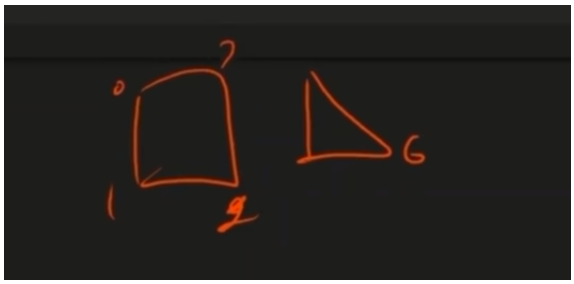
Graphs Level Up

05 September 2022 16:46

1. Heaviest Path in Terms of Weight



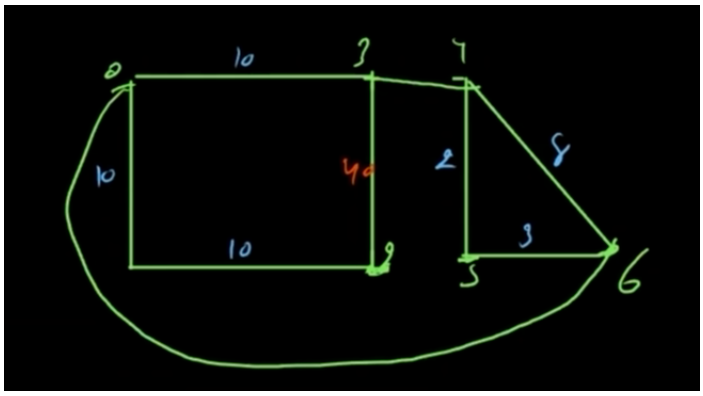
Agar malne destinaiton 6 hai aur agar graph aisa hua to mai kabhi bhi 6 tak pahunch ne nhi paunga, tabhi recAns.wsf != -1 ka check add kiya hai.

```
// b <============== Heaviest Path in terms of weight ============>

// ? Faith ye lagaya ki mujhe mere sare neighbours apne se pass hota hua
// ? destination tak ka heaviest path nikal ke leke aayenge. Ab mai unhe compare
// ? karke sabse max value ko store kar linga aur return kar dunga

public static class Pair {
    int wsf = -1; // Weight so far
    String psf = ""; // Path so far

    Pair() {

    }

    Pair(int wsf, String psf) {
        this.wsf = wsf;
        this.psf = psf;
    }
}
```

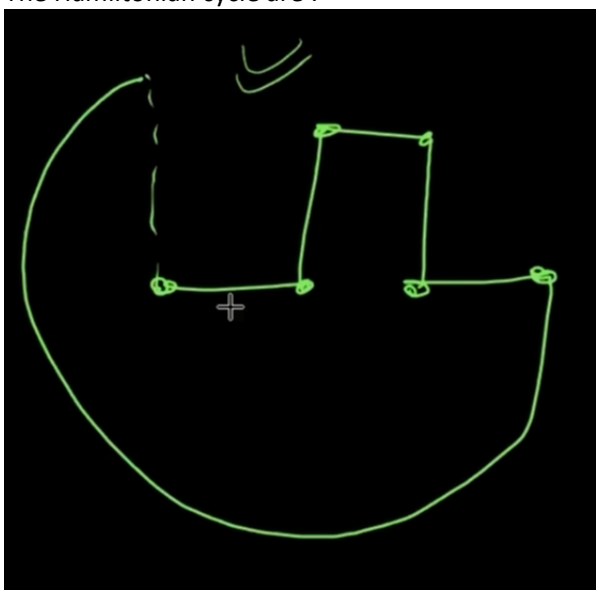
```
public static Pair heaviestPath(ArrayList<Edge>[] graph, int src, int dest, boolean[] vis) {

    if (src == dest) {
        return new Pair(wsf: 0, "" + src);
    }

    vis[src] = true;
    Pair ans = new Pair();
    for (Edge e : graph[src]) {
        if (!vis[e.v]) {
            Pair recAns = heaviestPath(graph, e.v, dest, vis);

            if (recAns.wsf != -1 && recAns.wsf + e.w > ans.wsf) { // recAns.wsf != -1 == > ye check isiliye lagaya
                                                                  // kyunki manle kabhi agar destination mila he
                                                                  // nhi, to use answer calculate karna he nhi
                                                                  // chahiye
                ans.wsf = recAns.wsf + e.w;
                ans.psf = src + recAns.psf;
            }
        }
    }

    vis[src] = false;
    return ans;
}
```

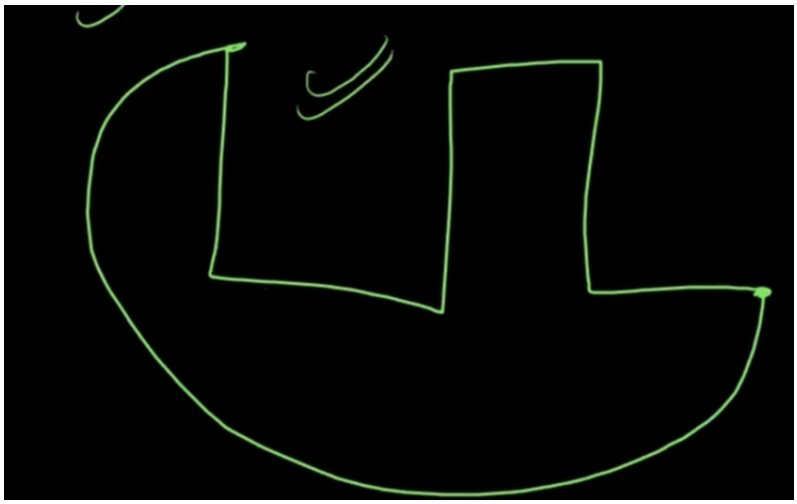
2. Hamiltonian Path and Cycle (Single Source Algorithms)

Note -> A hamiltonian path is such which visits all vertices without visiting any twice. A hamiltonian path becomes a cycle if there is an edge between first and last vertex.

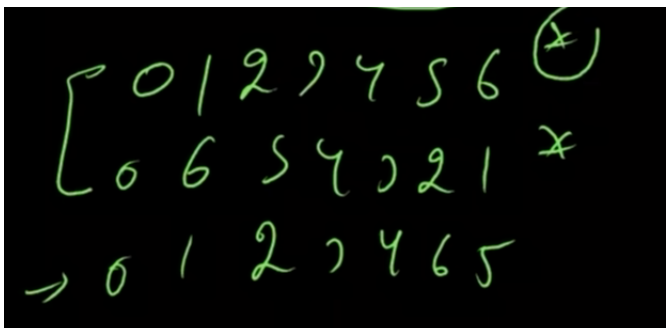


The Hamiltonian cycle are : -





Basically agar har hamiltonian path ke liye agar uske last vertex aur original source ke beech edge exist karta hai to use hamiltonian cycle bolte hain

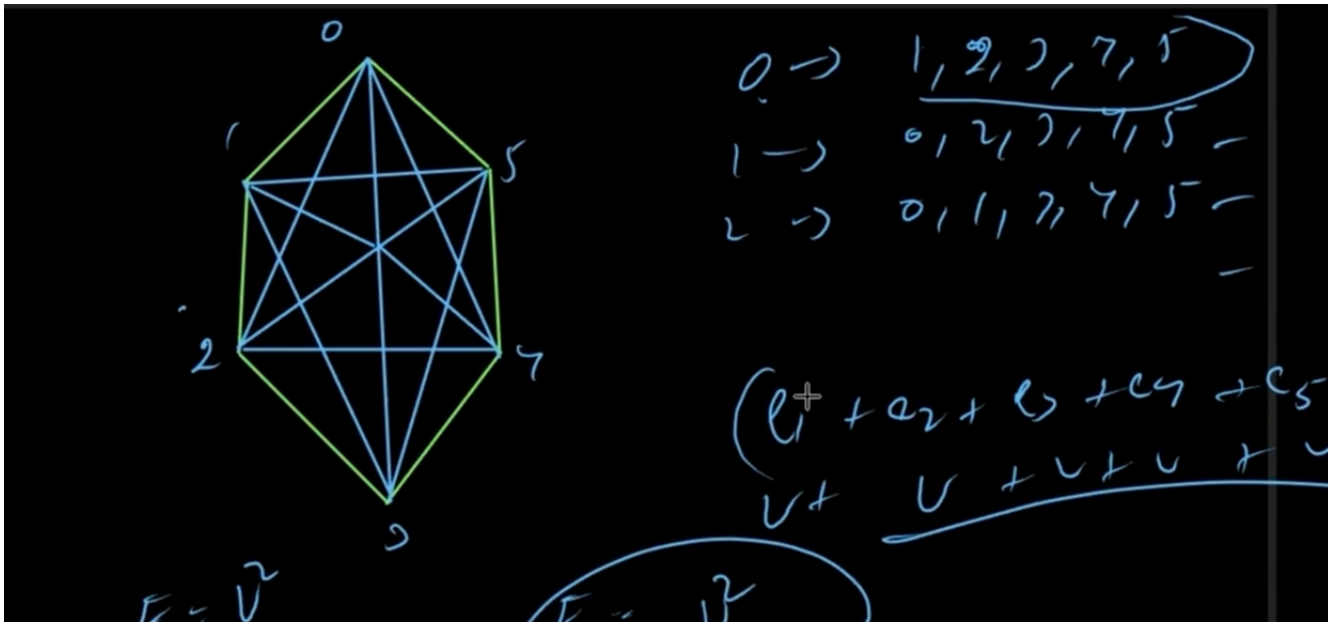


The first two are cycle and the last one is the path.

```java
// b <=============== Hamiltonian Path and Cycle ==========================>
// # Also know as single source algorithm since it is source dependent and the
// # answer can change w.r.t to source.

// A hamiltonian path is such which visits all vertices without visiting any
// twice. A hamiltonian path becomes a cycle if there is an edge between first
// and last vertex.

public static boolean hasHamiltonianCycle(ArrayList<Edge>[] graph, int src, int dest) {

    for (Edge e : graph[src]) {
        if (e.v == dest) {
            return true;
        }
    }

    return false;
}
```

```java
// osrc ==> original src to check the hamiltonian cycle.
// Agar mere total number of edges is total vertex - 1 so then only i can that i
// have traversed all the vertices.

// E=V-1;
public static void hamiltonianPathAndCycle(ArrayList<Edge>[] graph, int src, int osrc, boolean[] vis,
        int edgeCount, String psf) {

    if (edgeCount == graph.length - 1) {

        if (hasHamiltonianCycle(graph, osrc, src)) {
            System.out.println(psf + src + "*"); // Cycle
        } else {
            System.out.println(psf + src + "."); // Path
        }
    }

    vis[src] = true;

    for (Edge e : graph[src]) {
        if (!vis[e.v]) {
            hamiltonianPathAndCycle(graph, e.v, osrc, vis, edgeCount + 1, psf + src);
        }
    }
    vis[src] = false;
}
```

3. Complexity of a dense graph



Since we will be traversing all the edges of every vertex, therefore e1 + e2 + e3 + e4 + ... + ev-1.

But if we see closely, every vertex is connected to every other vertex. So e1=e2=e3=e4=V(Total number of vertex)

So her vertex ke liye v-1 edges traverse kar rahe hain. So total edge will be V^2.

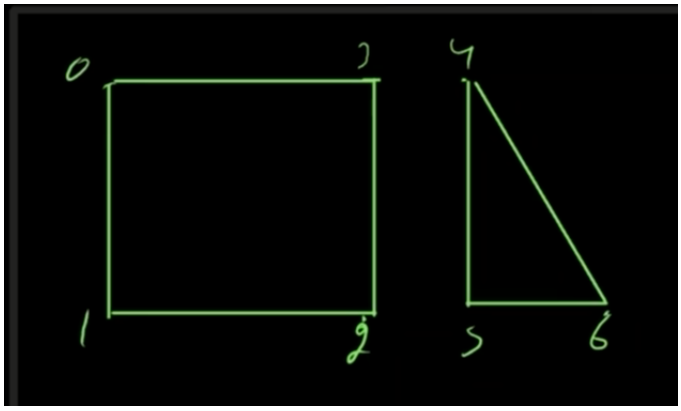
O(E) ==> $O(V^2)$

4. Just Random Information

   Hashmap is good if the the number of elements is less than 10^3. After that the number of collission increase in it, hence increasing the complexity.

   So hashmap O(1) can be equal to O(n).

5. Get Connected Components




```java
// b <==============Get Connected Components=============>

// O(E)
public static void dfs_compo(ArrayList<Edge>[] graph, int src, boolean[] vis) {

    vis[src] = true;

    for (Edge e : graph[src]) {
        if (!vis[e.v]) {
            dfs_compo(graph, e.v, vis);
        }
    }
}

// Get Connected Components
public static int gcc(ArrayList<Edge>[] graph) {
    int N = graph.length;
    boolean[] vis = new boolean[N];

    int components = 0;

    // This for loop is of complexity O(V + E) since all the vertices are traversed
    // and all the edges are traversed by dfs_compo for loop.
    for (int i = 0; i < N; i++) {
        if (!vis[i]) {
            components++;
            dfs_compo(graph, i, vis);
        }
    }
```

```
    return components;
}
```

6. Number of Islands

200. Number of Islands

Medium 16779 386 Add to List Share

Given an `m x n` 2D binary grid `grid` which represents a map of `'1'`s (land) and `'0'`s (water), return *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: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

Example 2:

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 300`
- `grid[i][j]` is `'0'` or `'1'`.

```
// b <============== Number of Islands ======================>
// https://leetcode.com/problems/number-of-islands/

// This is same as get connected compoenents.
// The whole one component is visited from one dfs call and we have to get all
// the components.

public static void dfs_NumberOfIslands(char[][] grid, boolean[][] vis, int[][] dir, int sr, int sc) {

    vis[sr][sc] = true;

    for (int d = 0; d < dir.length; d++) {
        int r = sr + dir[d][0];
        int c = sc + dir[d][1];

        if (r >= 0 && c >= 0 && r <= grid.length - 1 && c <= grid[0].length - 1 && grid[r][c] != '0'
            && !vis[r][c]) {
            dfs_NumberOfIslands(grid, vis, dir, r, c);
        }
    }
}
```

```
public int numIslands(char[][] grid) {
```

```
    int n = grid.length;
    int m = grid[0].length;

    int[][] dir = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };
    boolean[][] vis = new boolean[n][m]; // This is used since we should not modify the original data. Otherwise, we
                                         // would have to make changes in the original grid array.

    int numberOfIslands = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (!vis[i][j] && grid[i][j] != '0') {
                numberOfIslands++;
                dfs_NumberOfIslands(grid, vis, dir, i, j);
            }
        }
    }

    return numberOfIslands;
}
```

7. Max Area Of Island

695. Max Area of Island

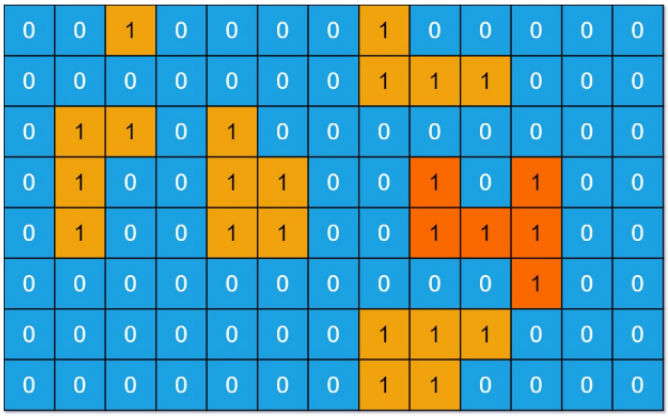
Medium 7815 175 Add to List Share

You are given an `m x n` binary matrix `grid`. An island is a group of `1`'s (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The **area** of an island is the number of cells with a value `1` in the island.

