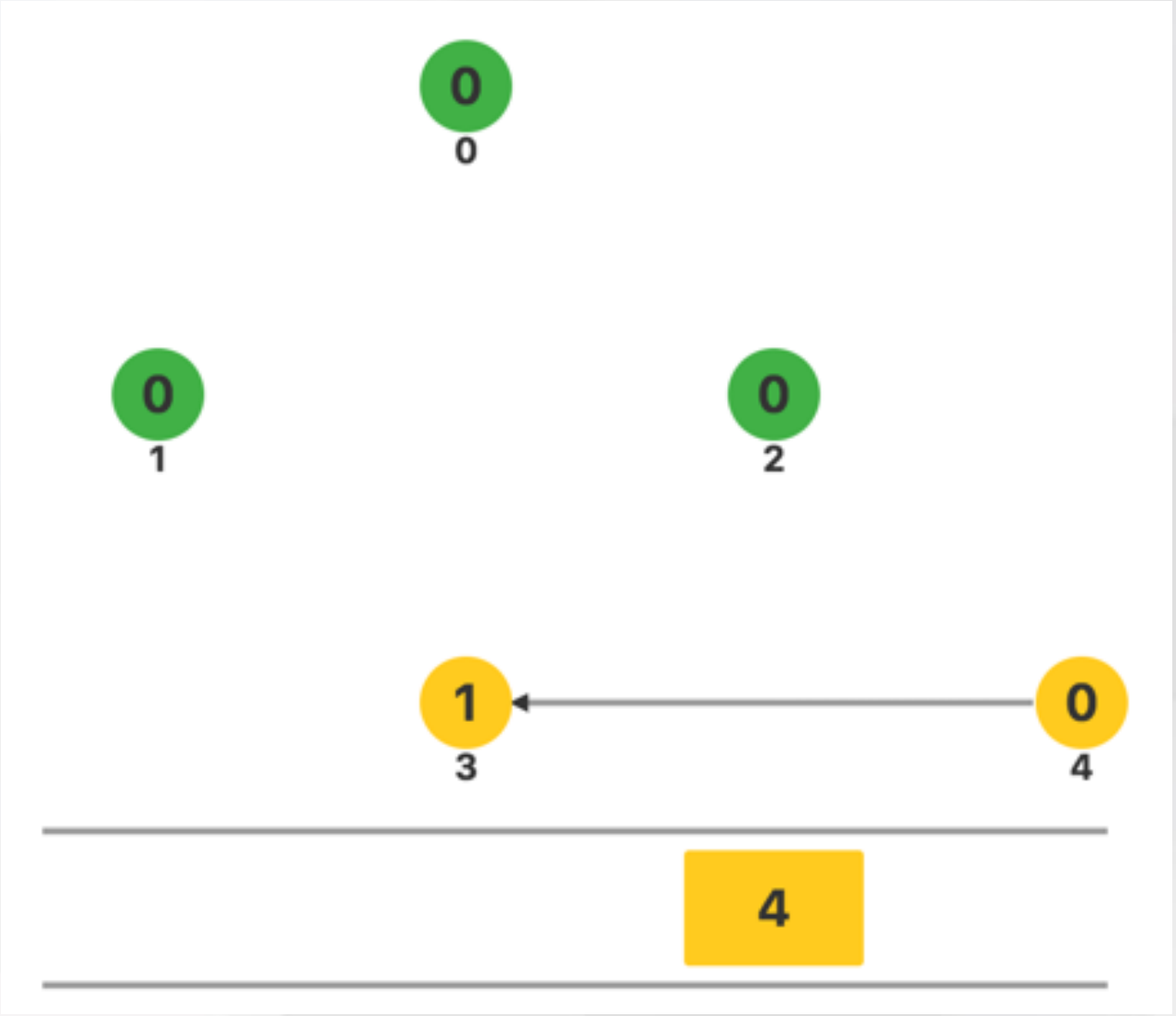
- 四、枚举 2连出的所有边，对于每条边，将另一端的顶点的入度减一。这时发现顶点 1 和顶点 4 的入度均为零，将它们都插入队列。



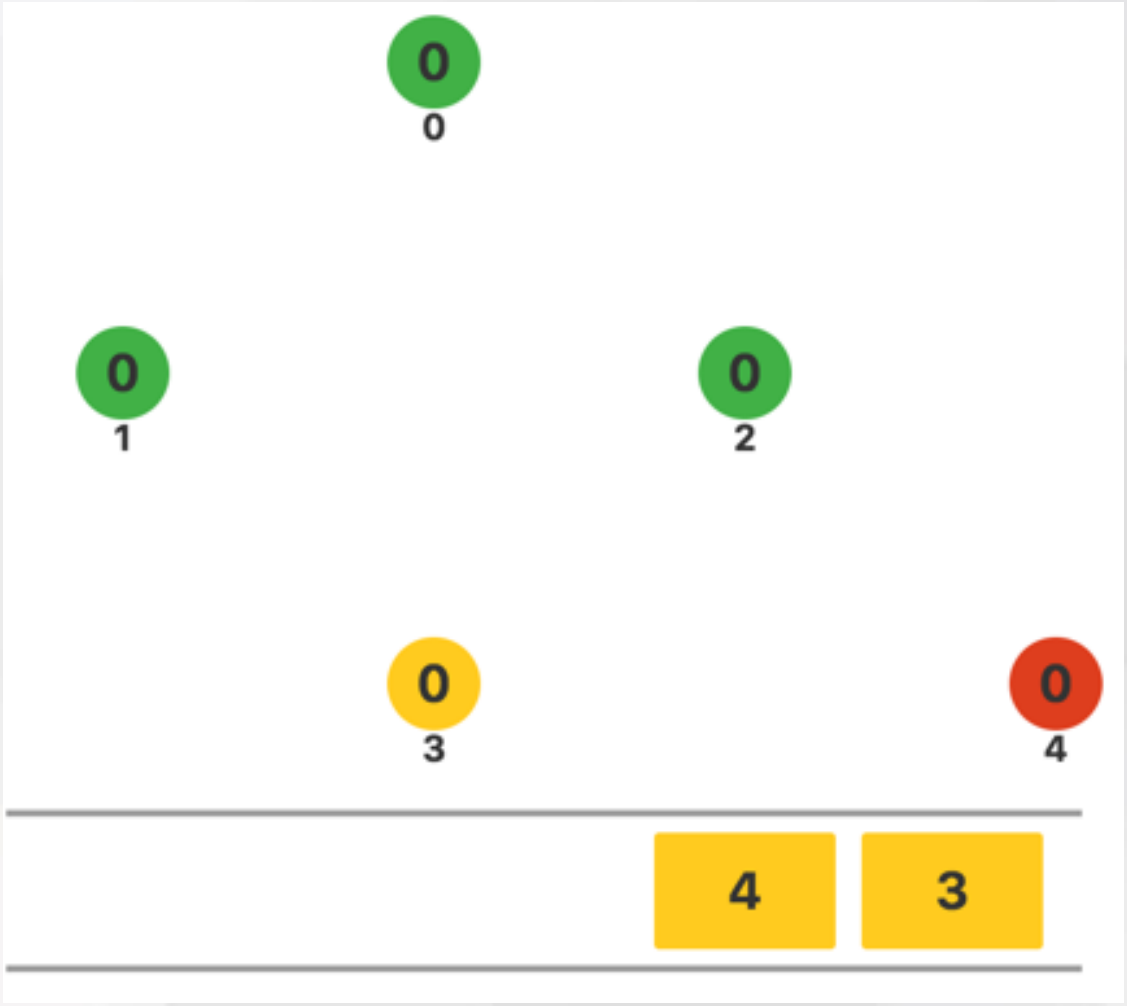
- 五、处理完成后将队首元素出队，继续访问当前的队首元素 1。



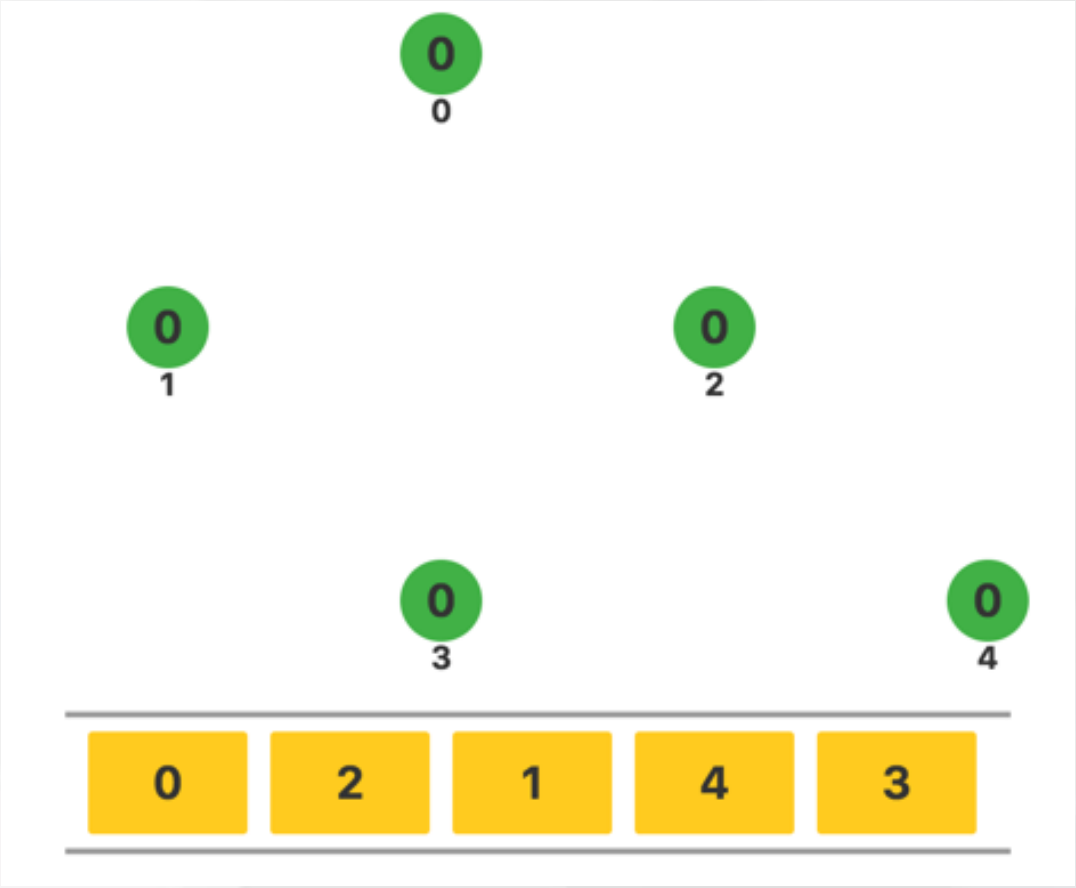
- 六、枚举 1 连出的所有边，修改对应的入度。



- 七、将 1出队，继续访问当前的队首元素 4，调整对应顶点的入度。发现顶点3的入度为零，将它插入队列中。



- 八、将 4 出队，目前队首顶点为 3，发现没有和它相邻的顶点，将其出队。此时队列为空，算法结束。
- 我们发现，这 5个顶点已经都访问过了，所以最终顶点入队的顺序就是这张有向图的一个拓扑排序。



