**题目列表：**

**[1.复原IP地址](#复原IP地址)**

**[2.验证IP地址](#验证IP地址)**

**[3.找到最终的安全状态](#找到最终的安全状态)**

**[4.连通网络的操作次数](#连通网络的操作次数)**

**[5.统计参与通信的服务器](#统计参与通信的服务器)**

**[6.尽量减少恶意软件的传播](#尽量减少恶意软件的传播)**

**[7.网络延迟时间](#网络延迟时间)**

**[8.处理含限制条件的好友请求](#处理含限制条件的好友请求)**

**[9.编辑距离](#编辑距离)**

**[10.同时运行 N 台电脑的最长时间](#同时运行N台电脑的最长时间)**

1. **复****原IP地址**

有效 IP 地址 正好由四个整数（每个整数位于 0 到 255 之间组成，且不能含有前导 0），整数之间用 '.' 分隔。

例如："0.1.2.201" 和 "192.168.1.1" 是 有效 IP 地址，但是 "0.011.255.245"、"192.168.1.312" 和 "192.168@1.1" 是 无效 IP 地址。

给定一个只包含数字的字符串 s ，用以表示一个 IP 地址，返回所有可能的有效 IP 地址，这些地址可以通过在 s 中插入 '.' 来形成。你 不能 重新排序或删除 s 中的任何数字。你可以按 任何 顺序返回答案。

提示：

1 <= s.length <= 20

s 仅由数字组成

示例 1：

输入：s = "25525511135"

输出：["255.255.11.135","255.255.111.35"]

示例 2：

输入：s = "0000"

输出：["0.0.0.0"]

示例 3：

输入：s = "101023"

输出：["1.0.10.23","1.0.102.3","10.1.0.23","10.10.2.3","101.0.2.3"]

1. **验证IP地址**

给定一个字符串 queryIP。如果是有效的 IPv4 地址，返回 "IPv4" ；如果是有效的 IPv6 地址，返回 "IPv6" ；如果不是上述类型的 IP 地址，返回 "Neither" 。

有效的IPv4地址 是 “x1.x2.x3.x4” 形式的IP地址。 其中 0 <= xi <= 255 且 xi 不能包含 前导零。例如: “192.168.1.1” 、 “192.168.1.0” 为有效IPv4地址， “192.168.01.1” 为无效IPv4地址; “192.168.1.00” 、 “192.168@1.1” 为无效IPv4地址。

一个有效的IPv6地址 是一个格式为“x1:x2:x3:x4:x5:x6:x7:x8” 的IP地址，其中:

* 1 <= xi.length <= 4
* xi 是一个 十六进制字符串 ，可以包含数字、小写英文字母( 'a' 到 'f' )和大写英文字母( 'A' 到 'F' )。
* 在 xi 中允许前导零。
* 例如 "2001:0db8:85a3:0000:0000:8a2e:0370:7334" 和 "2001:db8:85a3:0:0:8A2E:0370:7334" 是有效的 IPv6 地址，而 "2001:0db8:85a3::8A2E:037j:7334" 和 "02001:0db8:85a3:0000:0000:8a2e:0370:7334" 是无效的 IPv6 地址。

提示：

queryIP 仅由英文字母，数字，字符 '.' 和 ':' 组成。

示例 1：

输入：queryIP = "172.16.254.1"

输出："IPv4"

解释：有效的 IPv4 地址，返回 "IPv4"

示例 2：

输入：queryIP = "2001:0db8:85a3:0:0:8A2E:0370:7334"

输出："IPv6"

解释：有效的 IPv6 地址，返回 "IPv6"

示例 3：

输入：queryIP = "256.256.256.256"

输出："Neither"

解释：既不是 IPv4 地址，又不是 IPv6 地址

1. **找到最终的安全状态**

有一个有 n 个节点的有向图，节点按 0 到 n - 1 编号。图由一个 索引从 0 开始 的 2D 整数数组 graph表示， graph[i]是与节点 i 相邻的节点的整数数组，这意味着从节点 i 到 graph[i]中的每个节点都有一条边。