Return *the maximum **area** of an island in* `grid`. If there is no island, return `0`.

Example 1:




```
Input: grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,1,1,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,1,1,0,0,1,0,1,0,0],[0,1,0,0,1,1,0,0,1,1,1,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,0,1,1,0,0,0,0]]
Output: 6
Explanation: The answer is not 11, because the island must be connected 4-
directionally.
```

Example 2:

```
Input: grid = [[0,0,0,0,0,0,0,0]]
Output: 0
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 50`

- `grid[i][j]` is either `0` or `1`.

```java
// b <============== 695. Max Area of Island ================>
// https://leetcode.com/problems/max-area-of-island/

// Same as the number of islands but here we also have to calculat the area of
// each component and then find the maximum area.

// ? Faith ye rakha maine apne neighbours ko bola ki tum mujhe apna area aur jo
// ? tumse touched one hain unka area lake dedo. Ab mai total area nikalne ke
//
// liye
// ? jo total area mere neighours se aaya hai mai usme 1 add kar dunga.

public static int dfs_maxAreaOfIsland(int[][] grid, boolean[][] vis, int[][] dir, int sr, int sc) {

    vis[sr][sc] = true;

    int count = 0;

    for (int d = 0; d < dir.length; d++) {
        int r = sr + dir[d][0];
        int c = sc + dir[d][1];

        if (r >= 0 && c >= 0 && r <= grid.length - 1 && c <= grid[0].length - 1 && grid[r][c] != 0
                && !vis[r][c]) {
            count += dfs_maxAreaOfIsland(grid, vis, dir, r, c);
        }
    }

    return count + 1;
}
```

```java
public int maxAreaOfIsland(int[][] grid) {

    int n = grid.length;
    int m = grid[0].length;

    int[][] dir = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };
    boolean[][] vis = new boolean[n][m]; // This is used since we should not modify the original data. Otherwise,
                                         // we would have to make changes in the original grid array.

    int maxArea = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (!vis[i][j] && grid[i][j] != 0) {
                int area = dfs_maxAreaOfIsland(grid, vis, dir, i, j);
                maxArea = Math.max(maxArea, area);
            }
        }
    }

    return maxArea;
}
```

8. Perimeter of the Island

463. Island Perimeter

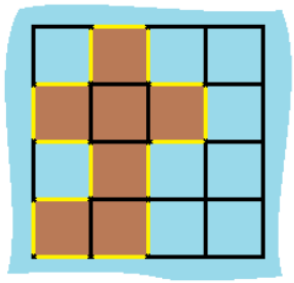
Easy 4873 254 Add to List Share

You are given `row x col` `grid` representing a map where `grid[i][j] = 1` represents land and `grid[i][j] = 0` represents water.

Grid cells are connected **horizontally/vertically** (not diagonally). The `grid` is completely surrounded by water, and there is exactly one island (i.e., one or more connected land cells).

The island doesn't have "lakes", meaning the water inside isn't connected to the water around the island. One cell is a square with side length 1. The grid is rectangular, width and height don't exceed 100. Determine the perimeter of the island.

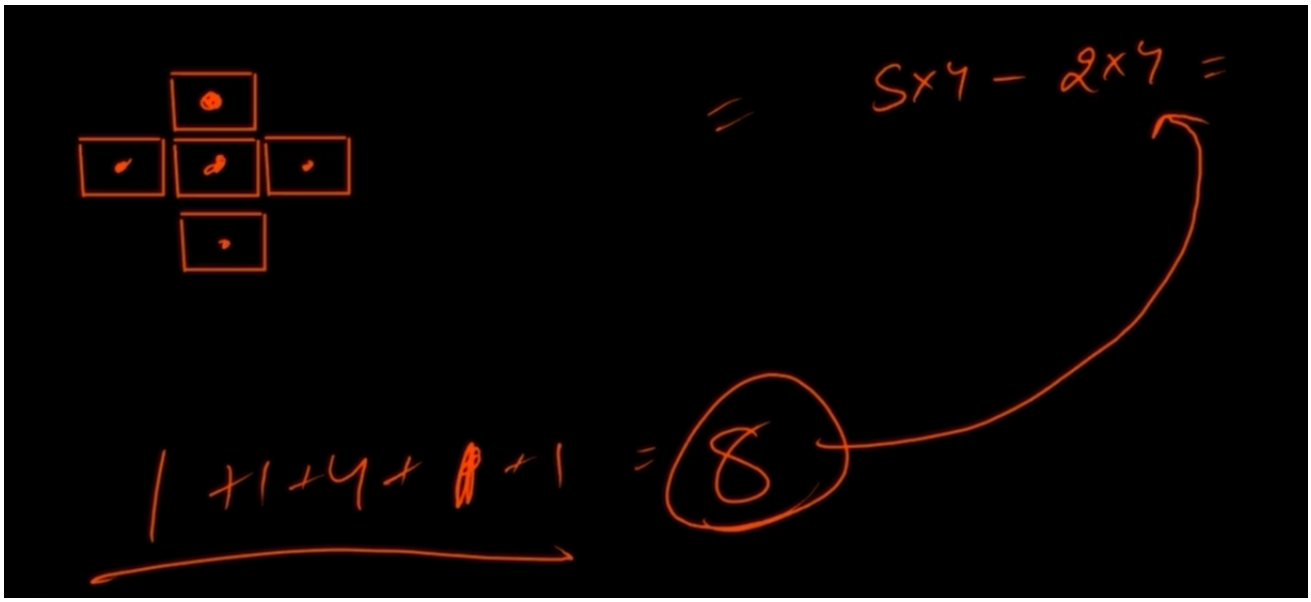
Example 1:



```
Input: grid = [[0,1,0,0],[1,1,1,0],[0,1,0,0],[1,1,0,0]]
Output: 16
Explanation: The perimeter is the 16 yellow stripes in the image above.
```

Example 2:

```
Input: grid = [[1]]
Output: 4
```




```
// b <================ 463. Island Perimeter ===============>
// https://leetcode.com/problems/island-perimeter/

// Yahan pe humne basically simple logic lagaya hai. Agar humpe 1 square hai to
// `uska perimeter will be 4. Agar hume 2 seperate squares ha, to un donno ka
// perimeter will be 2*4=8.

// Per agar hum dono ko merge kardein, then the two edges perimeter will be
// gone. So total will be 2*4 -2(one edge for each sqaures). There will be total
// 2 neighbours 1 for each square.

// So what we have done is counted the total number of squares that have 1 as
// their value denoting land. Now whenever we find 1, we check the number of
// neighbours of the square and add them.

// At last, we calcalated 4 * oneCount - nbrCount to get the total perimeter.
```

```
public int islandPerimeter(int[][] grid) {

    int n = grid.length, m = grid[0].length, oneCount = 0, nbrCount = 0;
    int[][] dir = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1) {
                oneCount++;
                for (int d = 0; d < dir.length; d++) {
```

```
                int r = i + dir[d][0];
                int c = j + dir[d][1];

                if (r >= 0 && c >= 0 && r < n && c < m && grid[r][c] == 1)
                    nbrCount++;
            }
        }
    }

    return 4 * oneCount - nbrCount;
}
```

9. Surrounded Regions

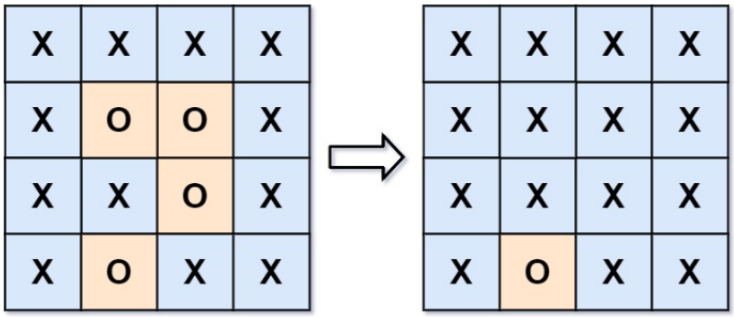
130. Surrounded Regions

Medium 5665 1336 Add to List Share

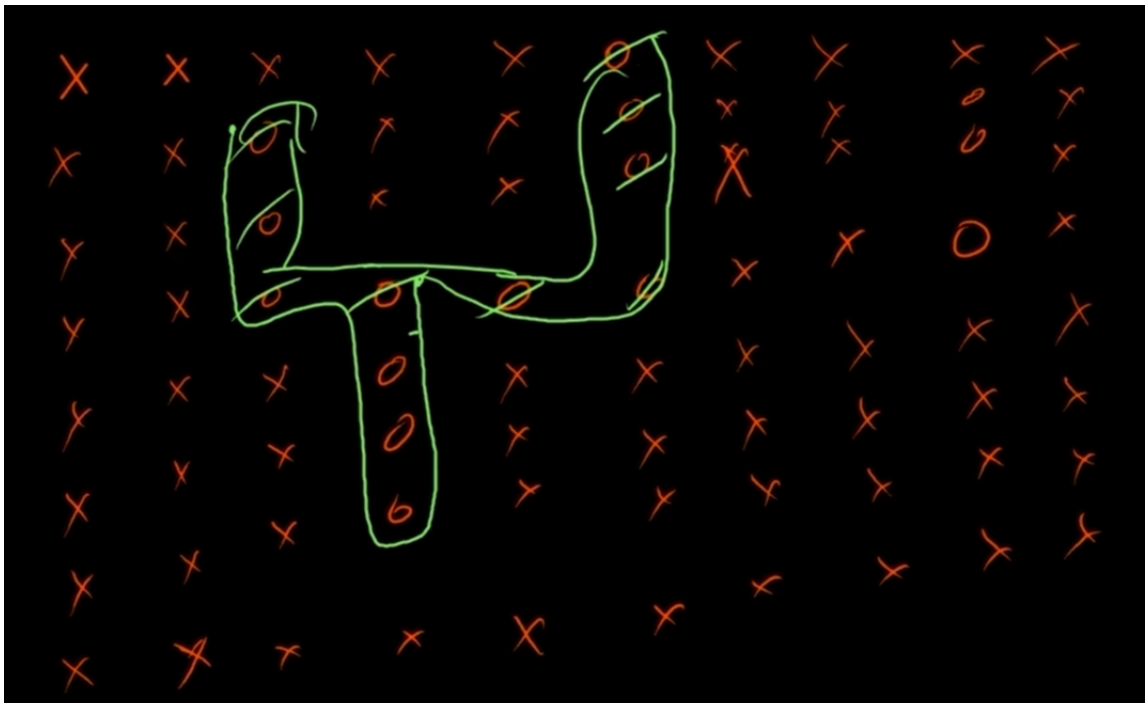
Given an `m x n` matrix `board` containing `'X'` and `'O'`, *capture all regions that are 4-directionally surrounded by* `'X'`.

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

Example 1:




```
Input: board = [["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: Notice that an 'O' should not be flipped if:
- It is on the border, or
- It is adjacent to an 'O' that should not be flipped.
The bottom 'O' is on the border, so it is not flipped.
The other three 'O' form a surrounded region, so they are flipped.
```



```java
// b <============= 130. Surrounded Regions ======================>
// https://leetcode.com/problems/surrounded-regions/

// Agar koi basically boundary waale O se connected O hai, wo to kabhi bhi
// surround to ho he nhi payega.

// ? To hum yahan pe boundary wale O se call lagate hain. Per humne center wale
// ? O jo O Beech mai hain unse call kyun nhi lagayienn ???

// Iska answer ye hain ki agar mai manle center se call lagake boundary wale O
// tak pahunch gaya, to backtrack karte time hum un O ko mark karenge ki wo
// ` boundary wale O se connected the. Per wapis mark karte time jo already
// ` backtrack ho chuke hain (matlab backtrack karne ke baad boundary wale O tak
// puhunche), unhe mark karna hum miss kar denge. Isiliye hum boundary wale O se
// call lagate hain bas.
```

```java
// To kiya ye ki jo bhi sare O jo boundary wale O se connected hain unhe $ mark
// kiya kyunki wo to acpure nhi ho sakte. Ab jo O iske baad bach gaye wo sare
// capture ho sakte hain isiliye unhe badme X mai convert kiya aur $ ko O mai
// dubara convert kar diya

public static void dfs_Surrounded(char[][] grid, int[][] dir, int sr, int sc) {

    grid[sr][sc] = '$';

    for (int d = 0; d < dir.length; d++) {
        int r = sr + dir[d][0];
        int c = sc + dir[d][1];

        if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 'O') {
            dfs_Surrounded(grid, dir, r, c);
        }
    }
}
```

```java
public void solve(char[][] grid) {

    int n = grid.length, m = grid[0].length;
    int[][] dir = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (i == 0 || j == 0 || i == n - 1 || j == m - 1) {
                if (grid[i][j] == 'O') {
                    dfs_Surrounded(grid, dir, i, j);
                }

            }
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == '$') {
                grid[i][j] = 'O';
            } else if (grid[i][j] == 'O') {
                grid[i][j] = 'X';
            }
        }
    }
}
```

10. Number of Distinct Islands

694. Number of Distinct Islands

Medium 1397 84 Add to List Share

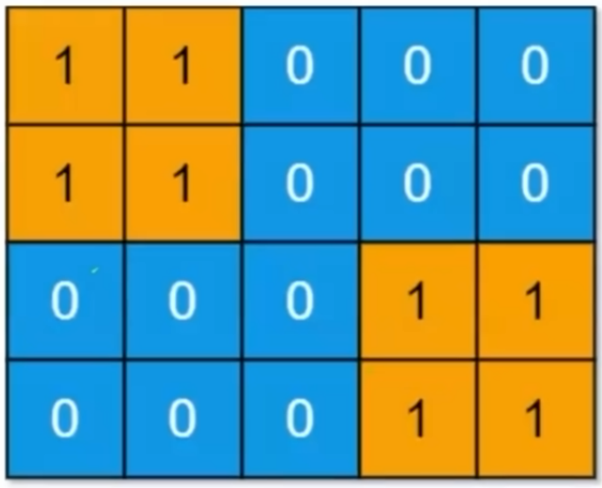
You are given an `m x n` binary matrix `grid`. An island is a group of `1`'s (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

An island is considered to be the same as another if and only if one island can be translated (and not rotated

An island is considered to be the same as another if and only if one island can be translated (and not rotated or reflected) to equal the other.

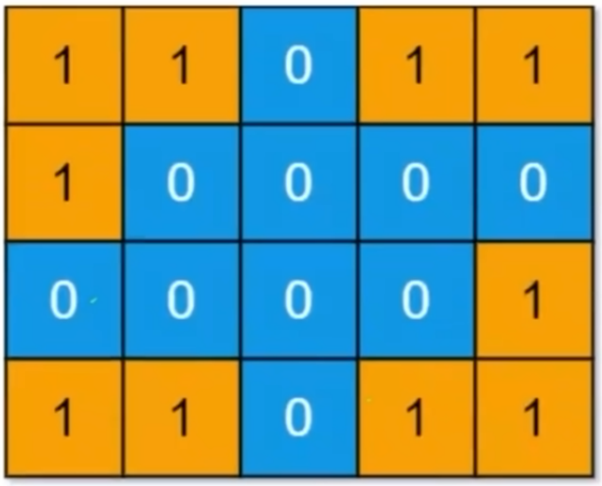
Return *the number of* ***distinct*** *islands.*

Example 1:




```
Input: grid = [[1,1,0,0,0],[1,1,0,0,0],[0,0,0,1,1],[0,0,0,1,1]]
Output: 1
```

Example 2:




```
Input: grid = [[1,1,0,1,1],[1,0,0,0,0],[0,0,0,0,1],[1,1,0,1,1]]
Output: 3
```

```
// b <============== Number of Distinct Islands ========================>
// https://www.lintcode.com/problem/860/

// ye basically same hai numbre of islands ki tarah. Bas ab yahan pe distinct
// islands batane hain. To iske liye hume har island ka path store karna padega.
// Iske liye humne hashset ka use kiya taki agar koi bhi duplicate path wala aa
// jaye to wo dubara count he na ho

// End mai humne hashset ka size return karwaya hai
```

```
public static void dfs_numberofDistinctIslands(int[][] grid, char[] dirS, int[][] dir, int sr, int sc,
        StringBuilder sb) {

    grid[sr][sc] = 2;

    for (int d = 0; d < dir.length; d++) {

        int r = sr + dir[d][0];
        int c = sc + dir[d][1];
```

```
        if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 1) {
            sb.append(dirS[d]);
            dfs_numberofDistinctIslands(grid, dirS, dir, r, c, sb);
            sb.append(str: "X");
        }
    }
}
```

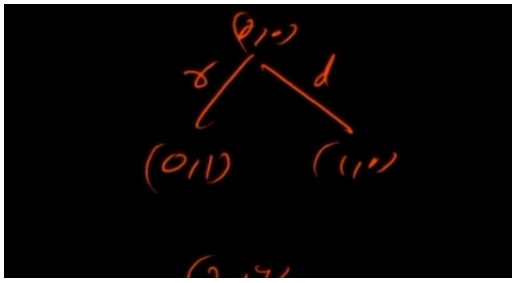
```
public int numberofDistinctIslands(int[][] grid) {

    char[] dirS = { 'U', 'R', 'D', 'L' };
    int[][] dir = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };

    int n = grid.length, m = grid[0].length;
    HashSet<String> set = new HashSet<>();

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1) {
                StringBuilder sb = new StringBuilder();
                dfs_numberofDistinctIslands(grid, dirS, dir, i, j, sb);
                set.add(sb.toString());
            }
        }
    }

    return set.size();
    // write your code here
}
```

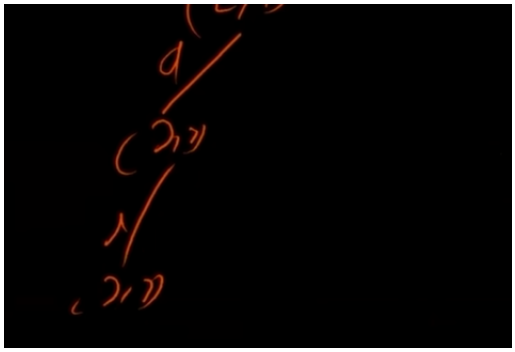
```
// ! Important point : Why added sb.append("X"); statement ?????

// consider a test case :

// 0 1 1 0 1 1
// 1 1 0 0 0 0
// 1 0 0 0 0 1
// 0 1 1 0 1 1

// For 1 placed in (0,1) ,the string will be RDLD.

// 0 1 1 0 1 1
// 0 1 1 0 0 0
// 0 1 0 0 0 1
// 0 0 1 0 1 1

// ` But the for this test case for 1 placed in (0,1), the string will again be
// RDLD. But they are two different structures. Therefore sb.append("X")
// statement was added so that we can tell that the direction was changed and
// now we have backtracked from it.

// Now for top one the string will be RXDLDXXX
// For the bottom test case the string will be RDLDXXXX

// Hence Denoting the two structures are different.
```

1 1
1

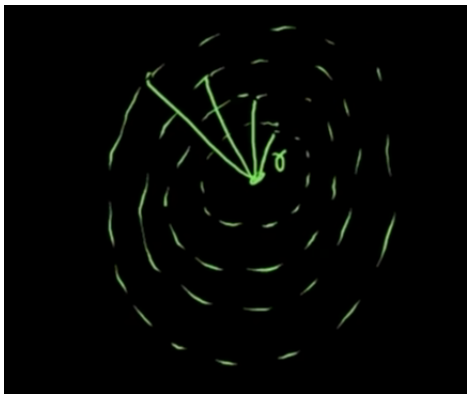
Is not same as 1 1
1

Ry run like this

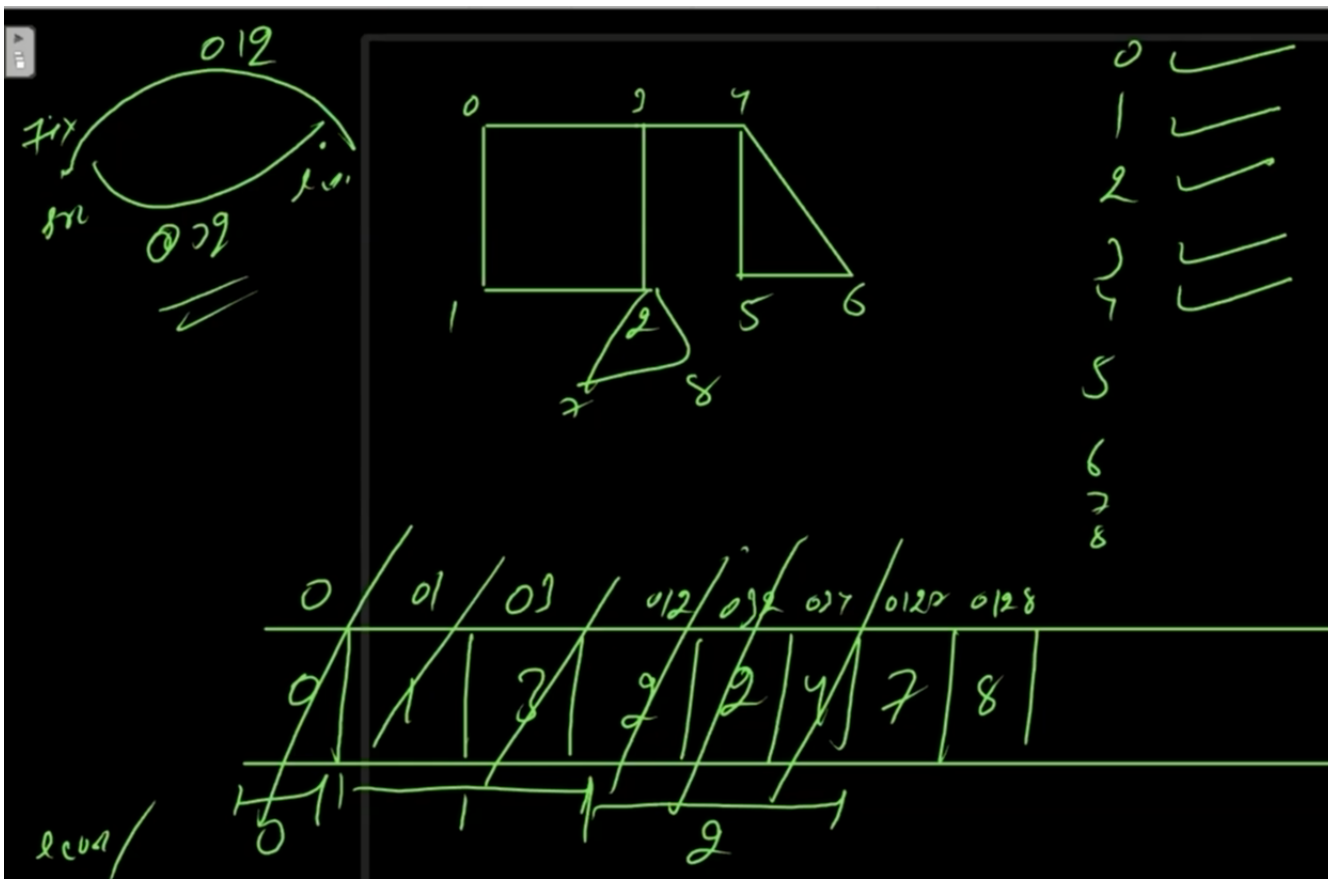




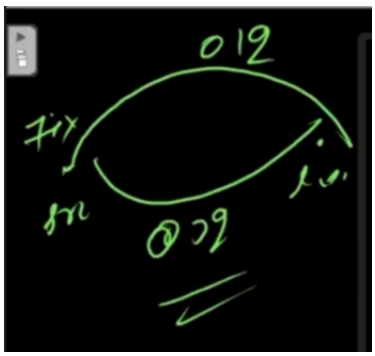
11. Cycle Bfs in Graph



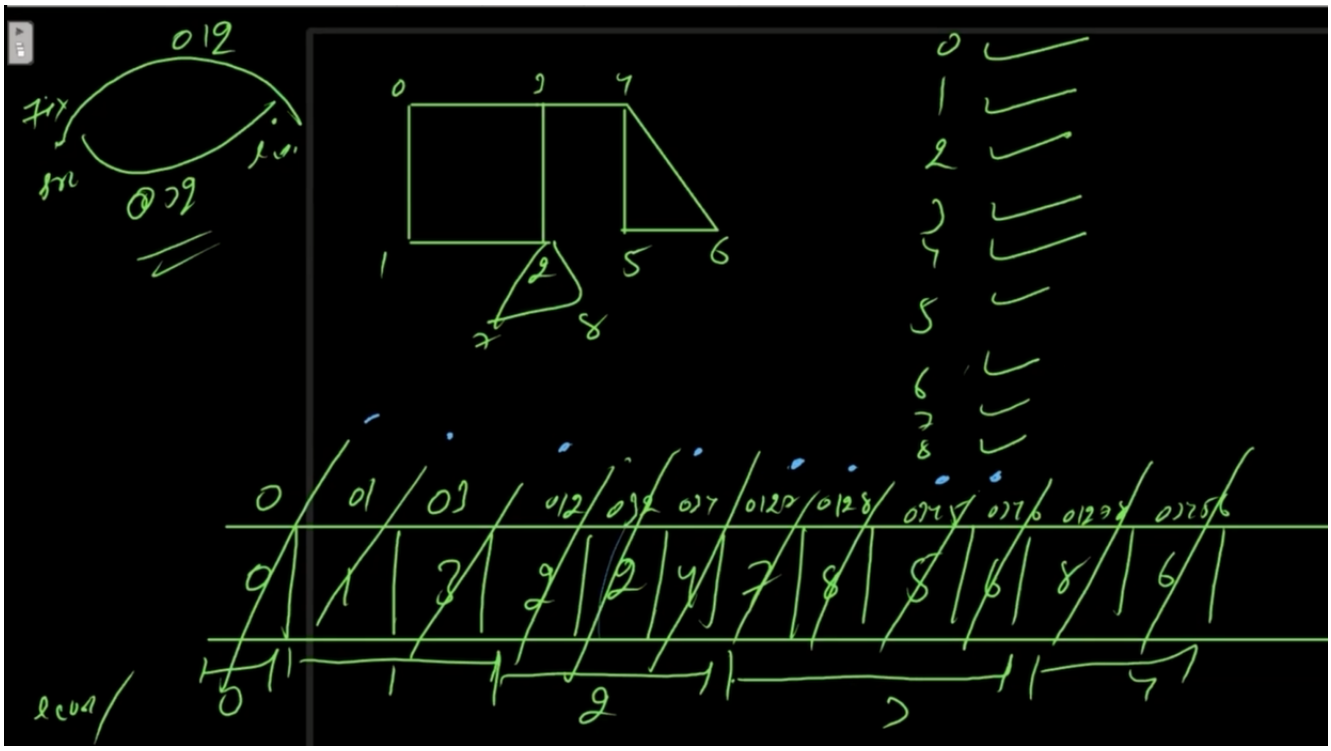
```
// ? It gives up us the shortest path from the source vtx to the destination
// ? vtx.

// Mai Bbfs mai radius wise move karta hain. Isiliye mujhe shortest path milta
// hai us vertex ka.

// Where radius is in terms of edges.

// To pehle mujhe sare wo vertices milenge jo mujhe ek edge ki duri pe hai. Tab
// wo milengi jo mujhse 2 edge ki dduri pe hai and so on ...
```



```
// # Agar koi vertex already visited hai aur wo dubara visit hoti hai, to us
// # graph mai cycle hai. Matlab us vertex tak pahunchne ke do raste hai from
// # the fixed source

// Level in Bfs indicate the minimum number of edges required to reach from that
// source vertex
```



```
// For ex : 2 tal pahunchne ke do raste hai 0 se. 012 and 032.
```



```
// ! iski complexity hogi O(E) kyunki aisa hua hai ki maine vertex ko ek se
// ! jyada bar visit kiya hai. To approximately ise O(E) bolenge
```

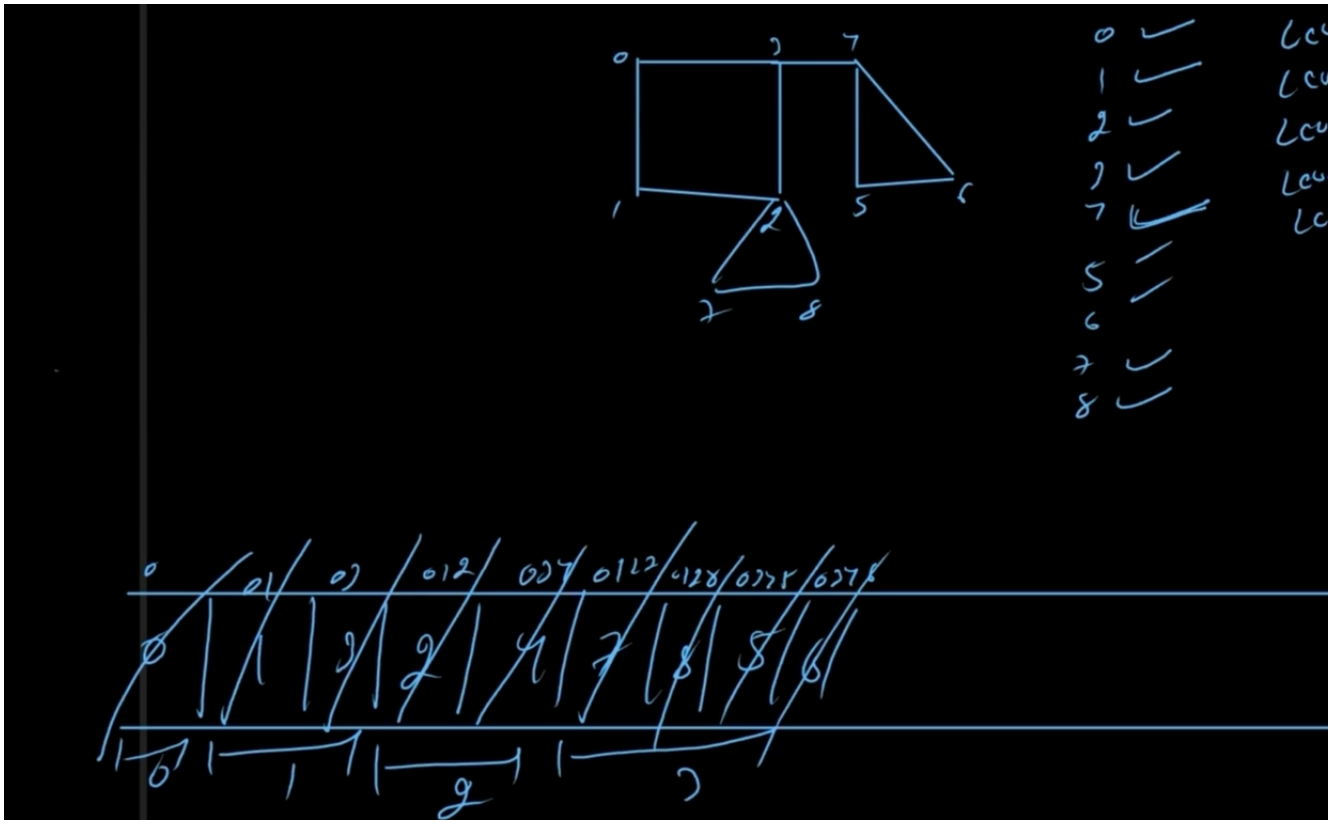
```java
// This to be used when we need to detect cycle in a question.
public static void bfs(ArrayList<Edge>[] graph, int src, boolean[] vis) {
    // Yahan pe jaise he maine kisi element ko que se nikala use maine mark kiya