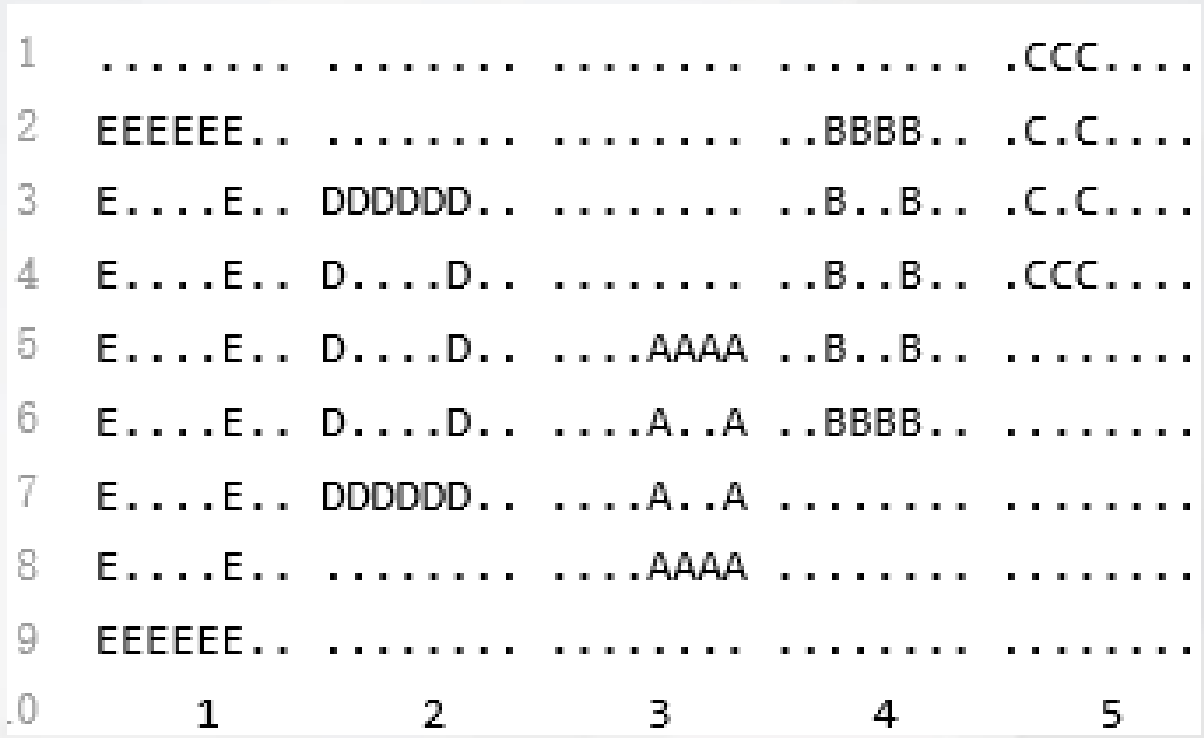
基于邻接表的 C++ 示例代码如下

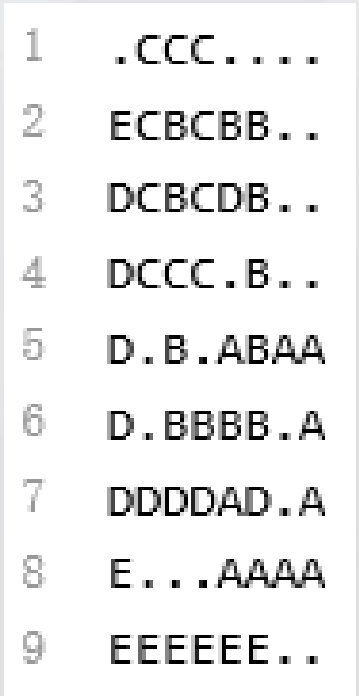
```
struct edge {// 使用邻接表存储
    int v, next;
} e[MAX_M];
int p[MAX_N], eid;

int topo() {
    queue<int> q;
    for (int i = 1; i <= n; i++) {
        if (indegree[i] == 0) {
            //将所有入度为零的顶点入队
            q.push(i);
        }
    }
```

```
while (!q.empty()) {
    int now = q.front();
    cout<<"visiting "<<now<<endl;
    q.pop();
    for (int i=p[now]; i!=-1; i=e[i].next)
    {
        int v = e[i].v;
        indegree[v]--;
        if (indegree[v] == 0) {
            // 将入度新变成零的顶点入队
            q.push(v);
        }
    }
}
}
```

例题：字母框

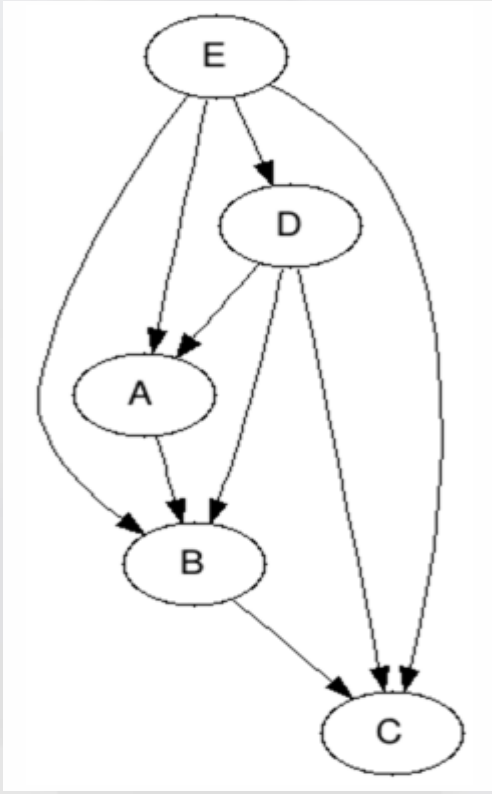
- 将若干个由字母组成的矩形叠加到一起，给定最终的叠加效果，求出一个合法的叠放次序。保证每个矩形只由一种字母组成，宽度为 1，且不同矩形对应的字母各不相同。
- 例如，对于左下面这 5 个矩形，有如右下的最终叠放效果，保证每个矩形的四条边上最终都至少出现一个点。





解法：

- 拓扑排序题目的关键，在于建立实际问题和图中有向边的对应关系。
- 由于题目保证，我们可以获得每个矩形四条边在图中占据的位置。如果两个矩形有公共点，例如矩形A和B，它们的公共点最终显示的是B，因此B一定在A之后放入。我们就可以从A点向B点连一条有向边。最终建立的有向图如下所示：
- 对于这张有向图，存在合法拓扑序列：
- E,D,A,B,C，这也就是合法的放入顺序。

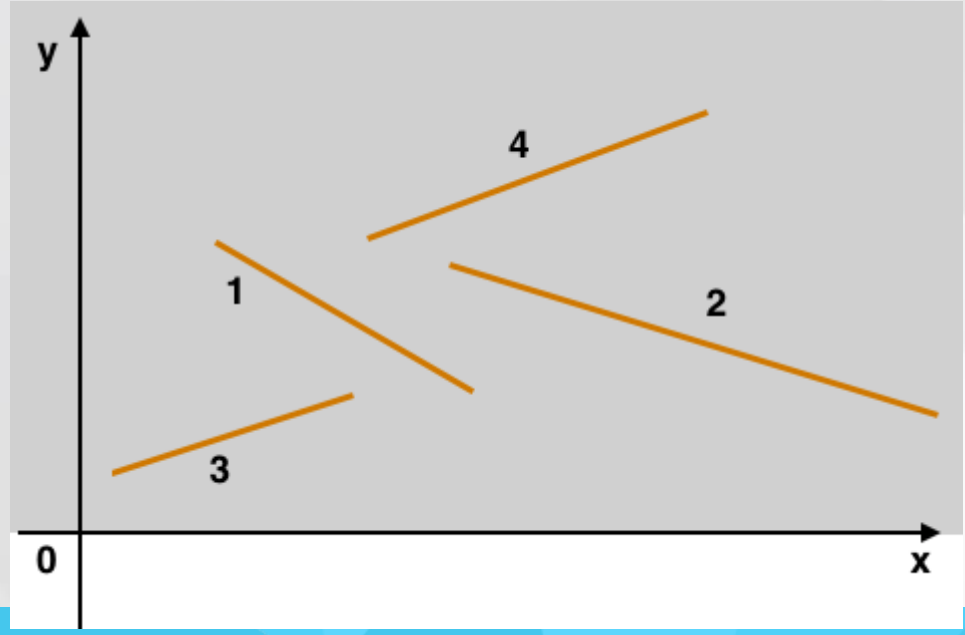


习题：威虎山上的分配

- 每年过年的时候，座山雕都会给兄弟们分银子，分银子之前，座山雕允许大伙儿发表意见，因为要是没法满足所有人的意见，指不定谁要搞出什么大新闻。不过每个人在提意见的时候只能说：“我认为 A 分的银子应该比 B 多！”。座山雕决定要找出一种分配方案，满足所有人的意见，同时使得所有人分得的银子总数最少，并且每个人分得的银子最少为 100 两。
- 输入格式：第一行两个整数 n,m($0<n\leq10000, 0<m\leq20000$)，表示总人数和总意见数；
- 以下 m行，每行两个整数 a,b，之间用一个空格隔开，表示某个意见认为第 a 号小弟所分得的银两应该比第 b 号小弟多，所有小弟的编号由 1 开始。
- 输出格式：若无法找到合法方案，则输出Unhappy!(不包含引号)，否则输出一个数表示最少总银两数。
- 样例输入
- 3 2
- 1 2
- 2 3
- 样例输出
- 303

习题：回收元件

- 小明在做一个高级的电学实验，工作面板上放了很多电子元件，这些电子元件可以看成直的导线。
- 实验结束，小明想要回收这些元件，但是小明不能直接接触他们把他们拿走，这样会导致元件的损坏，并且因为元件上还带有一些正负电荷，所以元件之间也不可以互相接触，否则可能发生短路。小明现在有一个特制的设备，可以沿工作面板的一个边缘把元件一个一个依次吸走，而元件在吸走的过程当中不会发生旋转，也就是说元件是通过平移离开工作台的。
- 如下图所示小明可以把元件依次按 3,1,2,4 的顺序吸走（规定只能朝 y 轴负方向平移）
- 如上的坐标系中，原点坐标是（0,0），工作面板右上角是（12000,12000），所有电子元件以线段的形式存在，并且只能朝y 轴负方向平移。你的任务是帮助小明决定吸走电子元件的次序。
- 输入格式：第一行输入一个整数 N（N≤6000），表示电子元件数量。接下来 N 行每行输入四个整数 x1,y1,x2,y2(0≤x1,y1,x2,y2≤12000)表示电子元件两个端点的坐标 (x1,y1),(x2,y2)。
- 输出格式：一行 N个整数，表示取出元件的顺序，元件序号按输入顺序由 1 计数。如果有多种方案，输出任意一个即可。本题答案不唯一，符合要求的答案均正确
- 样例输入
- 4
- 3 5 7 4
- 6 5 15 4
- 1 1 3 2
- 5 6 10 8
- 样例输出
- 3 1 2 4



7.4. 欧拉回路

- 若图 G 中存在这样一条路径，使得它恰好通过 G中每条边一次，则称该路径为 欧拉路径。若该路径是一个环路，则称为 欧拉（Euler）回路。具有欧拉回路的图称为 欧拉图。具有欧拉路径但不具有欧拉回路的图称为 半欧拉图。
- 有关欧拉路的重要性质：

无向图

1. 一个无向图 G 存在欧拉路径当且仅当无向图 G 是连通图，且该图中有两个奇度顶点（度数为奇数）或者无奇度顶点。

2. 当无向图 G 是包含两个奇度顶点的连通图时，G的欧拉路径必定以这两个奇度顶点为端点。

3. 一个无向图 G 存在欧拉回路当且仅当无向图 G 连通且不存在奇度顶点。

有向图

1. 一个有向图 G存在欧拉路径当且仅当 G是连通的有向图，且满足以下两个条件之一：

○ 所有顶点的入度和出度相等；

○ 有一个顶点的出度与入度之差为 1，一个顶点的出度与入度之差为 −1，其余顶点的入度和出度相等。

2. 当有向图 G 包含两个入度和出度不相同的顶点且有欧拉路径时，欧拉路径必定以这两个入度出度不相同的顶点为端点。

3. 一个有向图 G 存在欧拉回路当且仅当图 G 是连通的有向图，且所有顶点的入度和出度相等。

无向图欧拉路径

- 计算无向图的欧拉路经可以用“套圈法”、基于深度优先搜索来实现。
- 从一个合适的顶点出发进行深度优先搜索。当搜到一个顶点 u时：

1. 如果此时没有顶点与该顶点相连，那么就将 u加入路径中并回溯；
2. 如果有顶点 v与顶点 u相连，那么就删除无向边 <u,v>，并继续搜索顶点 v。将所有和 u 相邻的顶点遍历完之后，将 u 加入路径中。

```
1.  const int MAX_N = 100;
2.  int mat[MAX_N][MAX_N];
3.  int match[MAX_N];  // 表示顶点剩余的度
4.  int n;  // 顶点个数
5.  void solve(int u) {
6.      if (match[u] > 0) {
7.          for (int i = 0; i < n; ++i) {
8.              if (mat[u][i]) {
9.                  mat[u][i]--;
10.                 mat[i][u]--;
11.                 solve(i);
12.             }
13.         }
14.     }
15.     cout << "visiting " << u << endl;
16. }
```