如果一个节点没有连出的有向边，则它是 终端节点 。如果没有出边，则节点为终端节点。如果从该节点开始的所有可能路径都通向 终端节点 ，则该节点为 安全节点 。

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

提示：

n == graph.length

1 <= n <= 104

0 <= graph[i].length <= n

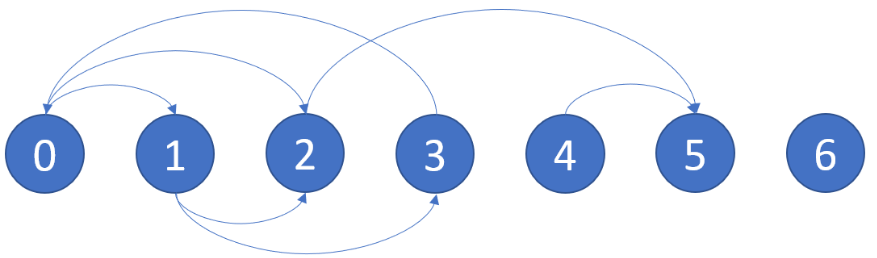
0 <= graph[i][j] <= n - 1

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 \* 104] 内。

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。

节点 5 和节点 6 是终端节点，因为它们都没有出边。

从节点 2、4、5 和 6 开始的所有路径都指向节点 5 或 6 。

示例 2：

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

输出：[4]

解释:

只有节点 4 是终端节点，从节点 4 开始的所有路径都通向节点 4 。

1. **连通网络的操作次数**

用以太网线缆将 n 台计算机连接成一个网络，计算机的编号从 0 到 n-1。线缆用 connections 表示，其中 connections[i] = [a, b] 连接了计算机 a 和 b。

网络中的任何一台计算机都可以通过网络直接或者间接访问同一个网络中其他任意一台计算机。

给你这个计算机网络的初始布线 connections，你可以拔开任意两台直连计算机之间的线缆，并用它连接一对未直连的计算机。请你计算并返回使所有计算机都连通所需的最少操作次数。如果不可能，则返回 -1 。

提示：

1 <= n <= 10^5

1 <= connections.length <= min(n\*(n-1)/2, 10^5)

connections[i].length == 2

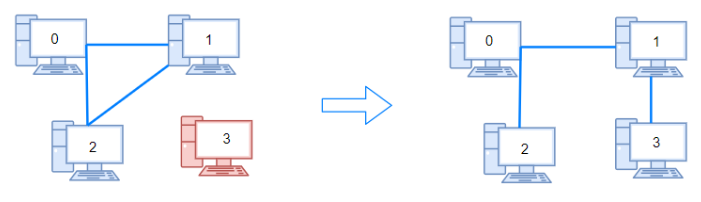
0 <= connections[i][0], connections[i][1] < n

connections[i][0] != connections[i][1]

没有重复的连接。

两台计算机不会通过多条线缆连接。

示例 1：



输入：n = 4, connections = [[0,1],[0,2],[1,2]]

输出：1

解释：拔下计算机 1 和 2 之间的线缆，并将它插到计算机 1 和 3 上。

示例 2：



输入：n = 6, connections = [[0,1],[0,2],[0,3],[1,2],[1,3]]

输出：2

示例 3：

输入：n = 6, connections = [[0,1],[0,2],[0,3],[1,2]]

输出：-1

解释：线缆数量不足。

1. **统计参与通信的服务器**

这里有一幅服务器分布图，服务器的位置标识在 m \* n 的整数矩阵网格 grid 中，1 表示单元格上有服务器，0 表示没有。

如果两台服务器位于同一行或者同一列，我们就认为它们之间可以进行通信。

请你统计并返回能够与至少一台其他服务器进行通信的服务器的数量。

提示：

m == grid.length

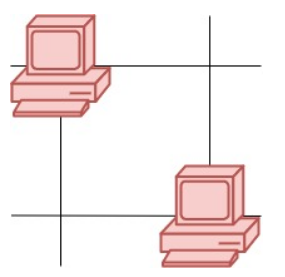
n == grid[i].length

1 <= m <= 250

1 <= n <= 250

grid[i][j] == 0 or 1

示例 1：

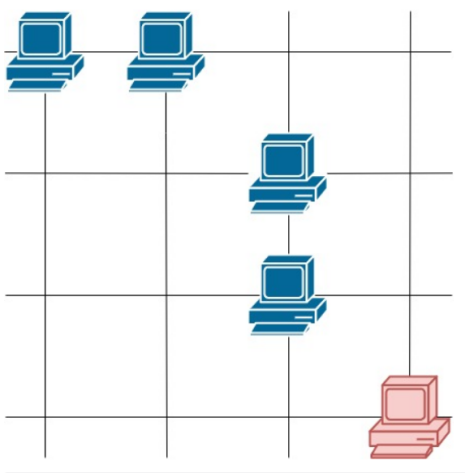


输入：grid = [[1,0],[0,1]]