    LinkedList<Integer> que = new LinkedList<>();
    que.add(src);

    int level = 0;
    boolean isCyclePresent = false;
    while (que.size() != 0) {
        int size = que.size();
        System.out.print("Level: " + level + " ->");

        while (size-- > 0) {
            int vtx = que.removeFirst();
```

```
            if (vis[vtx]) {
                System.out.println(x: "cycle");
                isCyclePresent = true;
                continue;
            }

            System.out.print(vtx + ", ");

            vis[vtx] = true;
            for (Edge e : graph[vtx]) {
                if (!vis[e.v])
                    que.addLast(e.v);
            }
        }

        level++;
        System.out.println();
    }
}
```

```
if (vis[vtx]) {
    System.out.println(x: "cycle");
    isCyclePresent = true;
    continue;
}
```

```
// To hum jab cycle aayegi to continue kar denge kyunki mai nhi chahta ki dubara
// ` us vertex ke neighbours dalein kyunki wo pehle se dal chuke hain. Agar aisa
// nhi kiya to infinite loop jaa sakta hai
```

12. Bfs without cycle

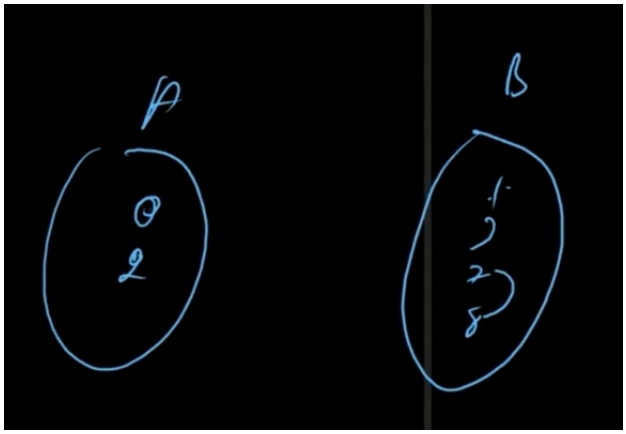


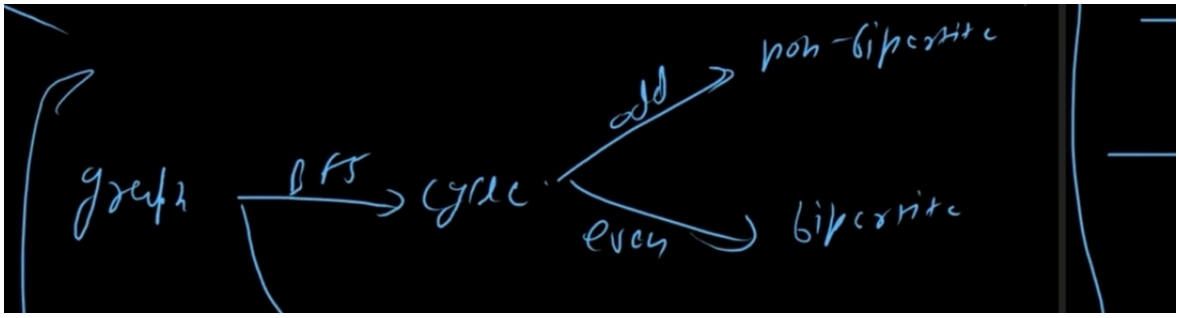
```
// b <================bfs_withoutCycle=======================>

// Yahan pe humne source ko dalte time que mai he mark kiya and jab bhi kisi
// vertex ko que mai dal rahe hain, to pehle mark kar rahe hai aur tab queue mai
// dal rahe hain

// Shortest path to a vertex ye bhi batata hai

// ! Its advantage :
```

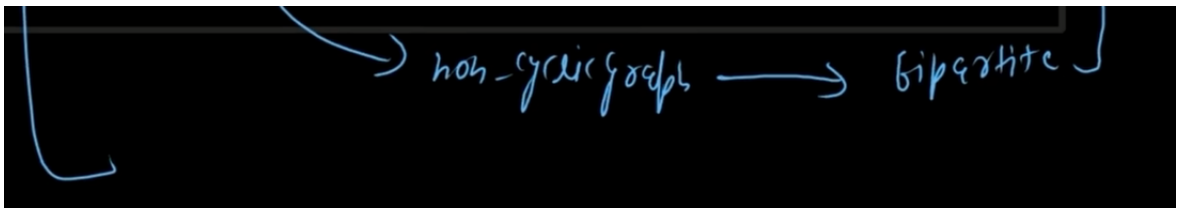
```
// ? It is slightly faster since every vertex is just visited once therofore the
// ? complexity is O(V), means it is faster than the above one.
```

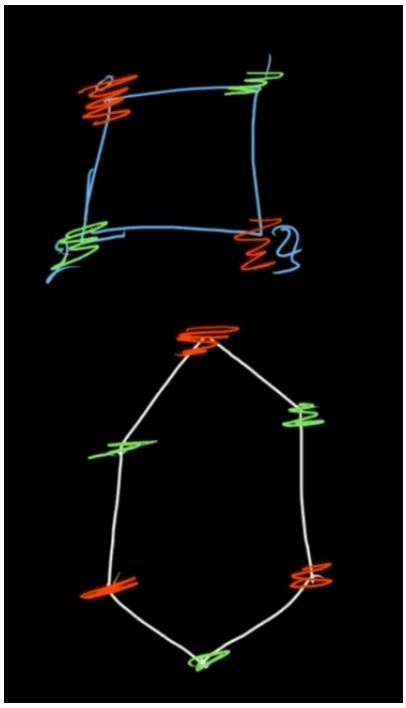
```java
public static void bfs_withouCycle(ArrayList<Edge>[] graph, int src, boolean[] vis) {
    LinkedList<Integer> que = new LinkedList<>();
    que.add(src);
    vis[src] = true;

    int level = 0;
    while (que.size() != 0) {
        int size = que.size();
        System.out.print("Level: " + level + " ->");

        while (size-- > 0) {
            int vtx = que.removeFirst();
            System.out.print(vtx + ", ");

            for (Edge e : graph[vtx]) {
                if (!vis[e.v]) {
                    vis[e.v] = true;
                    que.addLast(e.v);
                }
            }
        }

        level++;
        System.out.println();
    }
}
```

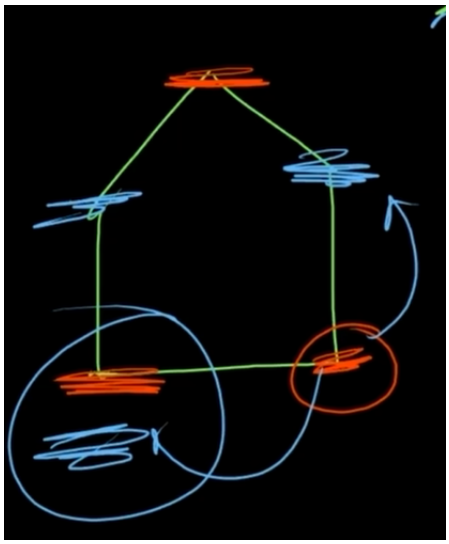
13. Bipartite Graph











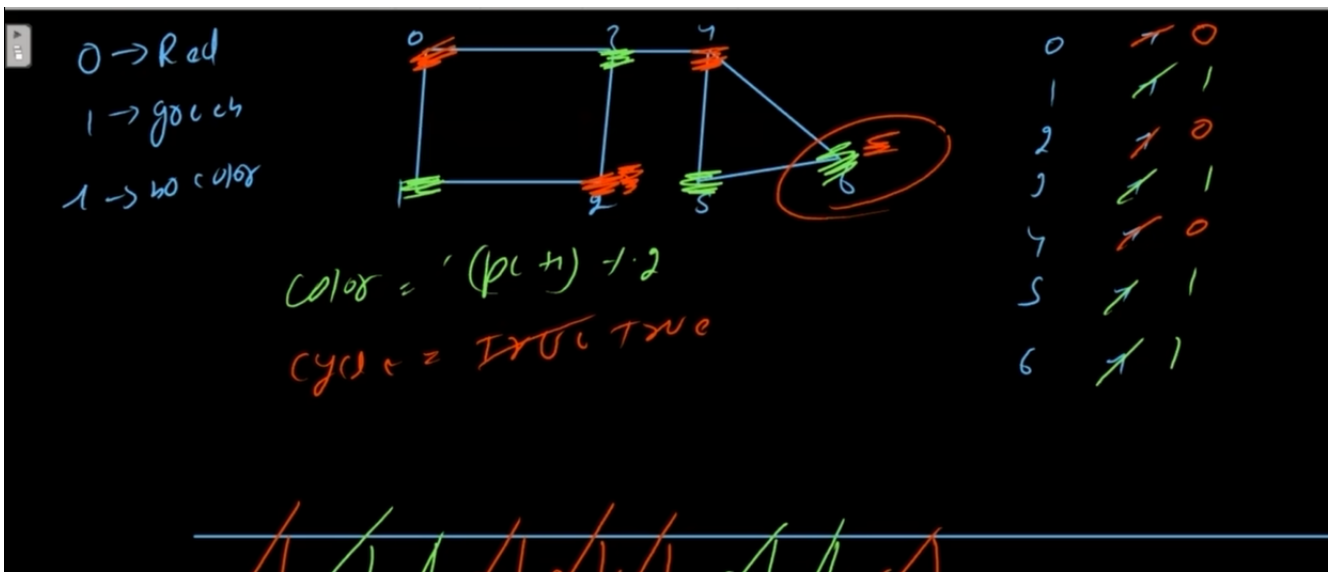
```
// b <=======================Is Graph Bipartite?==================>
// https://leetcode.com/problems/is-graph-bipartite/

// ` Basically agar tu graph ki sari vertices ko 2 aise sets mai devide kar
// sakta hai such that ki within a set, koi bhi edge form na ho vertices ,
// ke beech mai use bi-partite graph bolte hain.

// Jaise koi directed graph hai jaise ki tree, jisme cycle exist he nhi karti wo
// humesha bi-partite hoga.

// Agar even length ki cycle mili, tab bhi wo bi-partite hoga
// Per agar odd ki cycle mili, to wo bi-partite nhi hoga

// # To yahan pe kiya kya???

// To kiya ye ki hume pata hai ki agar ek vertex agar A set mai hai to uske sare
// neighbours B set mai hone chahiye bipartite hone ke liye. To set ko mark
// karne ke liye colours ka use kiya. Aur agar kahin pe bho conflict aaya (matlab
// aisa hua ki vertex ko red hona tha per wo blue hai ya vice versa), to graph
// `bipartite nhi hoga
```

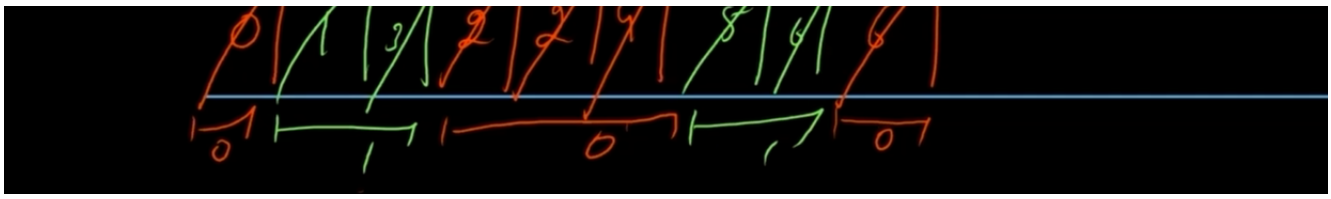
```
// Level ye indicate karta hai ki uska color kya hona chahiye. Same radius wala
// comcept.
```

```
// ? 0 : Not visited, 1 : red Marked, 2 : Blue Marked
public boolean isBipartite_(int[][] graph, int src, int[] vis) {

    LinkedList<Integer> que = new LinkedList<>();
    que.addLast(src);
    int level = 0;

    while (que.size() != 0) {

        int size = que.size();

        while (size-- > 0) {
            int vtx = que.removeFirst();

            if (vis[vtx] != 0) {
                int colorShouldBe = level % 2 == 0 ? 1
                        : 2;
                int actualColor = vis[vtx];

                if (colorShouldBe != actualColor) // conflict
                    return false;
                continue;
            }
            vis[vtx] = level % 2 == 0 ? 1 : 2;

            for (int e : graph[vtx]) {
                if (vis[e] == 0) {
                    que.addLast(e);
                }
            }
        }
        level++;
    }
    return true;
}
```

```
public boolean isBipartite(int[][] graph) {
    int n = graph.length;
    int[] vis = new int[n];

    for (int i = 0; i < n; i++) {
        if (vis[i] == 0 && !isBipartite_(graph, i, vis)) { // Agar koi ek bhi component bi-partite nhi hua to pura
                                                           // graph he bi-partitie nhi hhoga
            return false;
        }
    }
    return true;
}
```

14. Bi-partite and Even Odd Cycle





```java
//b <============== Bi-partite and odd and even cycle ==============>
public static void bipartite(ArrayList<Edge>[] graph, int src, int[] vis) {
    LinkedList<Integer> que = new LinkedList<>();
    que.addLast(src);

    // No Color : -1 , Red : 0, Green : 1
    int color = 0;
    boolean isCycle = false, isBipartite = true;

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int rvtx = que.removeFirst();
            if (vis[rvtx] != -1) {
                isCycle = true;
                if (color != vis[rvtx]) // conflict
                    isBipartite = false;
                continue;
            }
            vis[rvtx] = color;
            for (Edge e : graph[rvtx]) {
                if (vis[e.v] == -1) {
                    que.addLast(e.v);
                }
            }
        }
        color = (color + 1) % 2;
    }
    if (!isCycle)
        System.out.println(x: "Bipartite Graph with no cycle");
    else {
        if (isBipartite)
            System.out.println(x: "Bipartite Graph with Even Length cycle");
        else
            System.out.println(x: "Non Bipartite Graph with Odd Length cycle");
    }
}
```

15. Rotten Oranges (Dfs)

994. Rotting Oranges

Medium 8195 305 Add to List Share

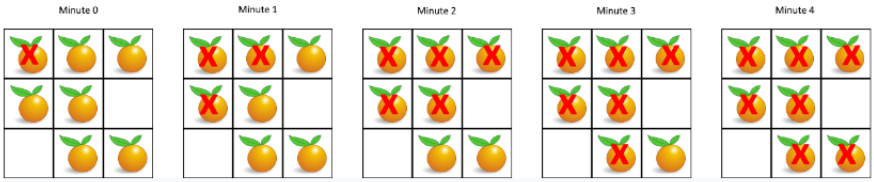
You are given an `m x n` `grid` where each cell can have one of three values:

- `0` representing an empty cell,
- `1` representing a fresh orange, or
- `2` representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return *the minimum number of minutes that must elapse until no cell has a fresh orange.* If *this is impossible, return* `-1`.

Example 1:



Input: grid = [[2,1,1],[1,1,0],[0,1,1]]

Output: 4

Example 2:

Input: grid = [[2,1,1],[0,1,1],[1,0,1]]
Output: -1
Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Example 3:

Input: grid = [[0,2]]
Output: 0
Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

```
{ {2, 1, 0, 2, 1},
  {1, 0, 1, 2, 1},
  {1, 0, 0, 2, 1}};
```

```
// b<===============Rotten Oranges ========================>
// https://leetcode.com/problems/rotting-oranges/

// So basically kiya ye ki jitne bhi rotten oranges hai unse call lagayi aur
// time + 1 karke grid mai h save karate rahe time ko. Isiliye time ko 2 se
// start kiya hai.