有向图欧拉路经

- 计算有向图的欧拉路经同样可以用“套圈法”。和无向图唯一不同的是，在记录方案的时候将顶点编号插入栈中，并在最后将栈中的元素依次输出。也就是说，将之前输出的顺序逆置。

```
const int MAX_N = 100;
const int MAX_M = 10000;
int mat[MAX_N][MAX_N];
int match[MAX_N]; // 表示顶点剩余的度
int n; // 顶点个数
int stk[MAX_M], top = 0; // 用数组 stk 来模拟一个栈
void solve(int u) {
    if (match[u] > 0) {
        for (int i = 0; i < n; ++i) {
            if (mat[u][i]) {
                mat[u][i]--;
                solve(i);
            }
        }
    }
    stk[top++] = u; // 将顶点 u 插入栈中
}
```

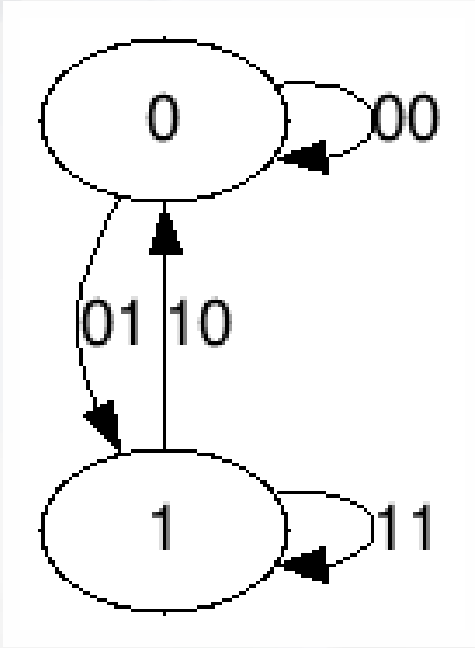
例题：De Bruijin 序列

- De Bruijin 序列是指这样一个序列：对于一个包含 n个元素的字符集，生成一个仅包含 n种字符的字符串，使得这个字符串中 k^n 种长度为k的子串均恰好出现一次。
- 例如，对于 n=2,k=2,S={0,1} 的 De Bruijin 序列，有如下满足要求的一个字符串：

```
1   00110
2   00
3    01
4     11
5      10
```

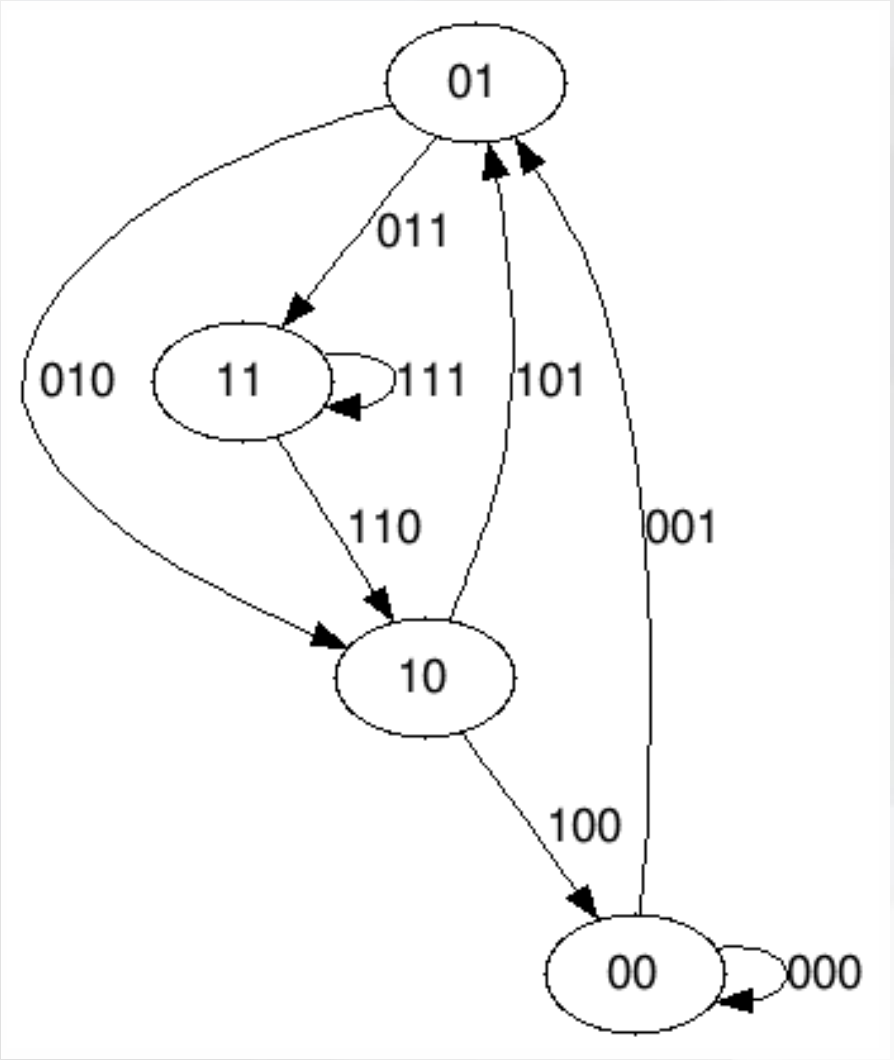
转化为图论问题

- 我们可以把这个问题转化为图论问题：图中的顶点代表每种长度为 k−1 的子串，每条边代表一个子串增加一个字符后的转移，如下图所示：



- 对于 n=2,k=2,S={0,1} 的情况，0增加一个字符0后，变成00，最后 k−1 位是0，所以从0连向0一条边00；0增加一个字符1后，变成01，最后 k−1 位是1；以此类推。

- 我们用 n=2,k=3,S={0,1} 来进一步理解这个建模：
- 图中的每条边都对应着一个长度为 k的子串，而每走一条边，就意味着在整个字符串的最后增加一个字符。这样，图中的一个欧拉路径就对应着一个满足要求的完整字符串。例如上图中，一个欧拉路径01 11 11 10 01 10 00 00 01就对应着字符串0111010001，其中恰好包含 2^3=8种不同的子串，且每种子串刚好出现一次。



习题：判定欧拉回路

- 你学过一笔画问题么？其实一笔画问题又叫欧拉回路，是指在画的过程中，笔不离开纸，且图中每条边仅画一次，而且可以回到起点的一条回路。
- 小明打算考考你，给你一个图，问是否存在欧拉回路?
- 输入格式：第 1 行输入两个正整数，分别是节点数N(1<N<1000) 和边数 M(1<M<100000)；紧接着 M 行对应 M条边，每行给出一对正整数，分别是该条边直接连通的两个节点的编号（节点从 1 到 N 编号）。
- 输出格式：若存在欧拉回路则输出 1，否则输出 0。
- 样例输入
- 10 11
- 1 2
- 2 3
- 3 4
- 4 5
- 5 6
- 6 7
- 7 8
- 8 9
- 9 10
- 10 3
- 3 1
- 样例输出
- 1

习题：单词拼接

- 花椰菜君给了小明 n个单词，如果一个单词的最后一个字母和另一个单词的第一个字母相同，那么两个单词就可以连接在一起组成一个新的单词。现在花椰菜君想要小明计算一下，给定的 n个单词是否可以全部连接在一起。
- 输入格式：第一行输入一个整数 n，代表一共有 n 个单词（1≤n≤100,000）。接下来输入 n 行，每行输入一个单词。单词均由小写字母组成，每个单词长度不超过 20。
- 输出格式：输出一行，如果所有的单词都可以连接在一起并且可以形成一个环，那么输出Euler loop；如果所有单词都可以连接在一起，但是不会形成环，输出Euler path；如果所有单词不能连在一起，那么输出impossible。
- 样例输入
- 3
- euler
- ruby
- jisuanke
- 样例输出
- Euler path

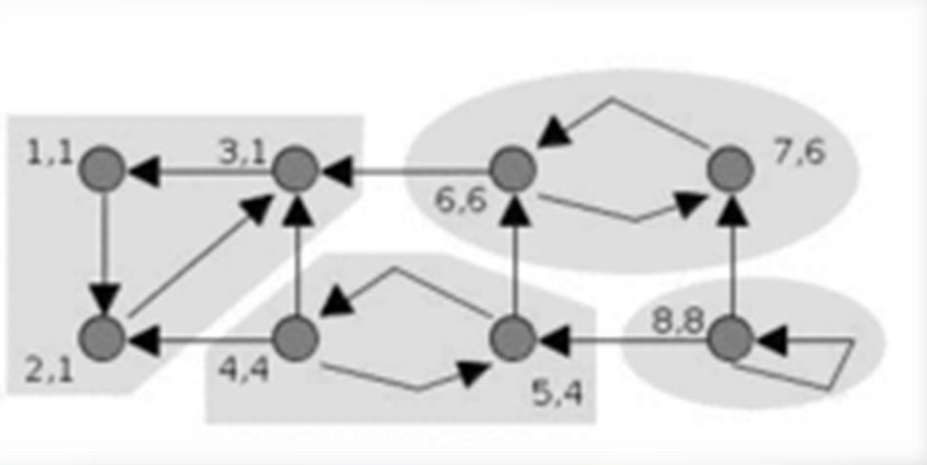
7.5. Tarjan 强连通分量和缩点

强连通分量

- 强连通(strongly connected)： 在一个有向图G里，设两个顶点 u,v，它们之间存在互相可达的两条路径：u...v 和 v...u ，我们就叫这两个顶点（u，v）强连通。
- 强连通图是指，对于图 G中的每一对顶点 u,v，它们都强连通。
- 有向图G的强连通分量(strongly connected components)是指 G的极大强连通子图。如果将每一个强连通分量缩成一个点，则原图 G将会变成一张有向无环图（DAG）。
- 寻找有向图强连通分量的常见高效算法有 Kosaraju 算法、Tarjan 算法、Gabow 算法。但是因为在 Tarjan 算法和 Gabow 算法的过程中，只需要进行一次的深度优先搜索，所以相比 Kosaraju 算法效率较高。

Tarjan 算法

- 任选一顶点开始进行深度优先搜索（若深度优先搜索结束后仍有未访问的顶点，则再从中任选一点再次进行）。搜索过程中已访问的顶点不再访问。搜索树的若干子树构成了图的强连通分量。
- 顶点按照被访问的顺序存入栈中。从搜索树的子树返回至一个结点时，检查该结点是否是某一强连通分量的根结点（见下）并将其从栈中删除。如果某结点是强连通分量的根，则在它之前出栈且还不属于其他强连通分量的结点构成了该结点所在的强连通分量。



- 图片来自 Reputation and Community。

算法描述

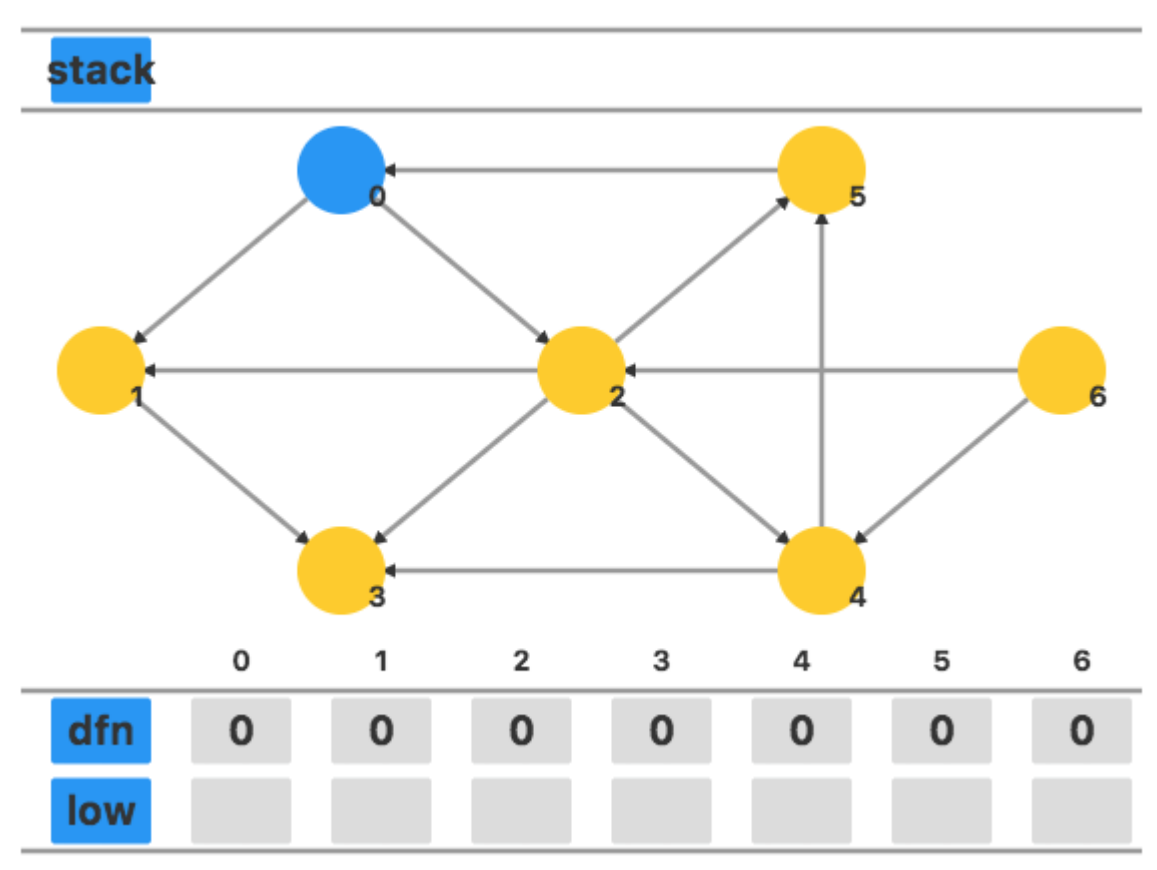
我们定义 dfn(u) 为顶点u被搜索的次序(时间戳)，low(u) 为 u 或 u 的子树能够追溯到的最早的栈中顶点的次序号。由定义可以得出算法，