输出：0

解释：没有一台服务器能与其他服务器进行通信。

示例 2：



输入：grid = [[1,1,0,0],[0,0,1,0],[0,0,1,0],[0,0,0,1]]

输出：4

解释：第一行的两台服务器互相通信，第三列的两台服务器互相通信，但右下角的服务器无法与其他服务器通信。

1. **尽量减少恶意软件的传播**

给出了一个由 n 个节点组成的网络，用 n × n 个邻接矩阵图 graph 表示。在节点网络中，当 graph[i][j] = 1 时，表示节点 i 能够直接连接到另一个节点 j。

一些节点 initial 最初被恶意软件感染。只要两个节点直接连接，且其中至少一个节点受到恶意软件的感染，那么两个节点都将被恶意软件感染。这种恶意软件的传播将继续，直到没有更多的节点可以被这种方式感染。

假设 M(initial) 是在恶意软件停止传播之后，整个网络中感染恶意软件的最终节点数。

如果从 initial 中移除某一节点能够最小化 M(initial)， 返回该节点。如果有多个节点满足条件，就返回索引最小的节点。

请注意，如果某个节点已从受感染节点的列表 initial 中删除，它以后仍有可能因恶意软件传播而受到感染。

提示：

n == graph.length

n == graph[i].length

2 <= n <= 300

graph[i][j] == 0 或 1.

graph[i][j] == graph[j][i]

graph[i][i] == 1

1 <= initial.length <= n

0 <= initial[i] <= n - 1

initial 中所有整数均不重复

示例 1：

输入：graph = [[1,1,0],[1,1,0],[0,0,1]], initial = [0,1]

输出：0

示例 2：

输入：graph = [[1,0,0],[0,1,0],[0,0,1]], initial = [0,2]

输出：0

示例 3：

输入：graph = [[1,1,1],[1,1,1],[1,1,1]], initial = [1,2]

输出：1

1. **网络延迟时间**

有 n 个网络节点，标记为 1 到 n。

给你一个列表 times，表示信号经过 有向 边的传递时间。 times[i] = (ui, vi, wi)，其中 ui 是源节点，vi 是目标节点， wi 是一个信号从源节点传递到目标节点的时间。

现在，从某个节点 K 发出一个信号。需要多久才能使所有节点都收到信号？如果不能使所有节点收到信号，返回 -1 。

提示：

1 <= k <= n <= 100

1 <= times.length <= 6000

times[i].length == 3

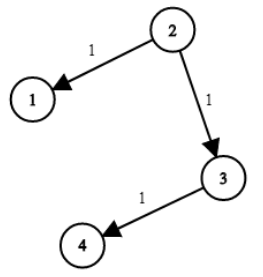
1 <= ui, vi <= n

ui != vi

0 <= wi <= 100

所有 (ui, vi) 对都 互不相同（即，不含重复边）

示例 1：



输入：times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2

输出：2

示例 2：

输入：times = [[1,2,1]], n = 2, k = 1

输出：1

示例 3：

输入：times = [[1,2,1]], n = 2, k = 2

输出：-1

1. **处理含限制条件的好友请求**

给你一个整数 n ，表示网络上的用户数目。每个用户按从 0 到 n - 1 进行编号。

给你一个下标从 0 开始的二维整数数组 restrictions ，其中 restrictions[i] = [xi, yi] 意味着用户 xi 和用户 yi 不能 成为 朋友 ，不管是 直接 还是通过其他用户 间接 。

最初，用户里没有人是其他用户的朋友。给你一个下标从 0 开始的二维整数数组 requests 表示好友请求的列表，其中 requests[j] = [uj, vj] 是用户 uj 和用户 vj 之间的一条好友请求。

如果 uj 和 vj 可以成为 朋友 ，那么好友请求将会 成功 。每个好友请求都会按列表中给出的顺序进行处理（即，requests[j] 会在 requests[j + 1] 前）。一旦请求成功，那么对所有未来的好友请求而言， uj 和 vj 将会 成为直接朋友 。

返回一个 布尔数组 result ，其中元素遵循此规则：如果第 j 个好友请求 成功 ，那么 result[j] 就是 true ；否则，为 false 。

注意：如果 uj 和 vj 已经是直接朋友，那么他们之间的请求将仍然 成功 。

提示：

2 <= n <= 1000

0 <= restrictions.length <= 1000

restrictions[i].length == 2

0 <= xi, yi <= n - 1

xi != yi

1 <= requests.length <= 1000

requests[j].length == 2

0 <= uj, vj <= n - 1

uj != vj

示例 1：

输入：n = 3, restrictions = [[0,1]], requests = [[0,2],[2,1]]