// # Important Point :-

// Ab aisa bhi ho sakta hai ki rotten oranges bahut sare ho aur pehle wale ne he
// sare fresh ko visit karke apna time save kara diya ho grid mai. Iske liye
// (grid[r][c] == 1 || grid[r][c] > time + 1) ye check add kiya hai ki agar
// fresh ho to call lagao ya fir agar current time ki value jahan jaa rahe hain
// grid mai usse kam hai to call lagao taki grid mai uska time update ho jaye

// ! For test case : [[2,1,1],[1,1,1],[0,1,2]], Dry run and the see the
// ! importance of the check.
```

```
public static void dfs_orangesRotting(int[][] grid, int sr, int sc, int[][] dir, int time) {

    grid[sr][sc] = time;

    for (int d = 0; d < dir.length; d++) {
        int r = sr + dir[d][0];
        int c = sc + dir[d][1];

        if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length
                && (grid[r][c] == 1 || grid[r][c] > time + 1)) {
            dfs_orangesRotting(grid, r, c, dir, time + 1);
        }
    }
}
```

```
public int orangesRotting(int[][] grid) {

    int n = grid.length, m = grid[0].length;
    int[][] dir = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };
    int minTimeTakenToRot = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 2) {
                dfs_orangesRotting(grid, i, j, dir, time: 2);
            }
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1) {
                return -1;
            }
            minTimeTakenToRot = Math.max(grid[i][j], minTimeTakenToRot);
```

```
        }
    }

    // Agar uper sab ke baad bhi minTimeTakenToRot 0 aaya matlab, koi bhi fresh
    // oranges tha he nhi.

    return minTimeTakenToRot == 0 ? 0 : minTimeTakenToRot - 2; // -2 since we started the time from 2 and considered
                                                               // it as 0th second

}
```

16. Rotten Oranges (Bfs ===> Used Parellel Bfs that has multiple source)

```
// # Here using bfs without cycle
```

```
// b <================= Rotten Oranges ================>
// https://leetcode.com/problems/rotting-oranges/

// ? Here we are going to use parallel bfs

// ? Means we are going to have multiple starting source and for each of them we
// ? are going to run bfs.

// To basically kiya ye ki jitne bhi initially rotten oranges hain unhe que mai
// store kara. Tab sabki sath mai bfs run kiya.

// Level ye denote karega ki kitna time lga hai rot hone mai.

// 0th level mai sare initial rotten aayenge.
// 1st level mai wo aayenge jo time 1 mai rot hue.

// Aisa isiliye kyunki bfs mai yhi to hota hai ki hum apne sare neighbours ko
// agle level mai dal dete hain. Hence hume min time mil jayega

// To ye hume minimum time nikalke dega jab humare sare oranges rot ho jayenge

// r=idx / m, c=idx % m, idx=r * m + c;
```

```
public int orangesRotting(int[][] grid) {

    LinkedList<Integer> que = new LinkedList<>();
    int n = grid.length, m = grid[0].length;
    int oneCount = 0;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 2) {
                que.addLast(i * m + j);
            } else if (grid[i][j] == 1) {
                oneCount++; // To calculate the number of fresh oranges
            }
        }
    }

    if (oneCount == 0)
        return 0;

    int[][] dir = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
    int time = 0; // denoting level;
    while (que.size() != 0) {
        int size = que.size();

        while (size-- > 0) {
            int rvtx = que.removeFirst();
            int sr = rvtx / m;
            int sc = rvtx % m;
            for (int d = 0; d < dir.length; d++) {

                int r = sr + dir[d][0];
                int c = sc + dir[d][1];

                if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 1) {
                    grid[r][c] = 2;
                    que.addLast(r * m + c);
                    oneCount--;
                    if (oneCount == 0) { // if by any chance the number reduces to 0, it means all are rotten.
                        return time + 1;
                    }
                }

            }
        }

        time++;
    }

    // Agar uper wale while loop se koi bahar aata hai, iksa matlab ye hoga ki abhi
    // ` bhi oneCount 0 nhi hai matlab fresh oranges hain
    return -1;
}
```

17. 1091. Shortest Path in Binary Matrix

1091. Shortest Path in Binary Matrix

Medium 3904 165 Add to List Share

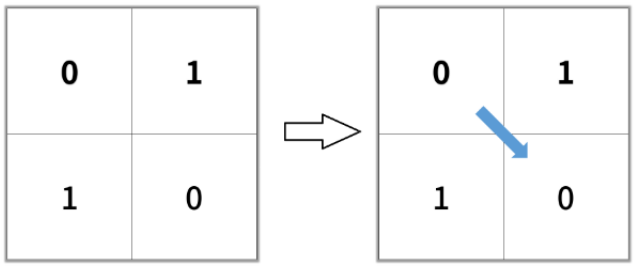
Given an `n x n` binary matrix `grid`, return *the length of the shortest **clear path** in the matrix*. If there is no clear path, return `-1`.

A **clear path** in a binary matrix is a path from the **top-left** cell (i.e., `(0, 0)`) to the **bottom-right** cell (i.e., `(n - 1, n - 1)`) such that:

- All the visited cells of the path are `0`.
- All the adjacent cells of the path are **8-directionally** connected (i.e., they are different and they share an edge or a corner).

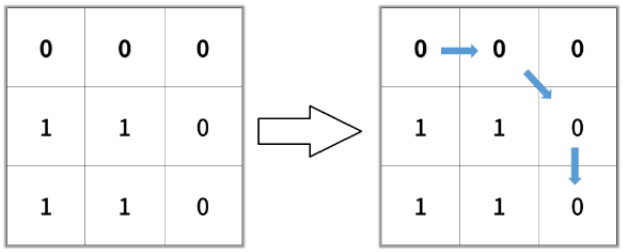
The **length of a clear path** is the number of visited cells of this path.

Example 1:




```
Input: grid = [[0,1],[1,0]]
Output: 2
```

Example 2:




```
Input: grid = [[0,0,0],[1,1,0],[1,1,0]]
Output: 4
```

Example 3:

```
Input: grid = [[1,0,0],[1,1,0],[1,1,0]]
Output: -1
```

Constraints:

- `n == grid.length`
- `n == grid[i].length`
- `1 <= n <= 100`
- `grid[i][j] is 0 or 1`

```
// b<============ 1091. Shortest Path in Binary Matrix ==========>
// https://leetcode.com/problems/shortest-path-in-binary-matrix/
```

```
// yahan pe basically ye logic lagaya ki bfs hume shortest path nikal ke deta
// hai kisi vertex ka.

// So again Humne 2-D ko 1-D mai convert karke index dale hain que ke andar
```

```java
public int shortestPathBinaryMatrix(int[][] grid) {
    int n = grid.length, m = grid[0].length;

    if (grid[0][0] == 1 || grid[n - 1][m - 1] == 1) {
        return -1;
    }

    int[][] dir = { { -1, 0 }, { 1, 0 }, { -1, 1 }, { 1, -1 }, { 0, 1 }, { 0, -1 }, { 1, 1 }, { -1, -1 } };

    LinkedList<Integer> que = new LinkedList<>();
    que.addLast(e: 0);

    int shortestPath = 1;
    while (que.size() != 0) {
        int size = que.size();

        while (size-- > 0) {

            int rvtx = que.removeFirst();

            int sr = rvtx / m, sc = rvtx % m;
            if (sr == n - 1 && sc == m - 1) { // agar bottom right cell mila to return kara level ka size which here
                                              // indicate the shortest path
                return shortestPath;
            }

            for (int d = 0; d < dir.length; d++) {
                int r = sr + dir[d][0];
                int c = sc + dir[d][1];

                if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 0) {
                    grid[r][c] = 1;
                    que.addLast(r * m + c);
                }
            }
        }
        shortestPath++;
    }

    // Agar uper wale loop ke bahar aaya iska matlab use answer mila he nhi, to
    // return -1 kiya.
    return -1;
}
```

18. 01 Matrix

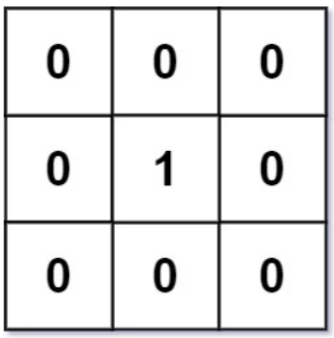
542. 01 Matrix

Medium 5560 277 Add to List Share

Given an `m x n` binary matrix `mat`, return *the distance of the nearest* `0` *for each cell.*

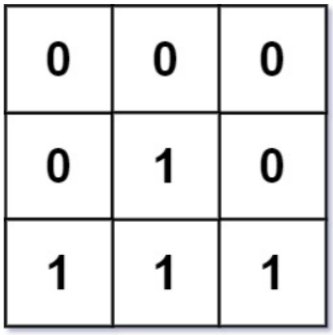
The distance between two adjacent cells is `1`.

Example 1:




```
Input: mat = [[0,0,0],[0,1,0],[0,0,0]]
Output: [[0,0,0],[0,1,0],[0,0,0]]
```

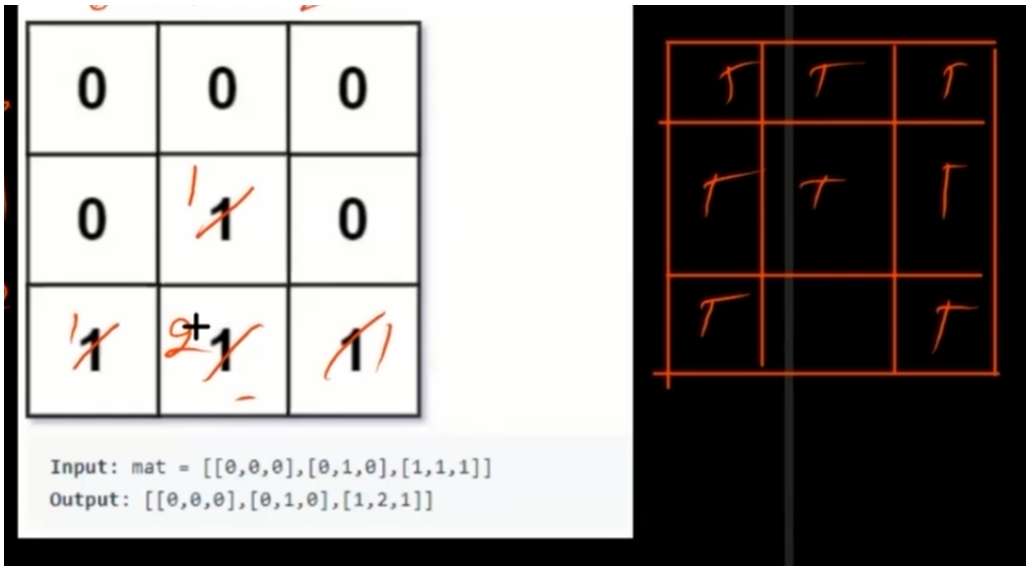
Example 2:

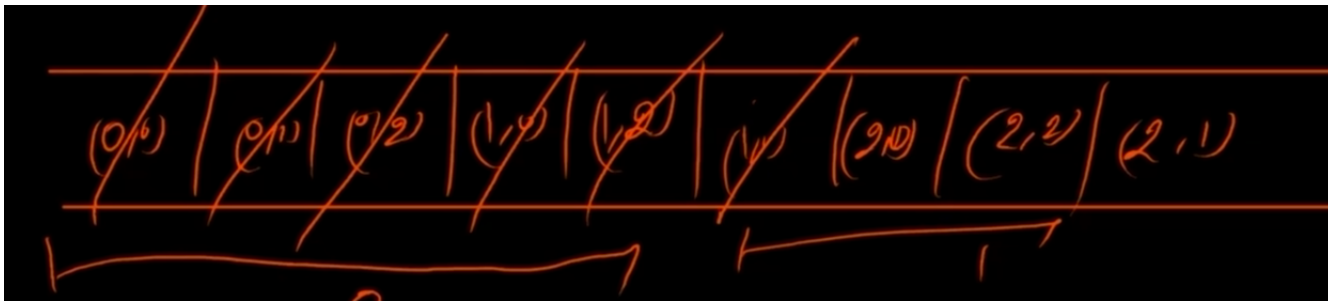



```
Input: mat = [[0,0,0],[0,1,0],[1,1,1]]
Output: [[0,0,0],[0,1,0],[1,2,1]]
```

Constraints:

- `m == mat.length`
- `n == mat[i].length`
- `1 <= m, n <= 10^4`
- `1 <= m * n <= 10^4`
- `mat[i][j]` is either `0` or `1`.
- There is at least one `0` in `mat`.





```cpp
// b <============= 542. 01 Matrix ================>
// https://leetcode.com/problems/01-matrix/

// ? Again applied the concept of parellel bfs

// kiya kya ki sare 0 ko pehle he dal diya que mia aur ek vis ka array banake
// ` unko mark kiya. Ab parallel bfs run kiya sare 0 ke liye. To jitne bhi 1 0
// se one distance pe the wo level 1 mai aayenge. Similarly jo 2 ki distance mai
// the wo level 2 mai aayenge. and so goes on.

// # So again Humne 2-D ko 1-D mai convert karke index dale hain que ke andar
// # r=idx/m, c=idx%m, idx=r*m + c, where m is the number of columns

// ! Baki dry run and figure out.

//
```

```
// ------------------------------------------------------------------------

// Ek aur tareeka tha per wo bahut he high complexity ka jaa raha tha.