1. 遍历一个顶点 u，得到时间戳 dfn_u，并计算由该点可追溯到的最老时间戳 low_u。
2. 枚举该点所有相邻顶点
   - 若相邻顶点 v 的dfn_v 为 0，表明未被搜索过，递归搜索之
   - 若相邻顶点v的 dfn_v 不为 0，则顶点 v 被搜索过，判断 v 是否在栈中，且 v 的时间戳 dfn_v 小于当前时间戳 dfn_u，可判定成环。将 low_u更新成 dfn_v 。
3. 若顶点 u的 low_u 和 dfn_u 相等，说明这个顶点是当前强连通分量内所有顶点在栈中最早的一个，将该强连通分量内所有顶点从栈中全部弹出。
   - 时间复杂度是 O(|V|+|E|)。

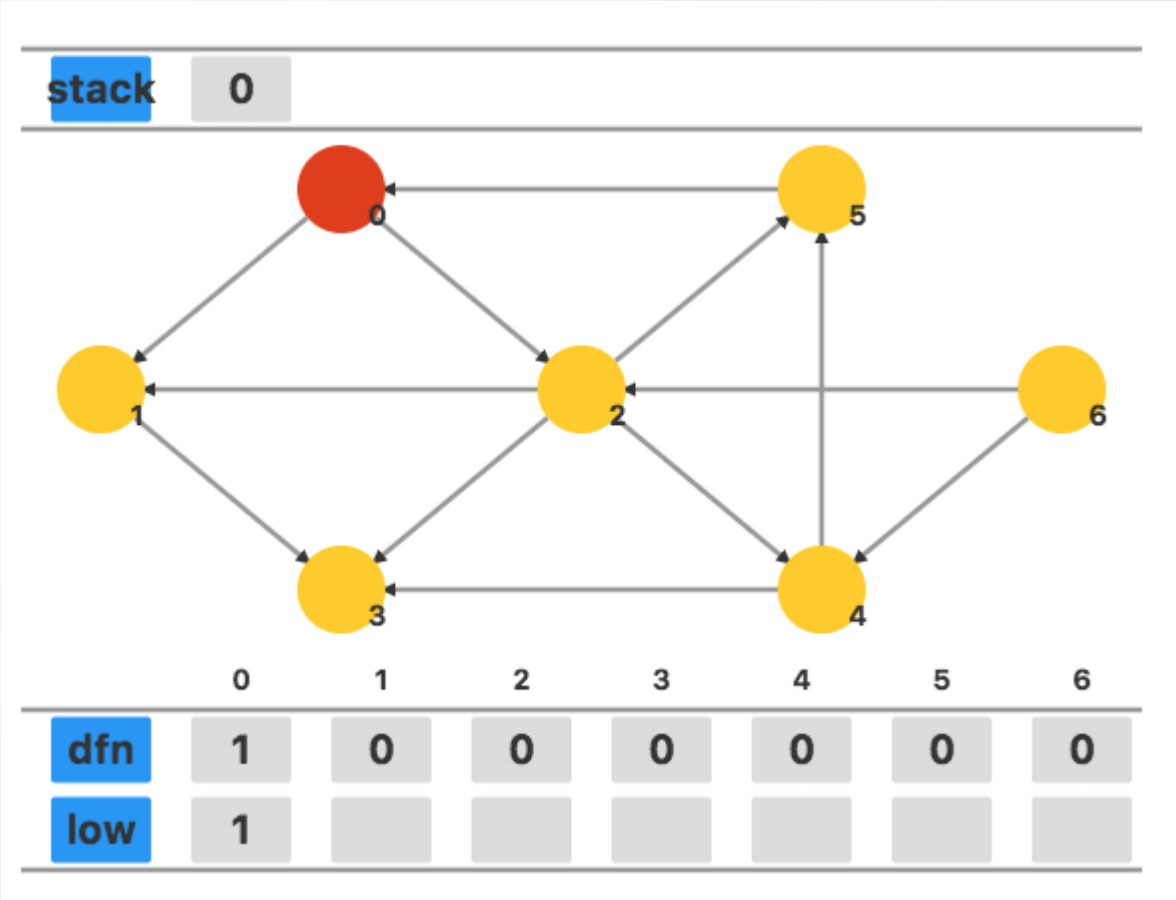
```
tarjan(u)
{
    dfn[u] = low[u] = ++Index
                // 为节点 u 设定次序编号和 low 初值
    S.push(u)  // 将节点 u 压入栈中
    for each (u, v) in E  // 枚举每一条边
        if (v is not visited)  // 如果节点 v 未被访问过
            tarjan(v)  // 继续向下找
            low[u] = min(low[u], low[v])
        else if (v in S)  // 如果节点 v 还在栈内
            low[u] = min(low[u], dfn[v])
    if (dfn[u] == low[u])
                // 如果节点 u 是强连通分量的根
        repeat
            v = S.pop
                // 将v退栈，为该强连通分量中一个顶点
            print v
        until (u== v)
}
```

举例

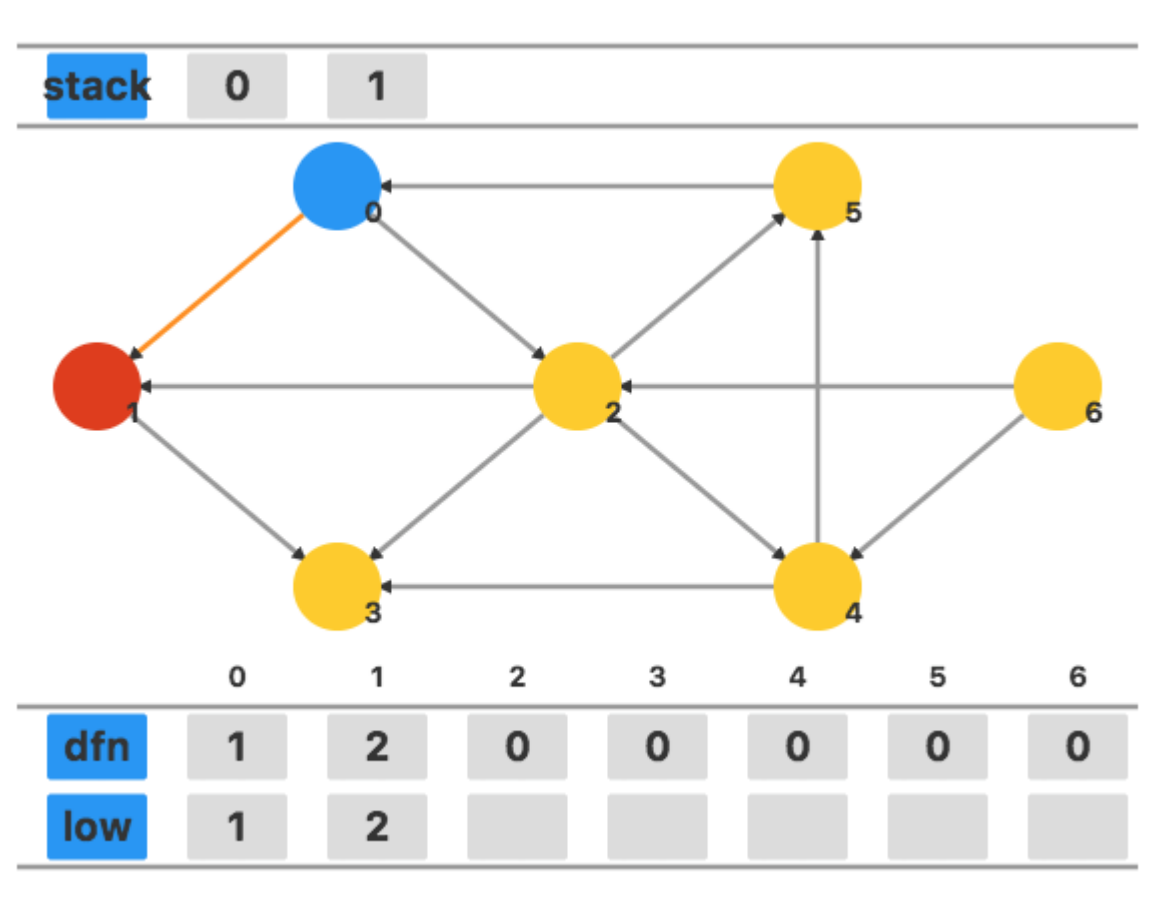
- 我们用一个例子来说明求强连通分量的 Tarjan 算法。
- 1.对于如下这个有向图，我们从顶点 0 开始进行 DFS。