输出：[true,false]

解释：

请求 0 ：用户 0 和 用户 2 可以成为朋友，所以他们成为直接朋友。

请求 1 ：用户 2 和 用户 1 不能成为朋友，因为这会使 用户 0 和 用户 1 成为间接朋友 (1--2--0) 。

示例 2：

输入：n = 3, restrictions = [[0,1]], requests = [[1,2],[0,2]]

输出：[true,false]

解释：

请求 0 ：用户 1 和 用户 2 可以成为朋友，所以他们成为直接朋友。

请求 1 ：用户 0 和 用户 2 不能成为朋友，因为这会使 用户 0 和 用户 1 成为间接朋友 (0--2--1) 。

1. **编辑距离**

给你两个单词 word1 和 word2， 请返回将 word1 转换成 word2 所使用的最少操作数  。

你可以对一个单词进行如下三种操作：

插入一个字符

删除一个字符

替换一个字符

提示：

0 <= word1.length, word2.length <= 500

word1 和 word2 由小写英文字母组成

示例 1：

输入：word1 = "horse", word2 = "ros"

输出：3

解释：

horse -> rorse (将 'h' 替换为 'r')

rorse -> rose (删除 'r')

rose -> ros (删除 'e')

示例 2：

输入：word1 = "intention", word2 = "execution"

输出：5

解释：

intention -> inention (删除 't')

inention -> enention (将 'i' 替换为 'e')

enention -> exention (将 'n' 替换为 'x')

exention -> exection (将 'n' 替换为 'c')

exection -> execution (插入 'u')

1. **同时运行 N 台电脑的最长时间**

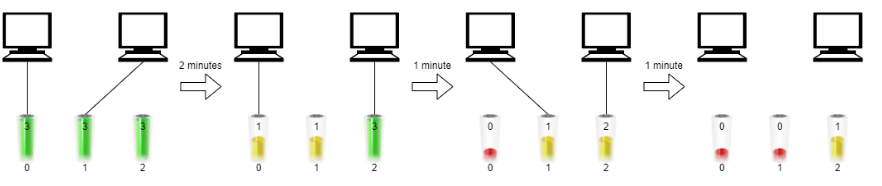
你有 n 台电脑。给你整数 n 和一个下标从 0 开始的整数数组 batteries ，其中第 i 个电池可以让一台电脑 运行 batteries[i] 分钟。你想使用这些电池让 全部 n 台电脑 同时 运行。

一开始，你可以给每台电脑连接 至多一个电池 。然后在任意整数时刻，你都可以将一台电脑与它的电池断开连接，并连接另一个电池，你可以进行这个操作 任意次 。新连接的电池可以是一个全新的电池，也可以是别的电脑用过的电池。断开连接和连接新的电池不会花费任何时间。

注意，你不能给电池充电。

请你返回你可以让 n 台电脑同时运行的 最长 分钟数。

示例 1：



输入：n = 2, batteries = [3,3,3]

输出：4

解释：

一开始，将第一台电脑与电池 0 连接，第二台电脑与电池 1 连接。

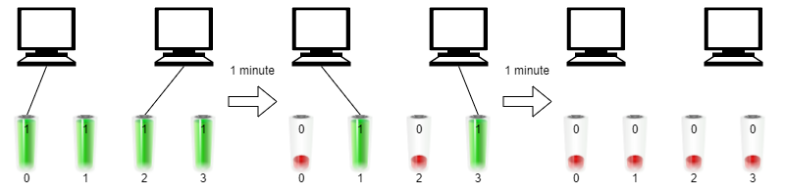
2 分钟后，将第二台电脑与电池 1 断开连接，并连接电池 2 。注意，电池 0 还可以供电 1 分钟。

在第 3 分钟结尾，你需要将第一台电脑与电池 0 断开连接，然后连接电池 1 。

在第 4 分钟结尾，电池 1 也被耗尽，第一台电脑无法继续运行。

我们最多能同时让两台电脑同时运行 4 分钟，所以我们返回 4 。

示例 2：



输入：n = 2, batteries = [1,1,1,1]

输出：2

解释：

一开始，将第一台电脑与电池 0 连接，第二台电脑与电池 2 连接。

一分钟后，电池 0 和电池 2 同时耗尽，所以你需要将它们断开连接，并将电池 1 和第一台电脑连接，电池 3 和第二台电脑连接。

1 分钟后，电池 1 和电池 3 也耗尽了，所以两台电脑都无法继续运行。

我们最多能让两台电脑同时运行 2 分钟，所以我们返回 2 。