**Union Find Al rithm**

**128. Longest Consecutive Sequence.**

Given an unsorted array of integers, find the length of the longest consecutive elements sequence. Your algorithm should run in O(n) complexity.

**Example:**

**Input:** [100, 4, 200, 1, 3, 2]

**Output:** 4

**Explanation:** The longest consecutive elements sequence is [1, 2, 3, 4].

Therefore, its length is 4.

**130. Surrounded Regions**

Given a 2D board containing 'X' and 'O' (the letter O), capture all regions surrounded by 'X' .

A region is captured by flipping all 'O' s into 'X' s in that surrounded region.

**Example:**

**Input:**

X X X X

X O O X

X X O X

X O X X

**Output:**

X X X X

X X X X

X X X X

X O X X

**Explanation:**

Surrounded regions should not be on the border, which means that any 'O' on the border of the board are not flipped to 'X'. Any 'O' that is not on the border and it is not connected to an 'O' on the border will be flipped to 'X'. Two cells are connected if they are adjacent cells connected horizontally or vertically.

**200. Number of Islands**

Given a 2d grid map of '1' s (land) and '0' s (water), count the number of islands. An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

**Example 1:**

**Input:**

11110

11010

11000

00000

**Output:** 1

**Example 2:**

**Input:**

11000

11000

00100

00011

**Output:** 3

**399. Evaluate Division**

Equations are given in the format A / B = k , where A and B are variables represented as strings, and k is a real number (floating point number). Given some queries, return the answers. If the answer does not exist, return -1.0 .

Given a / b = 2.0, b / c = 3.0. queries are: a / c = ?, b / a = ?, a / e = ?, a / a = ?, x / x = ? .

return [6.0, 0.5, -1.0, 1.0, -1.0 ].

**The input is:** vector<pair<string, string>> equations, vector<double>& values, vector<pair<string, string>> queries , where equations.size() == values.size() , and the values are positive. This represents the equations. Return vector<double>.

**According to the example above:**

equations = [ ["a", "b"], ["b", "c"] ],

values = [2.0, 3.0],

queries = [ ["a", "c"], ["b", "a"], ["a", "e"], ["a", "a"], ["x", "x"] ].

The input is always valid. You may assume that evaluating the queries will result in no division by zero and there is no contradiction.

**547. Friend Circles**

There are N students in a class. Some of them are friends, while some are not. Their friendship is transitive in nature. For example, if A is a direct friend of B and B is a direct friend of C, then A is an indirect friend of C. And we defined a friend circle is a group of students who are direct or indirect friends.

Given a N\*N matrix M representing the friend relationship between students in the class. If M[i] [j] = 1, then the ith and jth students are direct friends with each other, otherwise not. And you have to output the total number of friend circles among all the students.

**Example 1:**

**Input:**

[[1,1,0],

[1,1,0],

[0,0,1]]

Output: 2

**Explanation:** The 0th and 1st students are direct friends, so they are in a friend circle.

The 2nd student himself is in a friend circle. So return 2.

**Example 2:**

**Input:**

[[1,1,0],

[1,1,1],

[0,1,1]]

Output: 1

**Explanation:** The 0th and 1st students are direct friends; the 1st and 2nd students are direct friends, so the 0th and 2nd students are indirect friends. All of them are in the same friend circle, so return 1.

**Note:**

1. N is in range [1,200].

2. M[i][i] = 1 for all students.

3. If M[i][j] = 1, then M[j][i] = 1.

**684. Redundant Connection**

In this problem, a tree is an undirected graph that is connected and has no cycles.

The given input is a graph that started as a tree with N nodes (with distinct values 1, 2, ..., N), with one additional edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of edges . Each element of edges is a pair [u, v] with u < v , that represents an undirected edge connecting nodes u and v .

Return an edge that can be removed so that the resulting graph is a tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array. The answer edge [u, v] should be in the same format, with u < v .

**Example 1:**

**Input:** [[1,2], [1,3], [2,3]]

**Output:** [2,3]

**Explanation:** The given undirected graph will be like this:

1

/ \

2 - 3

**Example 2:**

**Input:** [[1, 2], [2, 3], [3,4], [1,4], [1,5]]

**Output:** [1, 4]

**Explanation:** The given undirected graph will be like this:

5 - 1 - 2

| |

4 - 3

**Note:**

* The size of the input 2D-array will be between 3 and 1000.
* Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

**685. Redundant Connection II**

In this problem, a rooted tree is a directed graph such that, there is exactly one node (the root) for which all other nodes are descendants of this node, plus every node has exactly one parent, except for the root node which has no parents.

The given input is a directed graph that started as a rooted tree with N nodes (with distinct values 1, 2, ..., N), with one additional directed edge added. The added edge has two different vertices chosen from 1 to N, and was not an edge that already existed.

The resulting graph is given as a 2D-array of edges . Each element of edges is a pair [u, v] that represents a directed edge connecting nodes u and v , where u is a parent of child v .

Return an edge that can be removed so that the resulting graph is a rooted tree of N nodes. If there are multiple answers, return the answer that occurs last in the given 2D-array.

**Example 1:**

Input: [[1,2], [1,3], [2,3]]

Output: [2,3]

Explanation: The given directed graph will be like this:

1

/ \

v v

2->3

**Example 2:**

Input: [[1,2], [2,3], [3,4], [4,1], [1,5]]

Output: [4,1]

Explanation: The given directed graph will be like this:

5 <- 1 -> 2

^ |

| v

4 <- 3

**Note:**

The size of the input 2D-array will be between 3 and 1000.

Every integer represented in the 2D-array will be between 1 and N, where N is the size of the input array.

**721. Accounts Merge**

Given a list accounts, each element accounts[i] is a list of strings, where the first element accounts[i][0] is a name, and the rest of the elements are emails representing emails of the account.

Now, we would like to merge these accounts. Two accounts definitely belong to the same person if there is some email that is common to both accounts. Note that even if two accounts have the same name, they may belong to different people as people could have the same name. A person can have any number of accounts initially, but all of their accounts definitely have the same name.

After merging the accounts, return the accounts in the following format: the first element of each account is the name, and the rest of the elements are emails in sorted order. The accounts themselves can be returned in any order.

**Example 1:**

**Input:**

accounts = [["John", "johnsmith@mail.com", "john00@mail.com"],

["John", "johnnybravo@mail.com"],

["John", "johnsmith@mail.com", "john\_newyork@mail.com"],

["Mary", "mary@mail.com"]]

**Output:**

[["John", 'john00@mail.com', 'john\_newyork@mail.com', 'johnsmith@mail.com'],

["John", "johnnybravo@mail.com"], ["Mary", "mary@mail.com"]]

**Explanation:**

The first and third John's are the same person as they have the common email "johnsmith@mail.com".

The second John and Mary are different people as none of their email addresses are used by other accounts.

We could return these lists in any order, for example the answer [['Mary', 'mary@mail.com'], ['John', 'johnnybravo@mail.com'], ['John', 'john00@mail.com', 'john\_newyork@mail.com', 'johnsmith@mail.com']] would still be accepted.

**Note:**

The length of accounts will be in the range [1, 1000] .

The length of accounts[i] will be in the range [1, 10] .

The length of accounts[i][j] will be in the range [1, 30] .

**765. Couples Holding Hands**

778. Swim in Rising Water

803. Bricks Falling When Hit

839. Similar String Groups

924. Minimize Malware Spread

928. Minimize Malware Spread II

947. Most Stones Removed with Same Row or Column

952. Largest Component Size by Common Factor

959. Regions Cut By Slashes

990. Satisfiability of Equality Equations