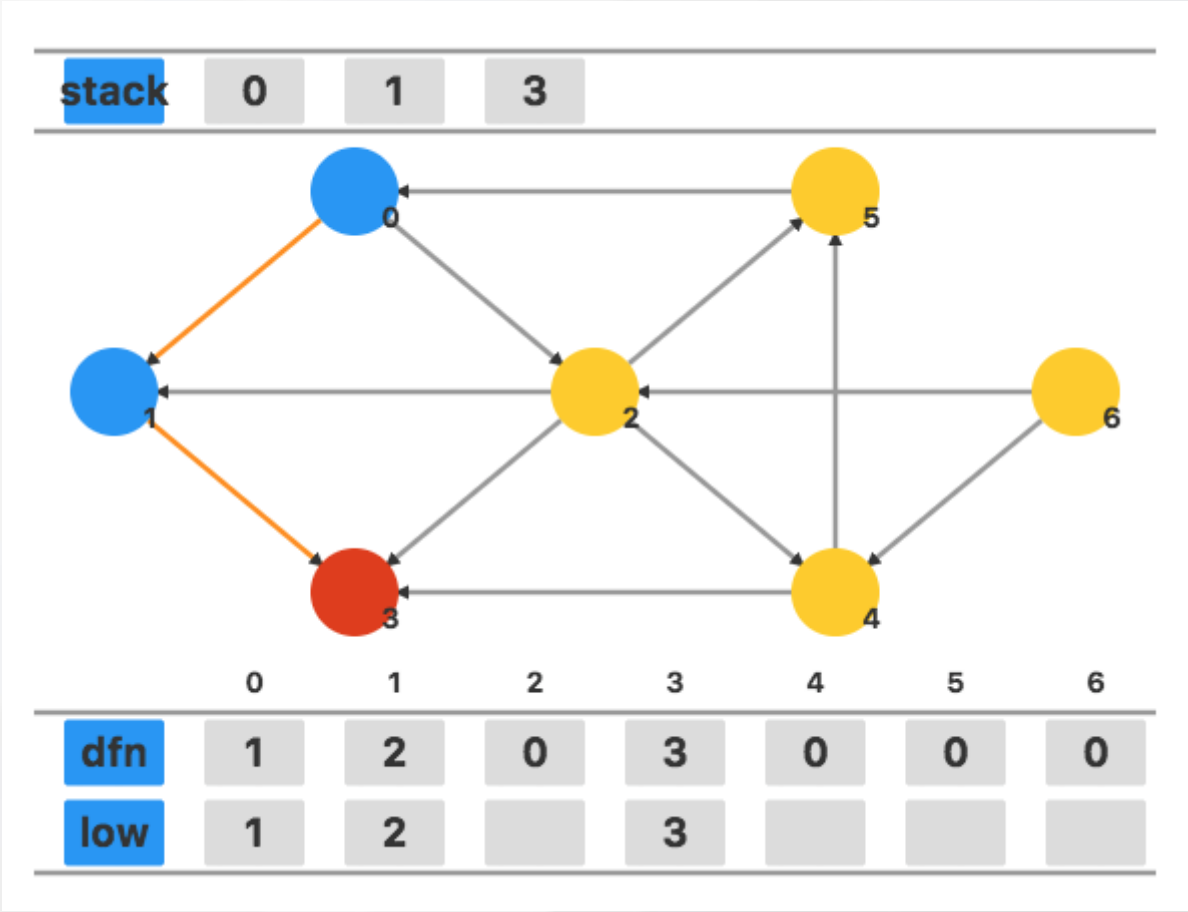
- 2.将顶点 0的 dfn[0] 和 low[0] 都设置为当前时间戳 1。



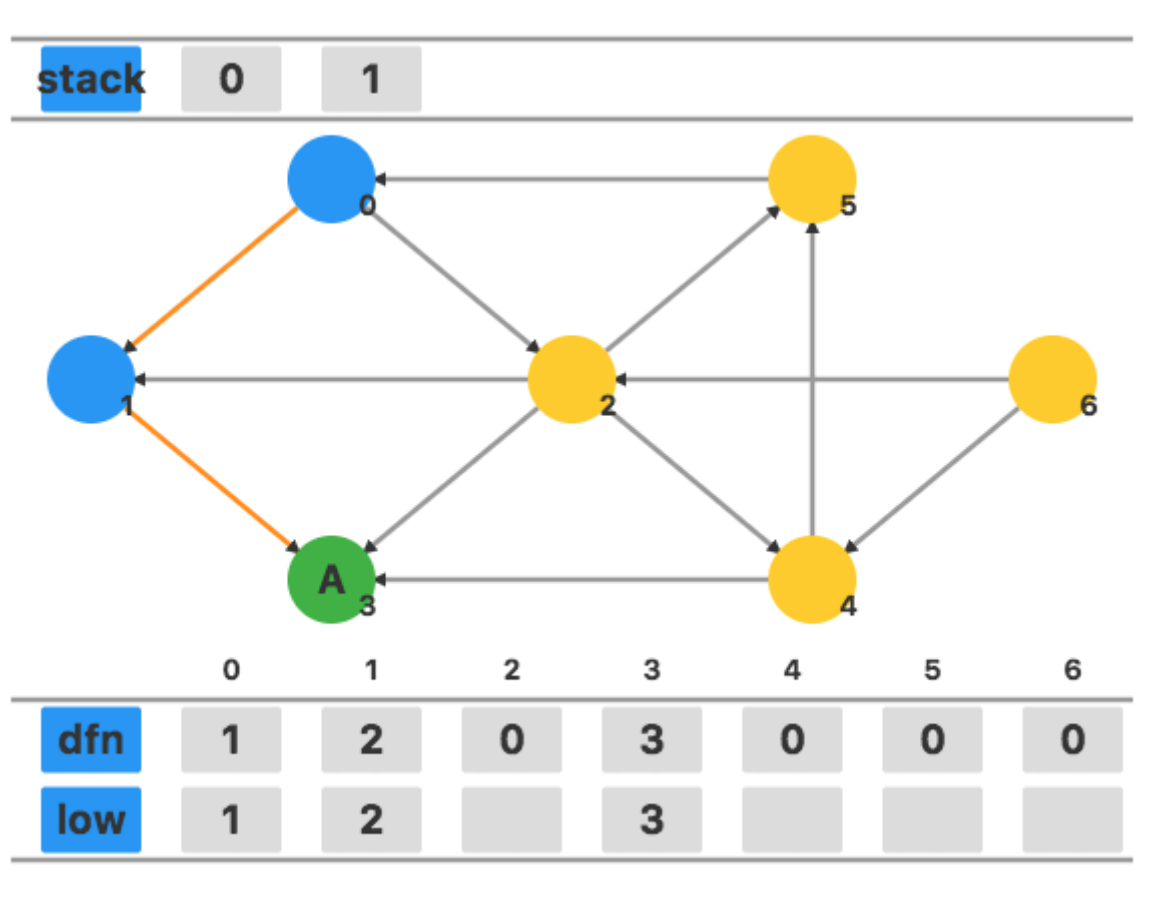
- 3.访问相邻未访问的顶点 1，将 dfn[1] 和 low[1] 都设置为当前时间戳 2。



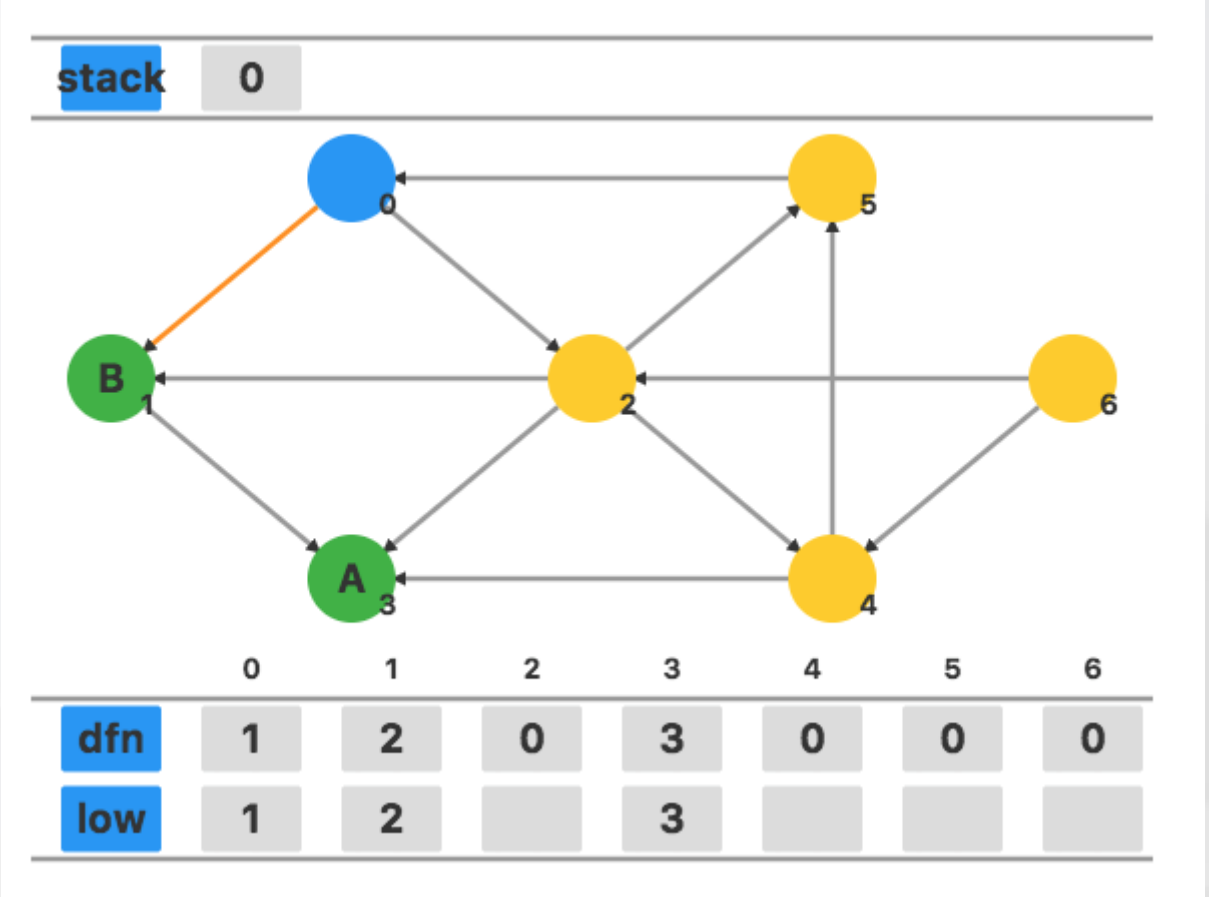
- 4.访问相邻未访问顶点 3，将 dfn[3] 和 low[3] 都设置为当前时间戳 3。



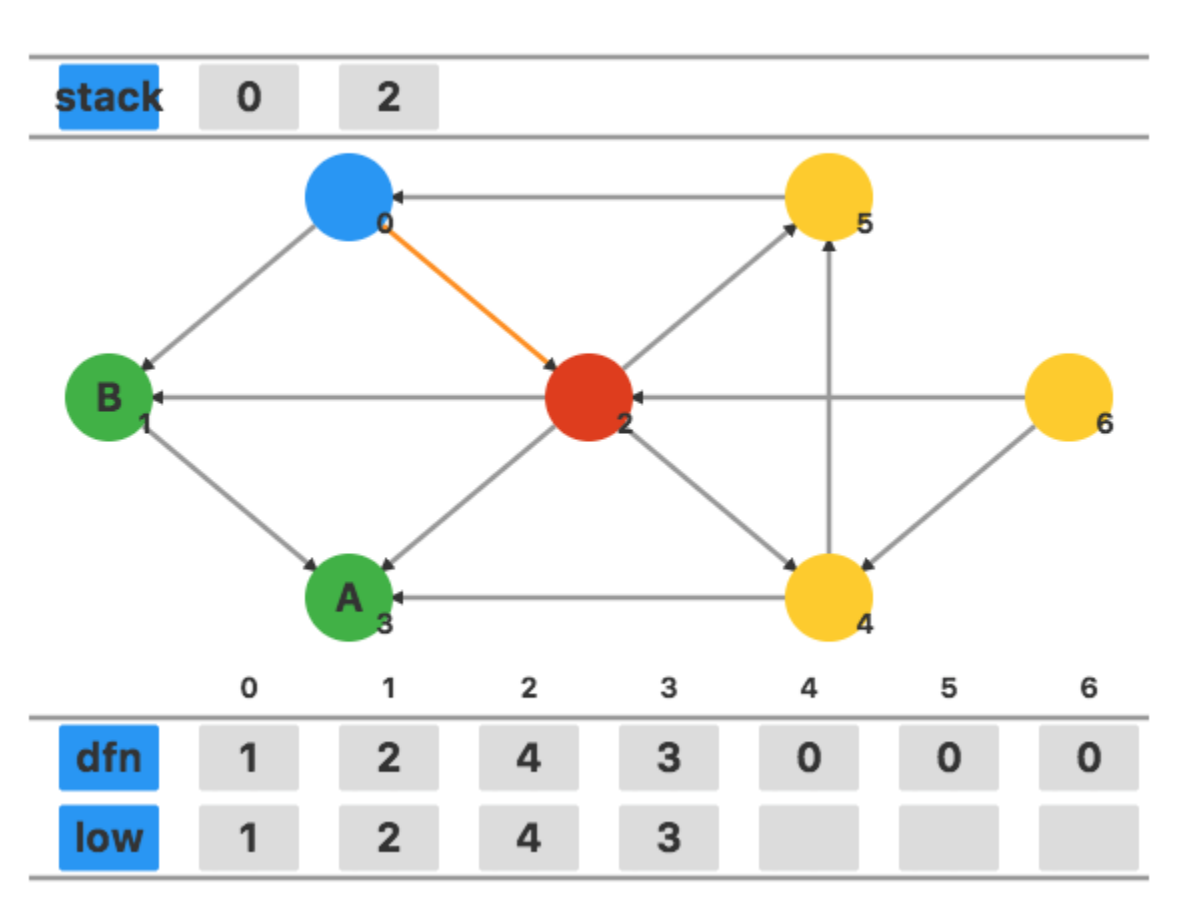
- 5.没有其他相邻顶点，此时 dfn[3]=low[3]，故将 3出栈作为一个强连通分量。