// Humse jitne bhi one hain, unke liye individually bfs call karte aur nearest 0
// nikalte. So basically ye humara n² approach hota.
```

```java
public int[][] updateMatrix(int[][] grid) {

    int n = grid.length, m = grid[0].length;
    if (n == 0 || m == 0) {
        return grid;
    }

    int[][] dir = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
    LinkedList<Integer> que = new LinkedList<>();
    boolean[][] vis = new boolean[n][m];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 0) {
                que.addLast(i * m + j);
                vis[i][j] = true;
            }
        }
    }

    int distance = 0;
    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int rvtx = que.removeFirst();
            int sr = rvtx / m, sc = rvtx % m;

            grid[sr][sc] = distance; // ` updated the grid with the distance(level of bfs)

            for (int d = 0; d < dir.length; d++) {
                int r = sr + dir[d][0];
                int c = sc + dir[d][1];

                if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && !vis[r][c]) {
                    vis[r][c] = true; // Marked so that the vertex that is visited is not added again in the queue.
                    que.addLast(r * m + c);
                }
            }
        }
        distance++;
    }
    return grid;
}
```

19. Number of Enclaves

1020. Number of Enclaves

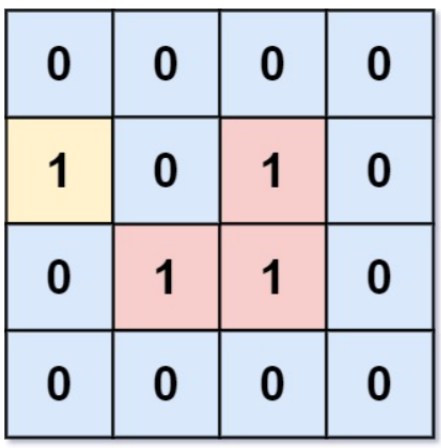
Medium 1642 36 Add to List Share

You are given an `m x n` binary matrix `grid`, where `0` represents a sea cell and `1` represents a land cell.

A **move** consists of walking from one land cell to another adjacent (**4-directionally**) land cell or walking off the boundary of the `grid`.

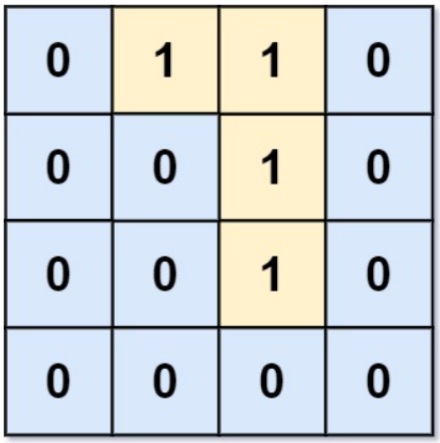
Return *the number of land cells in* `grid` *for which we cannot walk off the boundary of the grid in any number of* ***moves***.

Example 1:



Input: grid = [[0,0,0,0],[1,0,1,0],[0,1,1,0],[0,0,0,0]]
Output: 3
Explanation: There are three 1s that are enclosed by 0s, and one 1 that is not enclosed because its on the boundary.

Example 2:



Input: grid = [[0,1,1,0],[0,0,1,0],[0,0,1,0],[0,0,0,0]]
Output: 0
Explanation: All 1s are either on the boundary or can reach the boundary.

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 500`
- `grid[i][j]` is either `0` or `1`.

```
// b <=================== 1020. Number of Enclaves ====================>
// https://leetcode.com/problems/number-of-enclaves/

// ? Ye basically surrounded regions ki tarah hai.

// Isme hume wo land cells ka count return karna hai jinse hum grid se bahar nhi
// jaa sakate.

// So basically ye wo land cells honge jo agar boundary land cell se connected
// nhi honge.

// To humne kiya ye jo bhi land cells boundary wale land se connected thi, unko
// mark kar diya.

// # Ab jo land cells bachengi wo boundary wle land se connected nhi hongi, to
// # unhe count karke return kar diya.
```

```
public static void dfs_numEnclaves(int[][] grid, int sr, int sc, int[][] dir) {
    grid[sr][sc] = 2;

    for (int d = 0; d < dir.length; d++) {
        int r = sr + dir[d][0];
        int c = sc + dir[d][1];

        if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 1) {
            dfs_numEnclaves(grid, r, c, dir);
        }
    }
}
```

```
public int numEnclaves(int[][] grid) {

    int n = grid.length, m = grid[0].length;
    int[][] dir = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
```

```
            if ((i == 0 || j == 0 || i == n - 1 || j == m - 1) && grid[i][j] == 1) { // Called dfs for all land
                                                                                      // cells in the boundary.
                dfs_numEnclaves(grid, i, j, dir);
            }
        }
    }

    int landCount = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1) {
                landCount++;
            }
        }
    }

    return landCount;
}
```

20. Walls and Gates

286. Walls and Gates

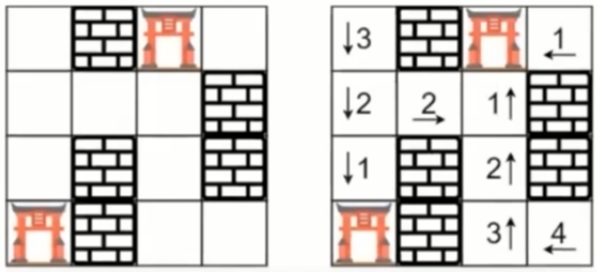
Medium 1761 27 Add to List Share

You are given an `m x n` grid `rooms` initialized with these three possible values.

- `-1` A wall or an obstacle.
- `0` A gate.
- `INF` Infinity means an empty room. We use the value $2^{31} - 1 = 2147483647$ to represent `INF` as you may assume that the distance to a gate is less than `2147483647`.

Fill each empty room with the distance to *its nearest gate*. If it is impossible to reach a gate, it should be filled with `INF`.

Example 1:




```
Input: rooms = [[2147483647,-1,0,2147483647],
[2147483647,2147483647,2147483647,-1],
[2147483647,-1,2147483647,-1],[0,-1,2147483647,2147483647]]
Output: [[3,-1,0,1],[2,2,1,-1],[1,-1,2,-1],[0,-1,3,4]]
```

```
// b <<================= 663 · Walls and Gates ==================>
// https://www.lintcode.com/problem/663/

// ? Same as rotten oranges, parallel bfs run kiya hai. Aur bfs_wihout cycle use
// ? kiya hai.

// Sabse pehle sare gates ko que mai dala uske baad bfs run karke find kiya ki
// empty cells ki distance kya hai gates se.

// ! Ek bar dry run karego to sub clear.
```

```
public void wallsAndGates(int[][] grid) {
    // write your code here

    int n = grid.length, m = grid[0].length;
    if (n == 0 || m == 0)
        return;

    int[][] dir = { { -1, 0 }, { 1, 0 }, { 0, 1 }, { 0, -1 } };
    LinkedList<Integer> que = new LinkedList<>();

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 0) {
```

```
                que.addLast(i * m + j);
            }
        }
    }

    int shortestPath = 0; // Level : denotes radius. Ki mujse kitni distance pe hai.
    while (que.size() != 0) {
        int size = que.size();

        while (size-- > 0) {
            int rvtx = que.removeFirst();
            int sr = rvtx / m, sc = rvtx % m;
            for (int d = 0; d < dir.length; d++) {

                int r = sr + dir[d][0];
                int c = sc + dir[d][1];

                if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 2147483647) {
                    grid[r][c] = shortestPath + 1; // Can also be written as grid[sr][sc] + 1. Matlab main jitni
                                                   // distance pe hun usse + 1 mai hai mera neighbour isiliye
                    que.addLast(r * m + c);
                }
            }
        }
        shortestPath++;
    }
}
```

21. Journey to the Moon

The member states of the UN are planning to send 2 people to the moon. They want them to be from different countries. You will be given a list of pairs of astronaut ID's. Each pair is made of astronauts from the same country. Determine how many pairs of astronauts from different countries they can choose from.

Example

$n = 4$

$astronaut = [1, 2], [2, 3]$

There are 4 astronauts numbered 0 through 3. Astronauts grouped by country are $[0]$ and $[1, 2, 3]$. There are 3 pairs to choose from: $[0, 1]$, $[0, 2]$ and $[0, 3]$.

Function Description

Complete the journeyToMoon function in the editor below.

journeyToMoon has the following parameter(s):

- int n: the number of astronauts
- int astronaut[p][2]: each element $astronaut[i]$ is a 2 element array that represents the ID's of two astronauts from the same country

Returns

- int: the number of valid pairs

Input Format

The first line contains two integers n and p, the number of astronauts and the number of pairs.
Each of the next p lines contains 2 space-separated integers denoting astronaut ID's of two who share the same nationality.

Constraints

- $1 \leq n \leq 10^5$
- $1 \leq p \leq 10^4$

Sample Input 0

```
5 3
0 1
2 3
0 4
```

Sample Output 0

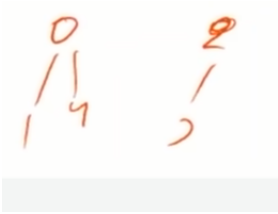
```
6
```

```
6
```

Explanation 0

Persons numbered $[0, 1, 4]$ belong to one country, and those numbered $[2, 3]$ belong to another. The UN has 6 ways of choosing a pair:

$[0, 2], [0, 3], [1, 2], [1, 3], [4, 2], [4, 3]$



Sample Input 1

```
4 1
0 2
```

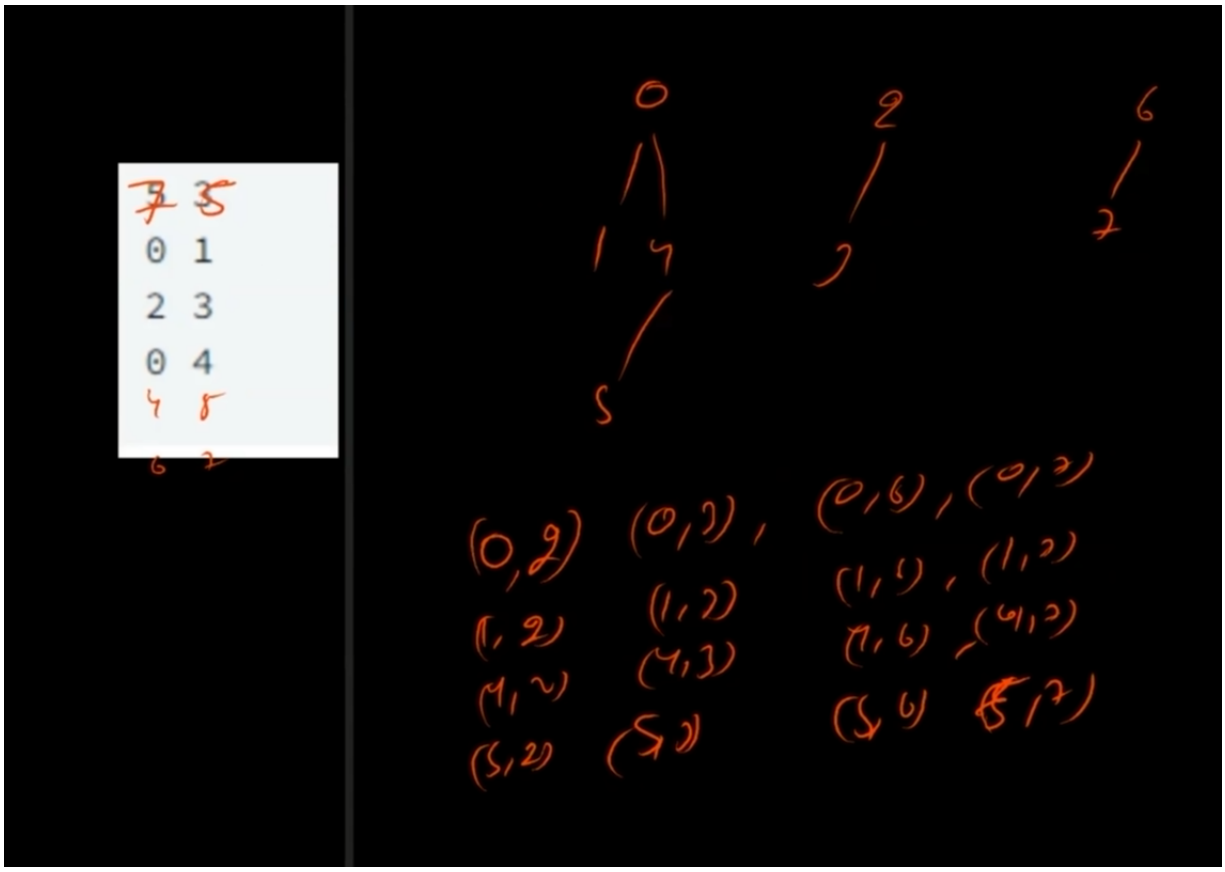
Sample Output 1

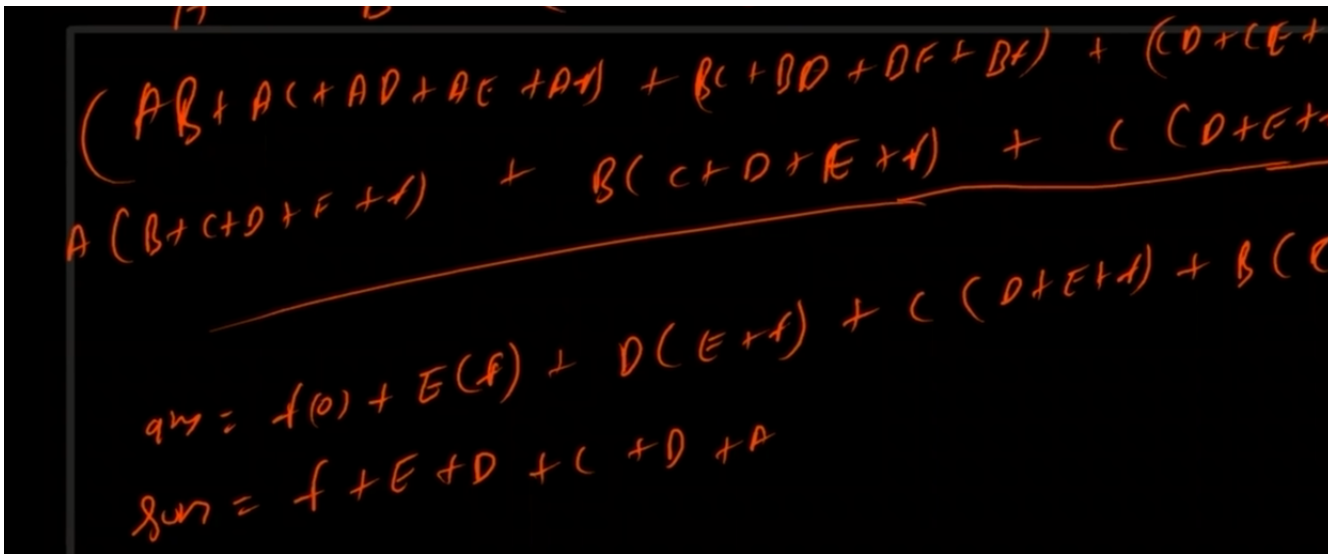
```
5
```

Explanation 1

Persons numbered $[0, 2]$ belong to the same country, but persons 1 and 3 don't share countries with anyone else. The UN has 5 ways of choosing a pair:

$[0, 1], [0, 3], [1, 2], [1, 3], [2, 3]$





```
// b <========== Journey to the Moon ==================>
// https://www.hackerrank.com/challenges/journey-to-the-moon/problem

// ? Hai kya question mai ??
// Hume basically manga ye hai ki number of pairs batao jo ban sakte hain. Aur
// condition ye hai ki pair mai dono different country ke hone chahiye

// aur humko jo do size ka array diya hai wo ye denote karta hai ki dono
// element(candidate) same country ko belong karte hain

// Ab hum agar isko ek edge ki tarah mante hain to hum apna graph create kar
// sakte hain aur different conponent of graph different country ko represent
// karte hain

// To hume itna pata lag gaya hai to bas ab combination nikalne baki hain.