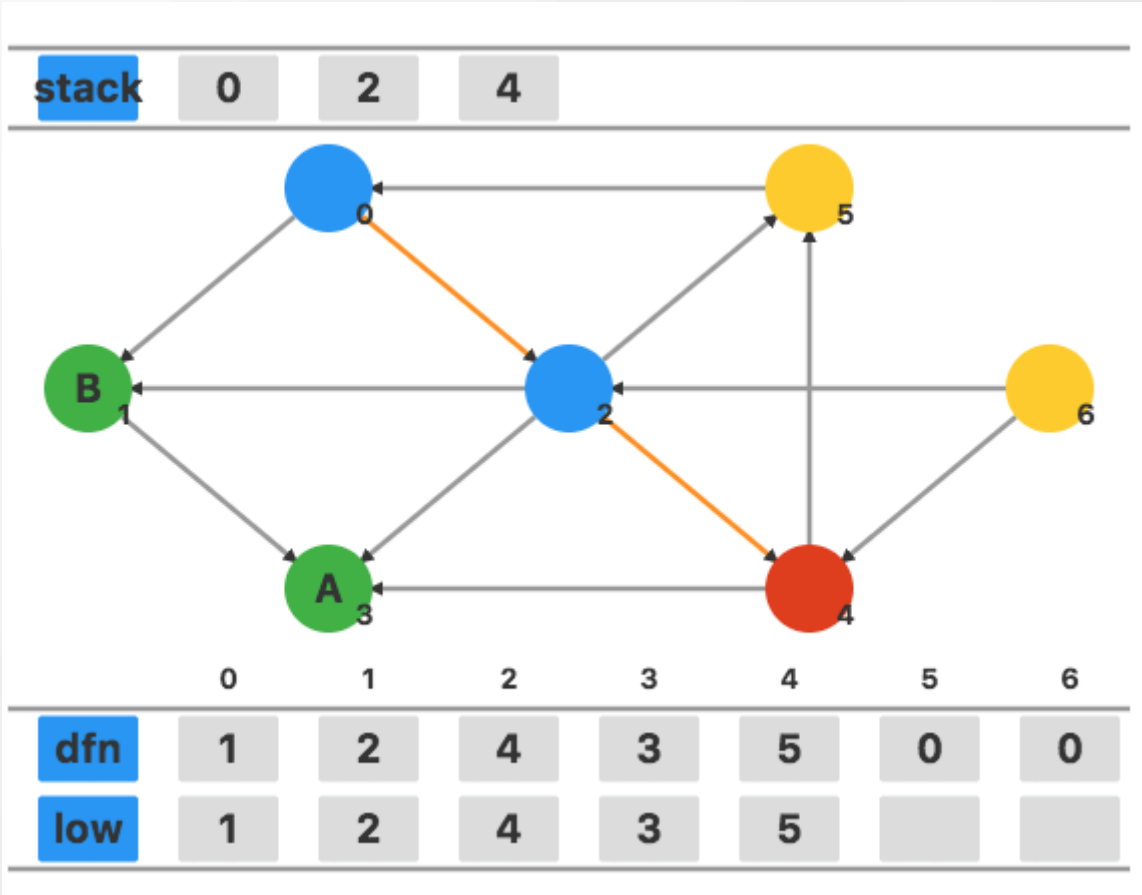
- 6.同样地，因为 dfn[1]=low[1]，所以将 1 出栈作为一个强连通分量。



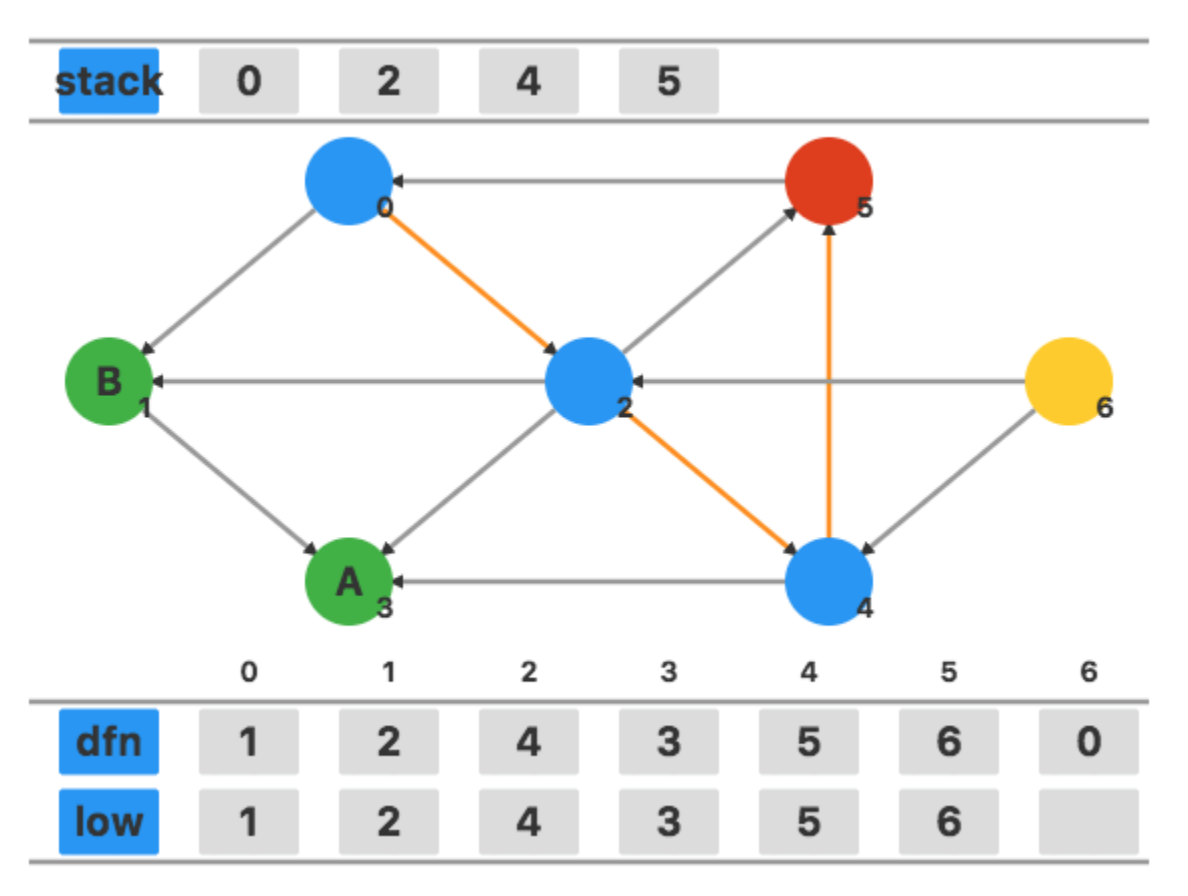
7.继续访问和 0 相邻的其他顶点 2，将 dfn[2]和low[2]都设置为当前的时间戳 4。



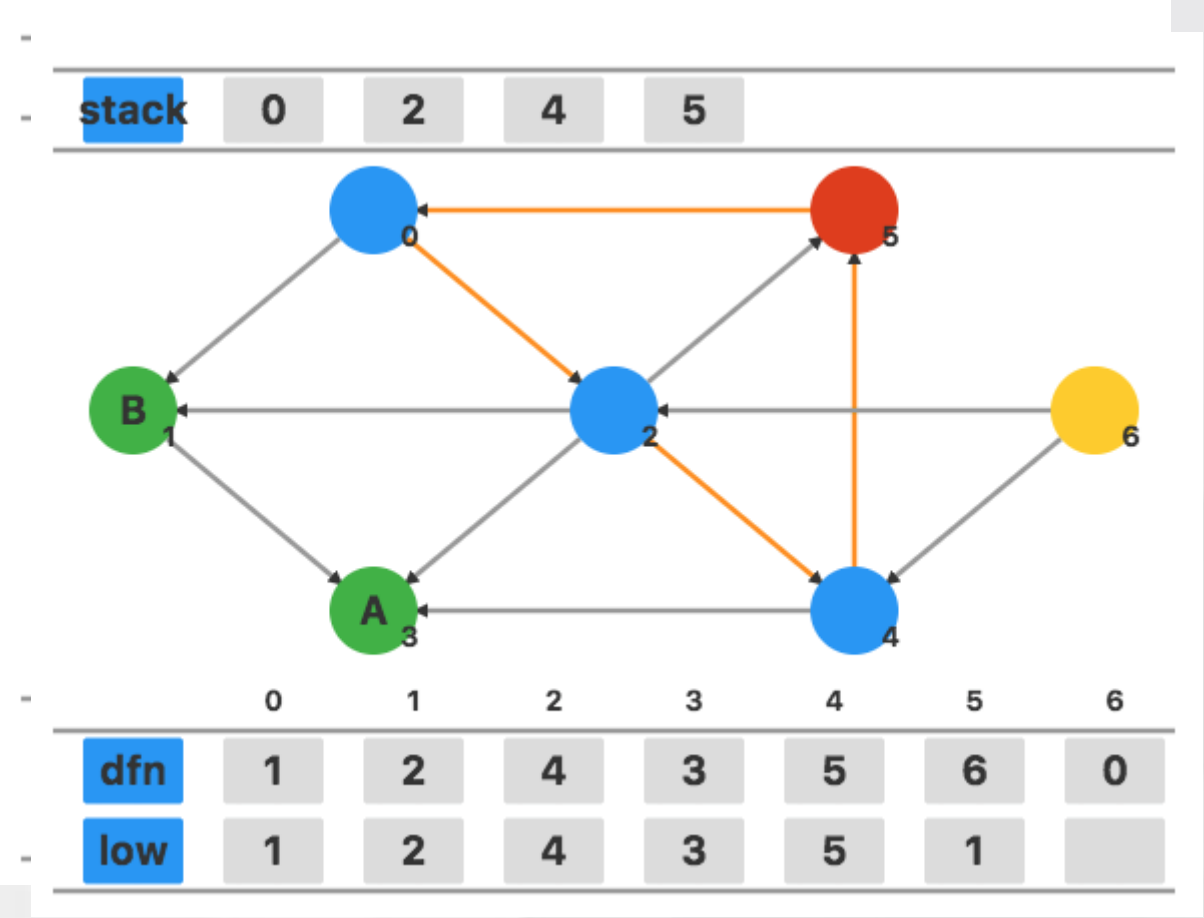
- 8.访问相邻顶点 4，将 dfn[4] 和 low[4] 都设置为当前的时间戳 5。



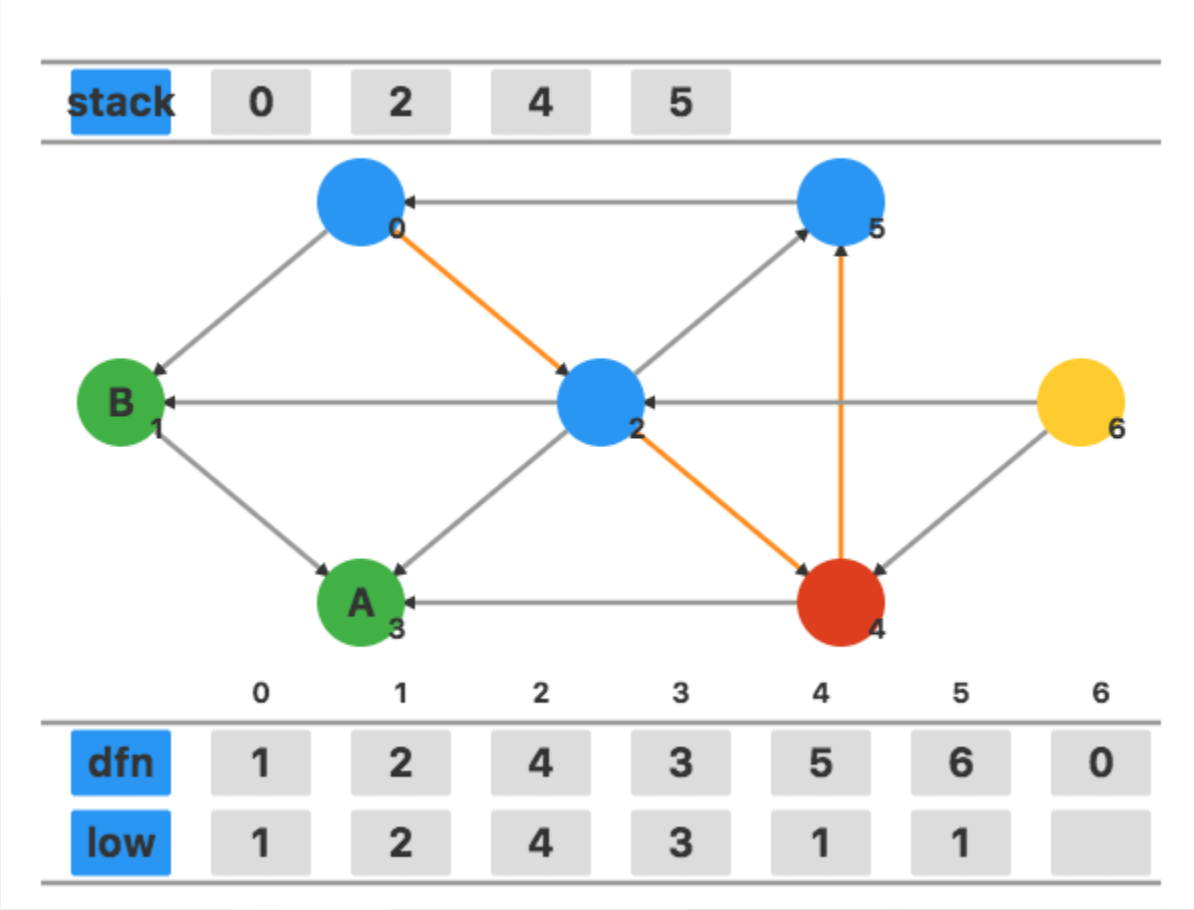
- 9.访问相邻顶点 5，将 dfn[5] 和 low[5] 都设置为当前的时间戳 6。



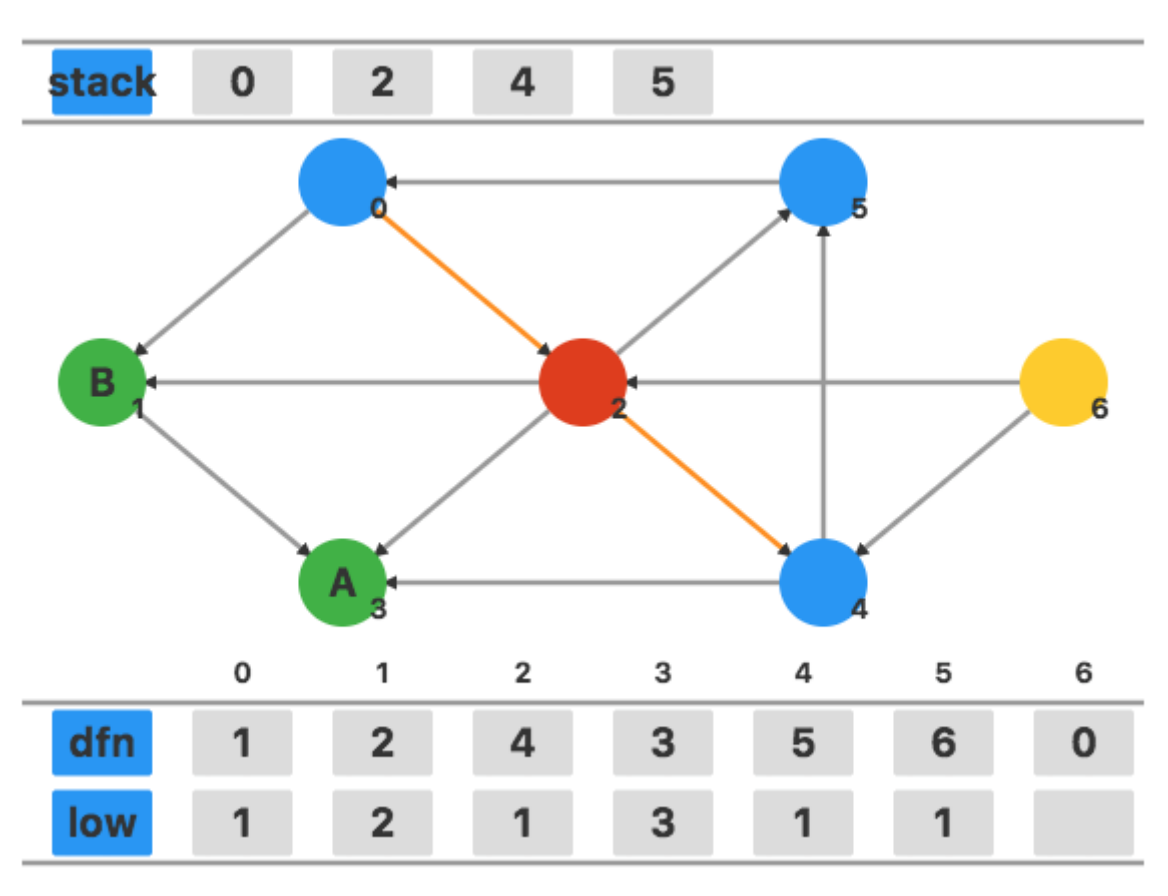
- 10.发现此时有指向栈中元素 0的边，将 low[5]更新为 dfn[0]=1。



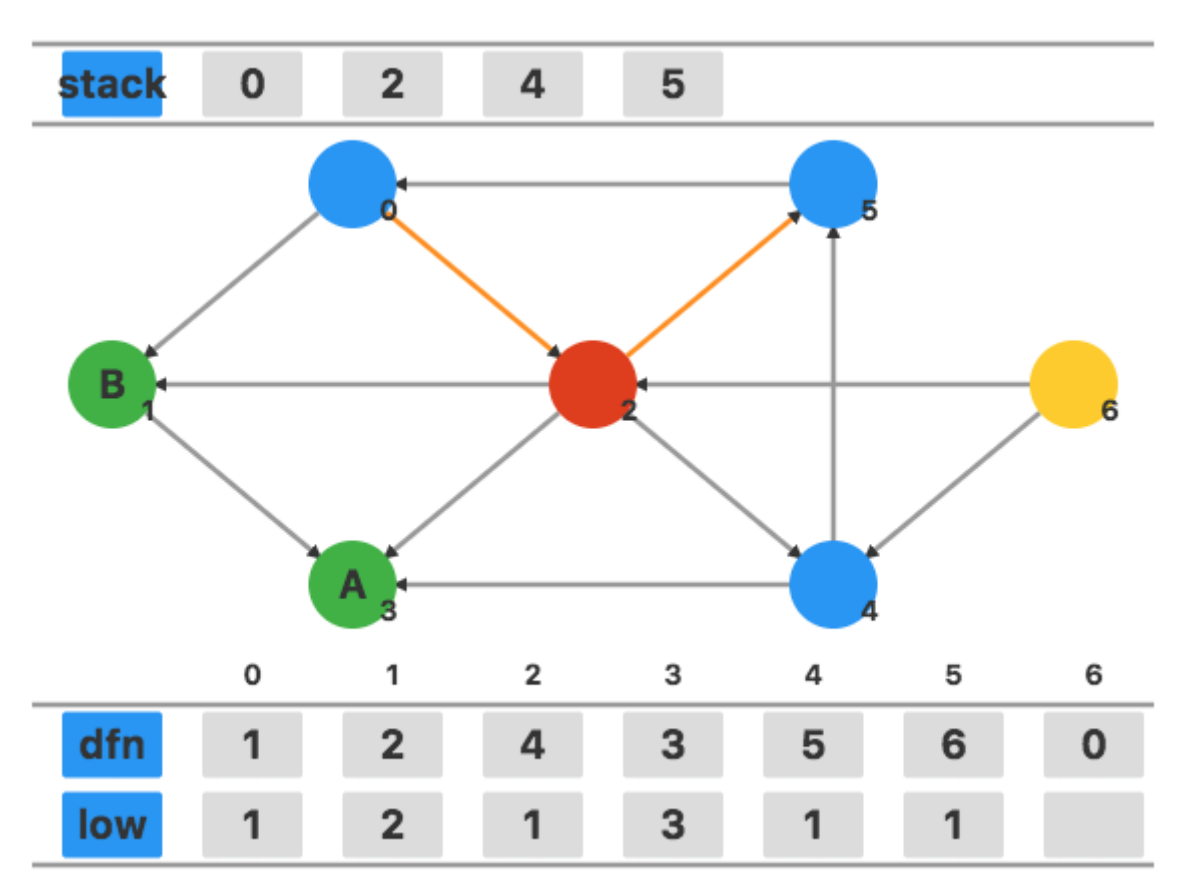
- 11.回溯到顶点 4，用 low[5] 更新 low[4]，更新为 1。



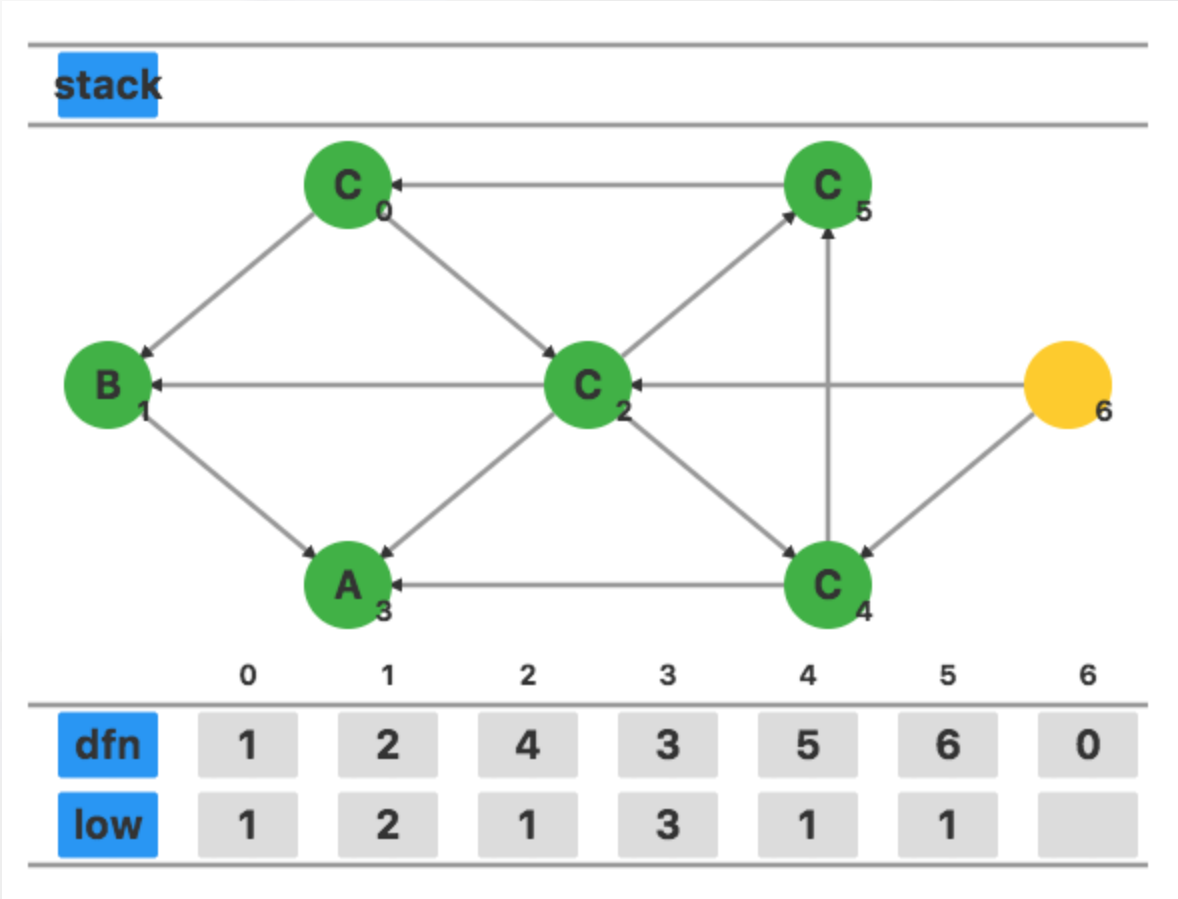
- 12.回溯到顶点 2，用 low[4] 更新 low[2]，更新为 1。



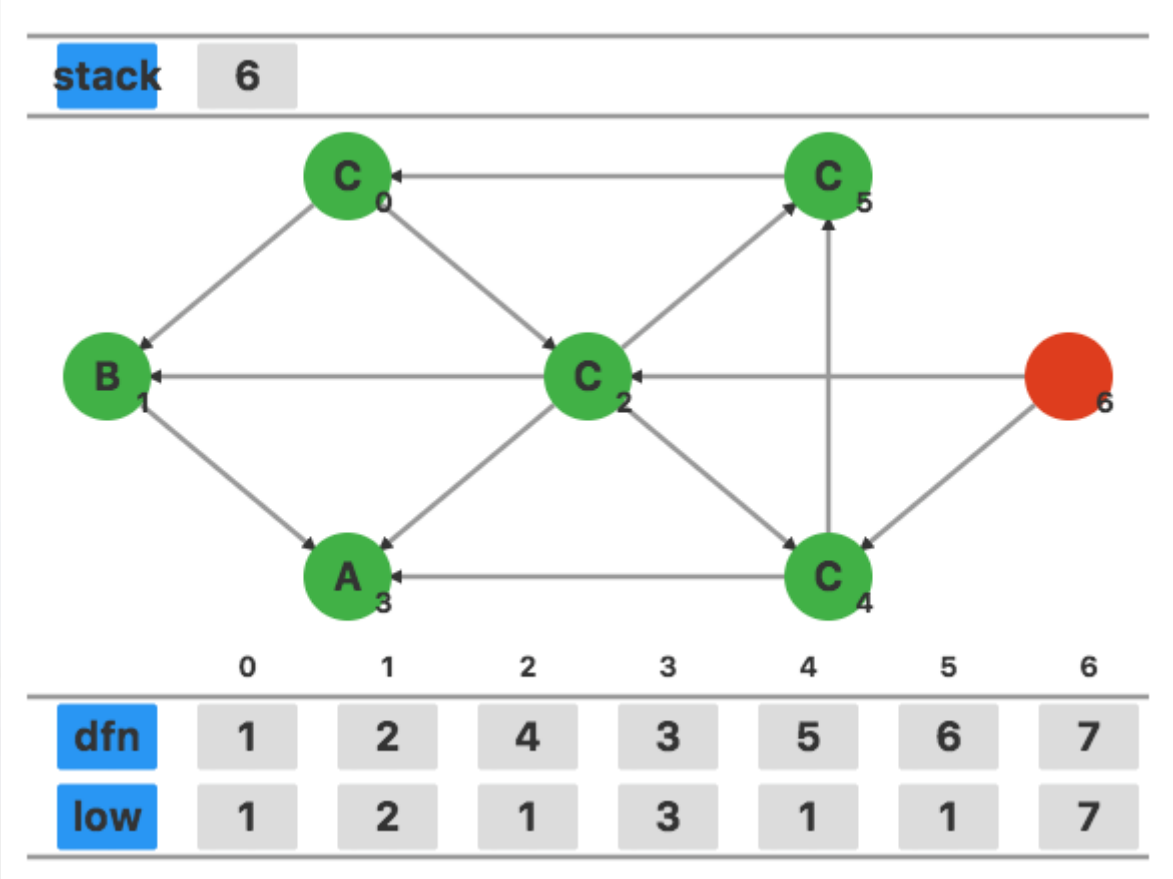
- 13.顶点 2有一个在栈中的相邻顶点5，用 dfn[5] 更新 low[2]，不发生任何变化。