// To for ex : Agar A, B , C, D ,E ,F different country hain aur inka size kuch
// a, b,c,d,e,f hai

// To jo total different pair banenge wo honge :
// a(b+c+d+e+f) + b(c+d+e+f) + c(d+e+f) + d(e+f) + e(f) + f(0)

// since the last element f will not have anything to pair with so multiplied it
// with 0 to get our equation
```

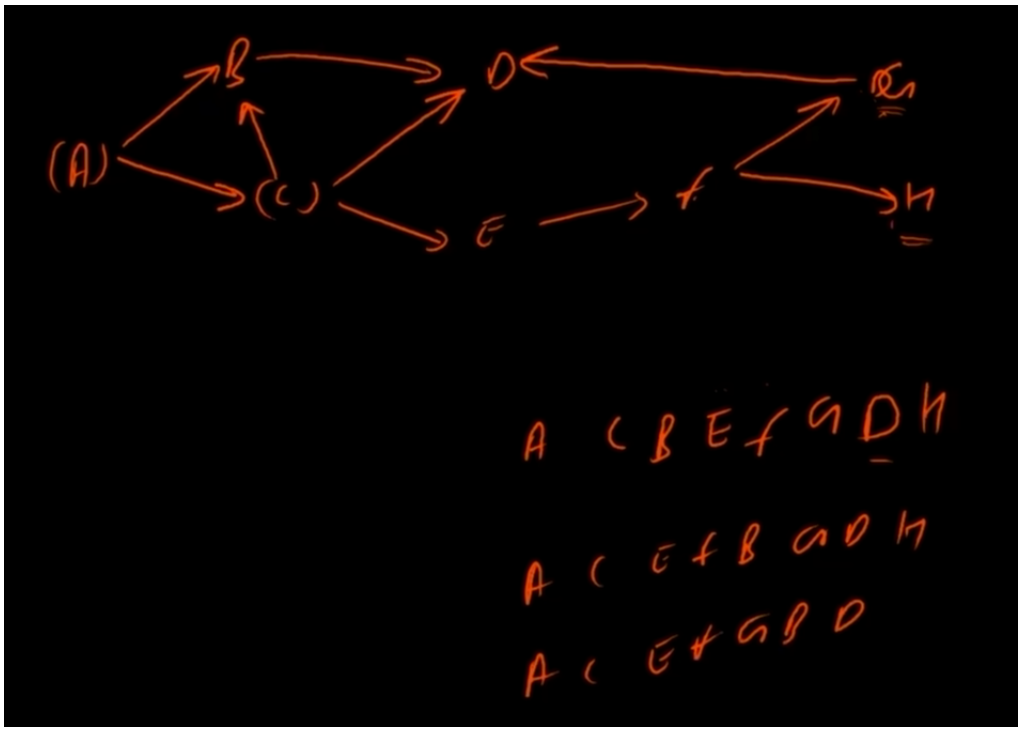
```java
public static int dfs_journeyToMoon(ArrayList<Integer>[] graph, int src, boolean[] vis) {

    vis[src] = true;
    int size = 0;

    for (int v : graph[src]) {
        if (!vis[v]) {
            size += dfs_journeyToMoon(graph, v, vis);
        }
    }

    return size + 1;
}
```

```java
public static long journeyToMoon(int n, List<List<Integer>> astronaut) {
    // Write your code here

    ArrayList<Integer>[] graph = new ArrayList[n];
    for (int i = 0; i < n; i++) {
        graph[i] = new ArrayList<>();
    }

    for (List<Integer> e : astronaut) {
        int vtx = e.get(index: 0);
        int nbr = e.get(index: 1);

        graph[vtx].add(nbr);
        graph[nbr].add(vtx);
    }
```

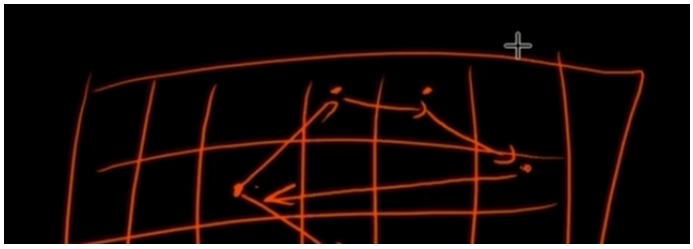
```
        boolean[] vis = new boolean[n];
        long sum = 0, ans = 0;
        for (int i = 0; i < n; i++) {
            if (!vis[i]) {
                int size = dfs_journeyToMoon(graph, i, vis); // Return the size of the component.
                ans += size * sum;
                sum = sum + size;
            }
        }

        return ans;

    }
```

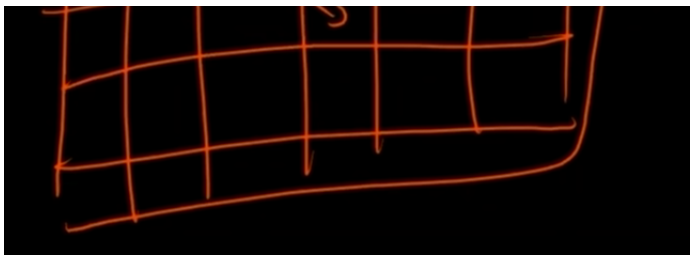
22. Topological Order

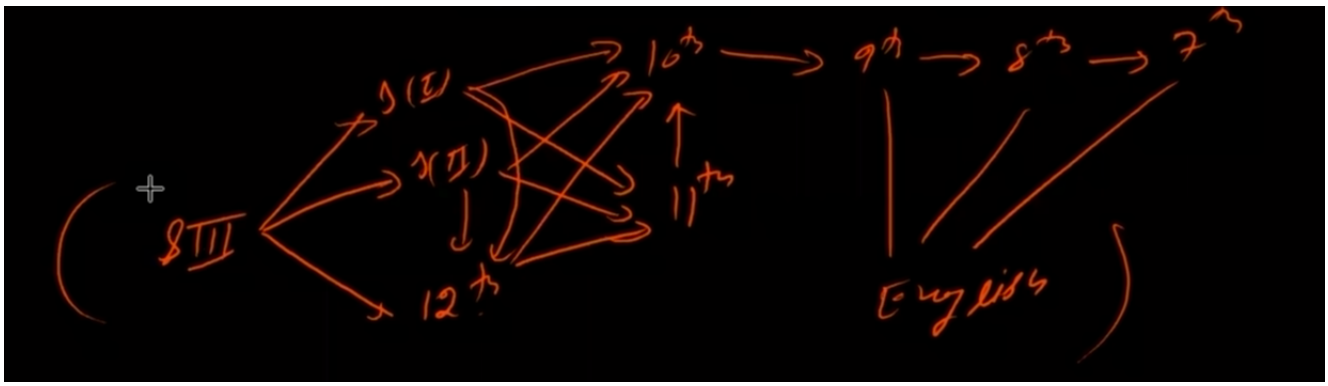
```
// b <================ Topological Order ==================>

// ? What is topological order ?
// Topological order basically wahan pe use hota hai janan pe dependencies hoti
// hain.

// Dependencies ka matlab ye hai ki mere kaam pure hone ke liye mujhe kisi aur
// pe dependent hun.
```




```
// ? For ex :

// 1. A Program execution during compilation. Aisa bahut bar hota hai ki kisi
// function ko pehle pure execute hone ke liye wo dusre function pe dependent
// hota hai

// 2. In Excel, the cells are sometimes dependent on the another cells for the
// answer and if there a cycle exists then the the excel shows the error.

// 3. To study the book of higher class, we need to have studied the lower class
// `books. So we are dependent on the lower class books in order to understand
// the higher class books
```







```
// ! So importantly the game of dependency is everywhere.

// ! The definition :

// # To ye aisa order nikal ke deta hai ki agar aap usse execution start karo to
// # aapko answer mil jayega

// So pehle meri sari dependencies resolve ho tabhi mai apna kaam kar paunga.

// ? Now this type of graph becomes directed. But why ??

// Consider ki do variables hain A and B. Aur A dependent hai B ke uper ki jab B
// ki value aajeyegi tabhi hum A ko calculate kar payenge. Per agar B A pe
// dependent nikla to ab ye possilbe he nhi hai calculate karna. To graph ke
// terms mai cycle exist kar jayegi.

// # Therefore a directed graph.
```

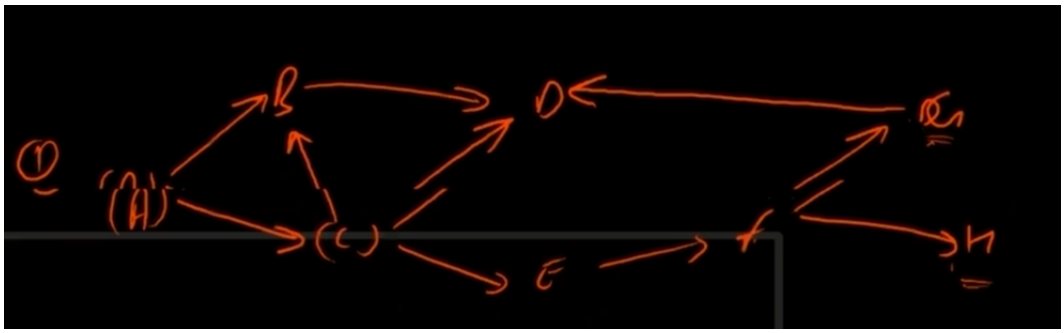
```
// ! Now to get a topological order, we add the vertices in post order. Why?

// Kyunki pehle mai apne neighbours ki dependencies ko reolve karunga tab jake
// to mai apni dependencies ko resolve kar paunga na

// Just like a tree, pehle mere sare children resolve hue tab maine apne aap ko
// revolve kiya
```

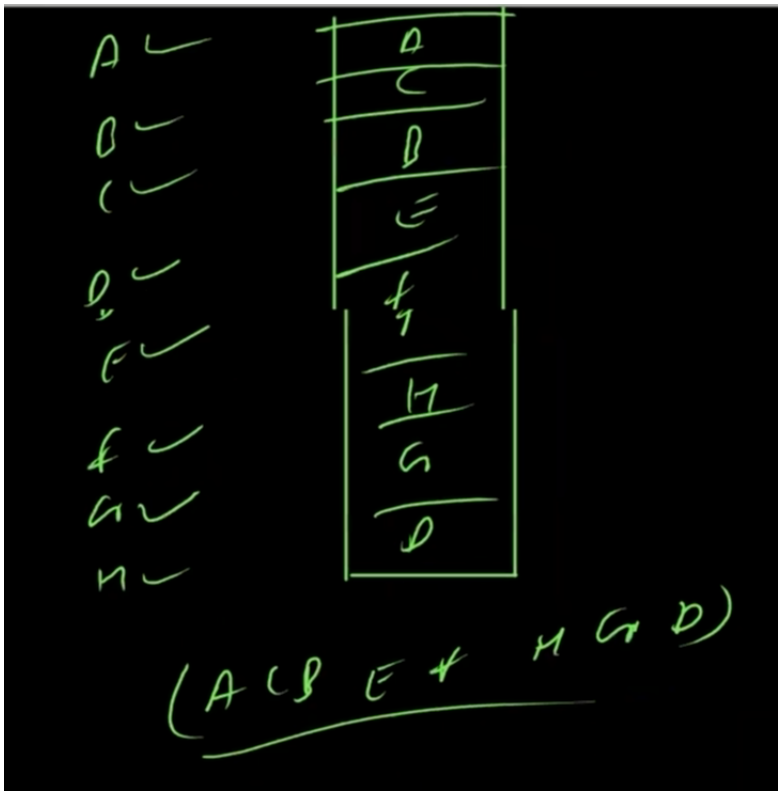
```
public static void topo_DFS(int src, ArrayList<Edge>[] graph, boolean[] vis, ArrayList<Integer> ans) {
    vis[src] = true;
    for (Edge e : graph[src]) {
        if (!vis[e.v]) {
            topo_DFS(e.v, graph, vis, ans);
        }
    }
    ans.add(src); // Added in post order

    // # This normal dfs call does not detect cycle and hence gives a wrong
    // # topological order whenever there is a cycle present in the graph.
}
                                                    l002_directed.Edge
public static void topologicalOrder(int n, ArrayList<Edge>[] graph) {
    boolean[] vis = new boolean[n];
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 0; i < n; i++) { // Hum yahan pe vertices ko randomly call bhi laga sakte hai and tabhi bhi
                                  // answer same he aayega. Aisa isiliye kyunki hum postorder mai kaam kar rahe
                                  // hain aur usme pehle sare children/neightbours visit honge tab hum
        if (!vis[i]) {
            topo_DFS(i, graph, vis, ans);
        }
    }

    for (int i = ans.size() - 1; i >= 0; i--) {
        System.out.println(ans.get(i));
```

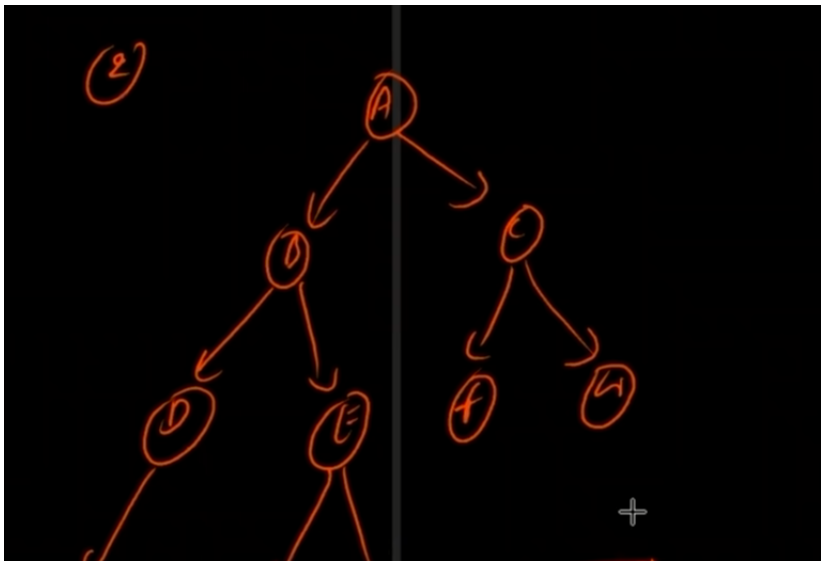
```
            System.out.println(ans.get(i));
        }
    }
```

```
// ! How to check if the topological order found is right or wrong :