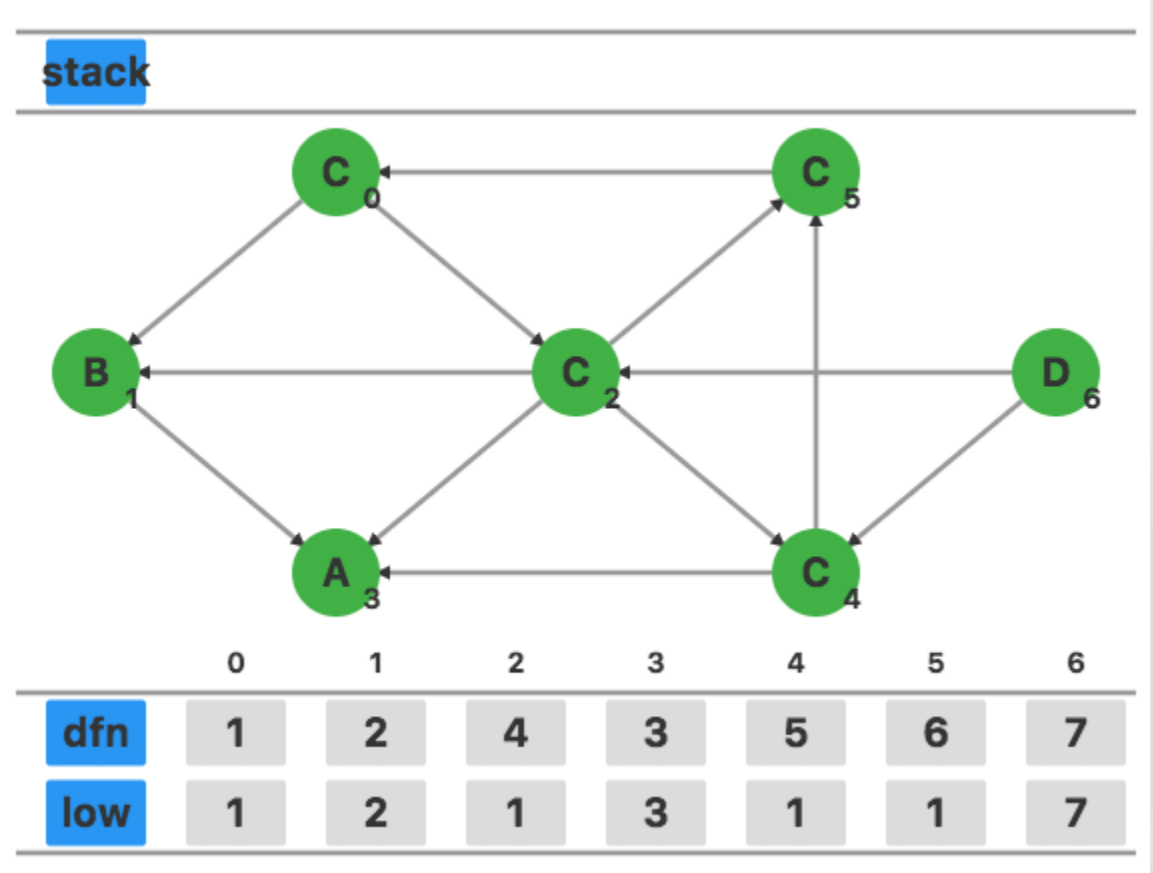
- 14.回溯到顶点 0，用 low[2] 更新 low[0]。此时发现 low[0]=dfn[0]，将栈中元素弹出作为一个新的强连通分量。



- 15.找到下一个未访问的顶点 6，设置 dfn[6]=low[6]=7。



- 16.发现顶点6 没有指向栈中元素的边，搜索结束，将 6作为一个新的强连通分量。



- 计算结束，{3},{1},{0,2,4,5},{6} 是图中的四个强连通分量。

缩点

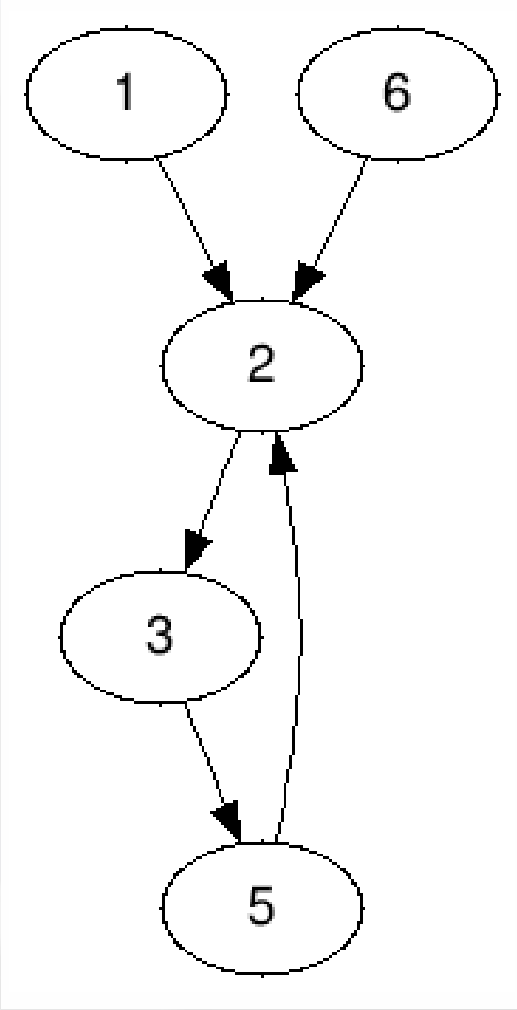
- 在有向图中，缩点（也可称缩环）是指把环（强连通分量）缩成一顶点。借助 Tarjan 求解强连通分量算法，找出每个点所属的强连通分量，再根据原来的有向图建立一个新的有向图，这个有向图是一个有向无环图（DAG）。
- 时间复杂度和 Tarjan 算法一样，是 O(V+E)。

标记每个顶点所属强连通分量的 Tarjan 算法 C++ 示例代码如下：

```cpp
// 使用邻接表存储
const int MAX_N = 10000;
const int MAX_M = 100000;
struct edge {
    int v, next;
} e[MAX_M];
int p[MAX_N], eid;
int dfn[MAX_N], low[MAX_N];
// dfn 中所有元素初始为 0
int idx = 0;  // 时间戳，初始为 0
int belong[MAX_N], scc = 0;
//  belong  数组标记了每个顶点属于哪个强连
//通分量，scc 为强连通分量的总数
int s[MAX_N], top = 0;
// Tarjan 算法中用到的栈
bool in_stack[MAX_N];
// 标记一个顶点是否在栈中
void tarjan(int u) {
    dfn[u] = low[u] = ++idx;
    s[top++] = u;
    in_stack[u] = true;
    for (int i = p[u]; i != -1; i = e[i].next) {
        int v = e[i].v;
        if (!dfn[v]) {
            tarjan(v);
            low[u] = min(low[u], low[v]);
        } else if (in_stack[v]) {
            low[u] = min(low[u], dfn[v]);
        }
    }
    if (dfn[u] == low[u]) {
        ++scc;
        do {
            belong[s[--top]] = scc;
            in_stack[s[top]] = false;
        } while (s[top] != u);
    }
}
```

例题：图的底部

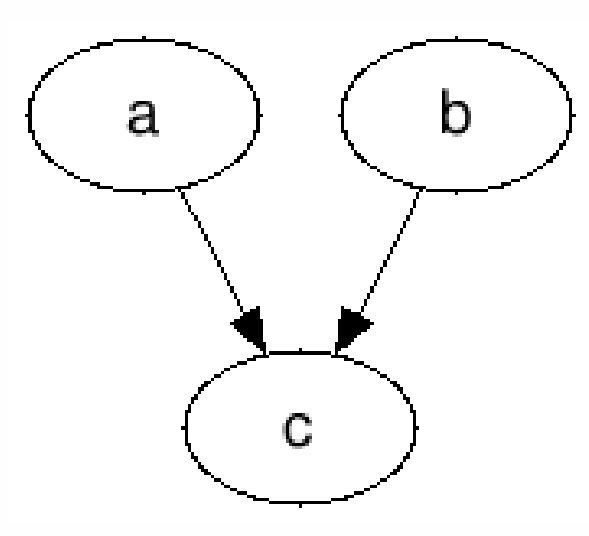
- 给定一个有向图，计算其中有多少个顶点满足：该顶点可达的顶点都可以到达该顶点。例如下面这张图：



- 其中顶点 2,3,5 均满足要求。

解法：

- 在有向图的强连通分量内，每个顶点之间都是互相可达的。那么，我们将这个有向图中的强连通分量缩点以后，找出其中出度为零的所有顶点，算出这些顶点对应原图中的顶点个数就可以了。
- 例如上面这张图，缩点后转化为如下的形式：



- 其中强连通分量 c中包含 3 个顶点。我们发现，缩点后只有一个顶点 c 满足要求，因此答案为 c 中的顶点个数 3。

习题：受欢迎的朋友

- 朋友圈里有很多人。每个人的梦想是成为最受欢迎的朋友。有 N 个人，有 M对二元关系 (A,B)，告诉你A 认为 B 是受欢迎的。如果 A 认为 B 是受欢迎的，B 认为 C 是受欢迎的，则 A 也认为 C 是受欢迎的。你的任务是计算被其余人都认为是受欢迎的朋友数量。
- 输入格式：第一行两个正整数 N 和 M，分别表示一共有 N 个人和 M 对二元关系 (1≤N≤10000,1≤M≤50000)。接下来 M 行，每行两个整数 A 和 B，表示 A 认为 B 是受欢迎的 (1≤A,B≤N)。
- 输出格式: 一行一个整数，表示答案。
- 样例输入
- 4 6
- 1 2
- 1 3
- 1 4
- 2 3
- 2 4
- 3 4
- 样例输出
- 1

习题：商业信息共享

- 有 N 个公司，从每个公司都能单向地向另外一个公司分享最新商业信息，因为他们之间有着某种合作，你需要解决两个问题：
  1. 现在有一个最新的商业信息，至少需要告诉多少个公司，使得所有的公司最终都能得到该信息。
  2. 在原有基础上，至少需要再让多少对公司建立这种合作，使任意一个公司获得某个最新商业信息后，经过若干次分享，所有的公司最终都能得到该信息。
- 输入格式：第一行输入一个整数 N (1≤N≤100)。
- 接下来 N 行，每行若干个整数，表示第 i 个公司可以向哪些公司分享信息，以 0 结束。
- 输出格式：输出共两行，每行一个整数，分别表示问题 1和问题 2 的答案。
- 样例输入
- 6
- 0
- 6 0
- 2 0
- 2 0
- 3 1 0
- 0
- 样例输出
- 2
- 2