// Humne egdes bana ke dekha us order mai. aur agar sare edges same direction
// mai aaye to order shi hai, nhi to order galat hai
```

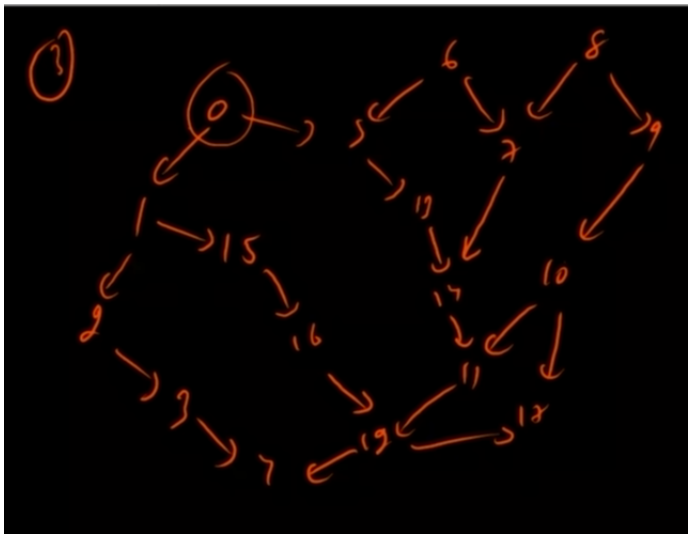
Ex :

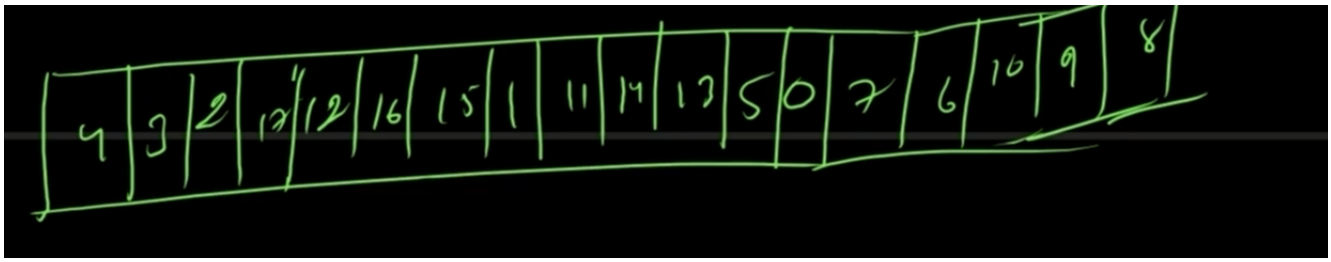


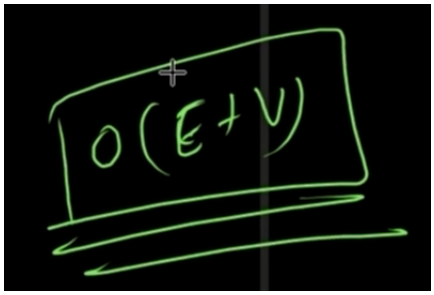


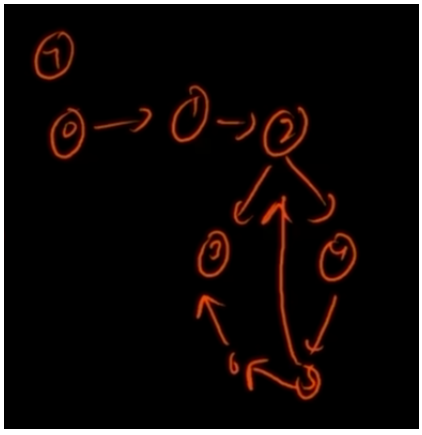


Normal Tree , just get the post order and check the topological order is right or wrong

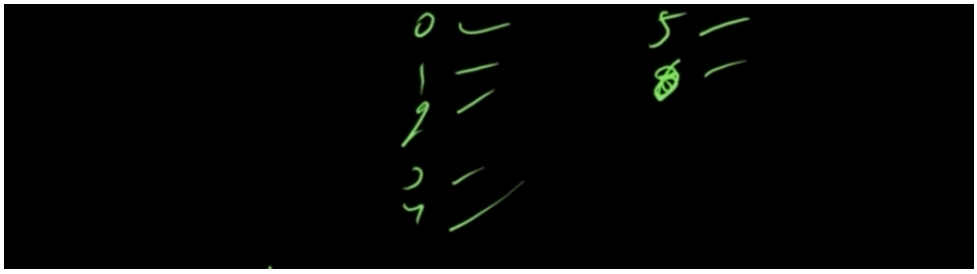


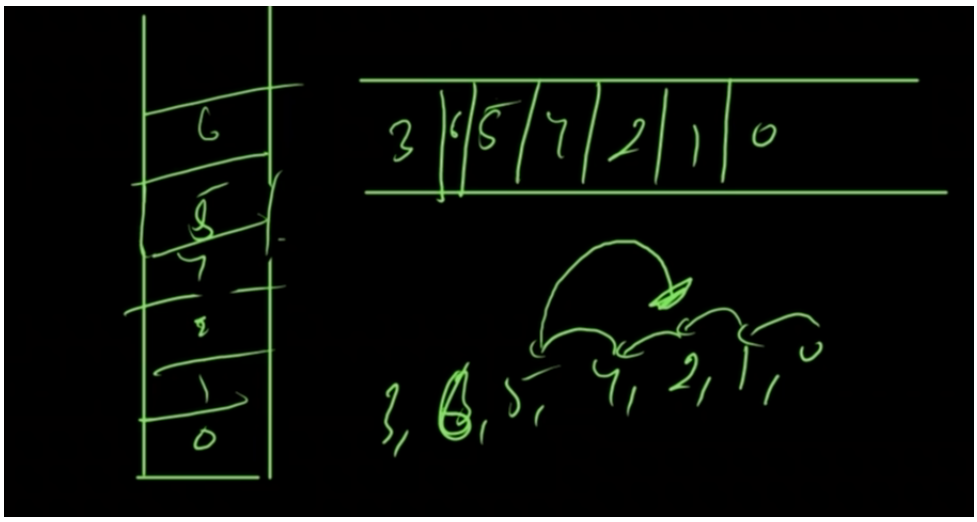






With Cycle





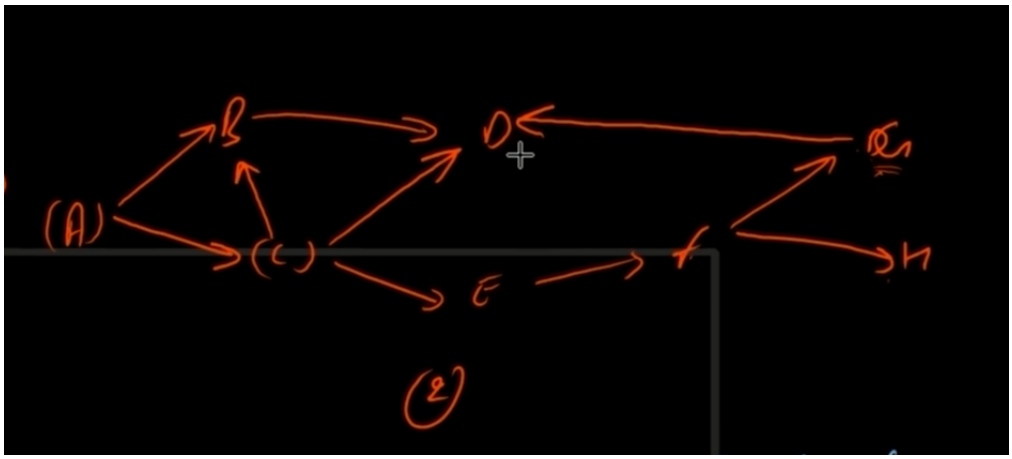
23. Kahn's Algorithm

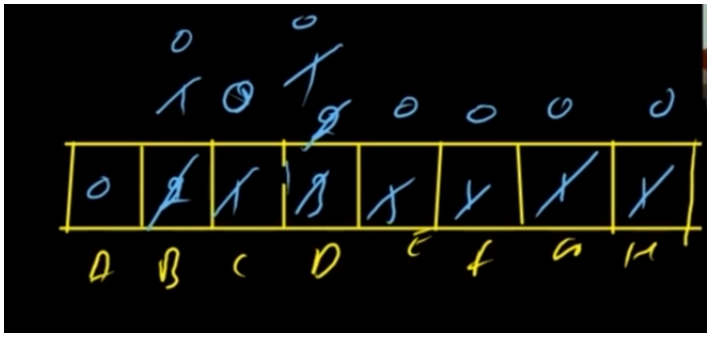
```
// b<=============== Khan's Algorithm ========================>

// Since the above normal postorder dfs does not determine the correct order in
// the
// case of cycle present in graph, therefore khan's algo helps to get the
// correct
// topological order

// # Indegree : Merepe kitne log dependent hain (Merepe kitne pointed arrows aa
// # rahe hain)

// # outDegree : Mai kitno pe dependent hun. (Merese kitne outer arrows nikal
// # rahe hain)
```

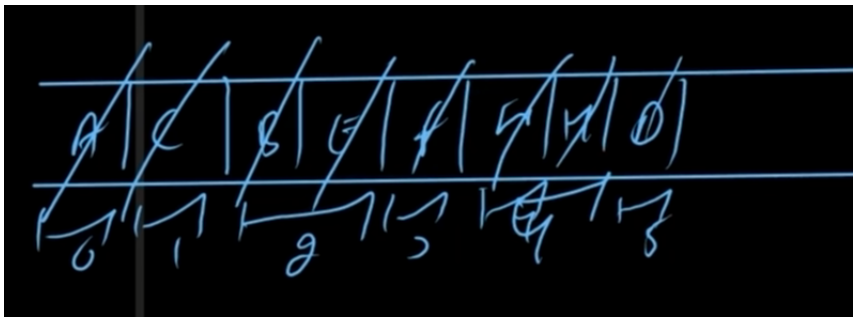
```
// ? Steps in Kahn's Algo :

// 1. Find the indegree of every vertices
// 2. Jitno ki bhi indergee 0 hai unhe que mai dala
// 3. Ab jaise he que se element nikala, jo wo vertex jin logon pe dependent hun
// na mai unke edges ko relax karke aata hun. Matlab mai man leta hun ki wo kaam
// ho jayega. To maine jin edges ko relax kiya unki vertex mai jake -1 karunga
// kyunki ab mai unpe dependent nhi raha.

// To mai aisa maan ke chal raha hun ki mera kaam to ho he jayega agar mai jin
// logon pe denpendent hun agar wo apna kaam shi se karlein to. To ab mai tumpe
// dependent rah he nhi isiliye -1 kiya

// 4. Jiski indegree ki value 0 ho jayegi, use utha ke queue mai dal denge
// kyunki ab wo independent and resolve ho gya hai isiliye

// 5. Jaise he hum queue se element nikalte hain use usi time ek arraylist mai
// store karte jayege taki hume topological order mil jaye

// 6. Aur ab agar end mai jo answer humne banaya hai uska size agar number of
// vertices ke equal nhi hai, to iska matlab sari dependencies resolve nhi hui
// hain,to matlab cycle exist karti hai.
```

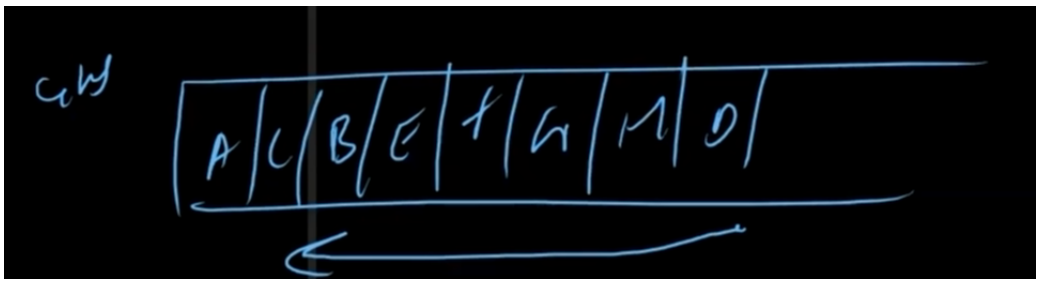
```
public static ArrayList<Integer> kahnsAlgo(int n, ArrayList<Edge>[] graph) {
    int[] indegree = new int[n];
    for (ArrayList<Edge> edgesList : graph) {
        for (Edge e : edgesList) {
            indegree[e.v]++;
        }
    }

    LinkedList<Integer> que = new LinkedList<>();
    ArrayList<Integer> ans = new ArrayList<>();

    for (int i = 0; i < n; i++)
        if (indegree[i] == 0)
            que.addLast(i);

    int level = 0;
    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int vtx = que.removeFirst();
```

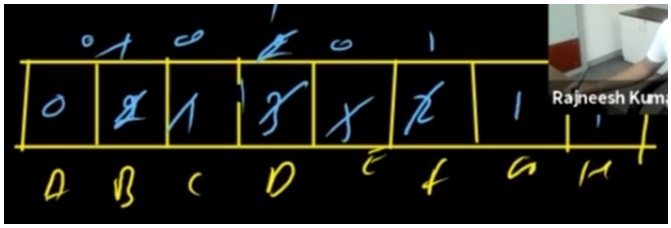
```
            ans.add(vtx);

            for (Edge e : graph[vtx]) {
                if (--indegree[e.v] == 0)
                    que.addLast(e.v);
            }

        }
        level++;
    }

    if (ans.size() != n) {
        System.out.println(x: "Topological Order is not possible due to Cycle");
        ans.clear();
    }

    return ans;
}
```

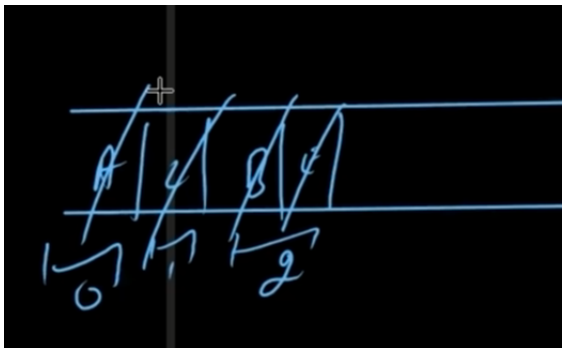






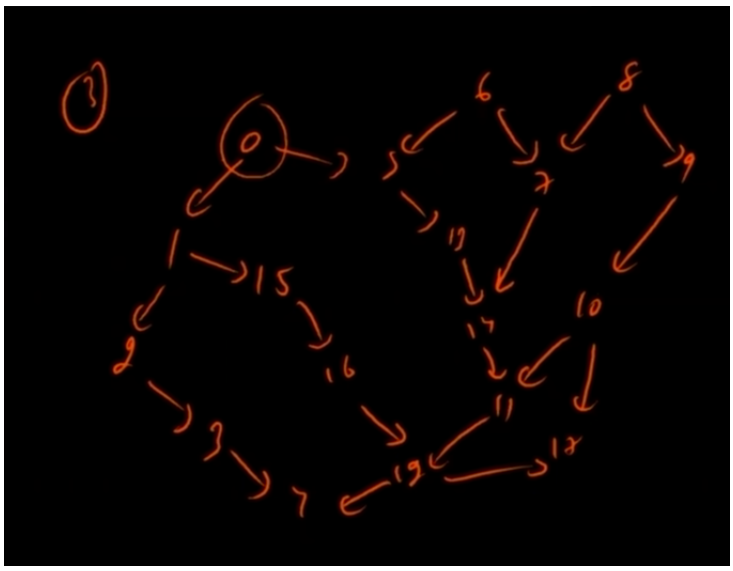


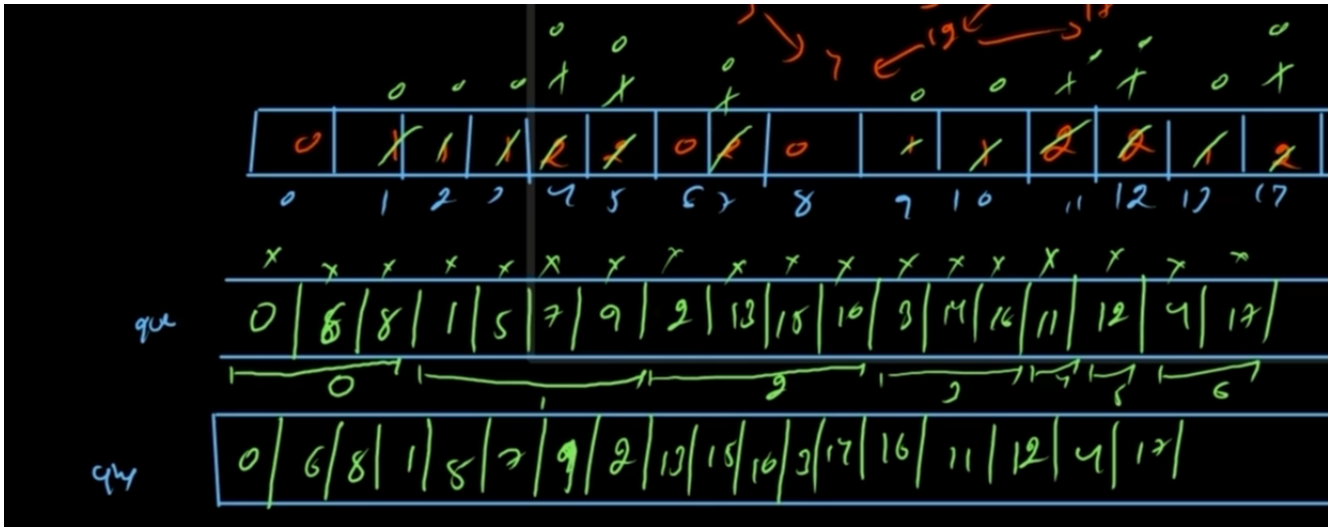


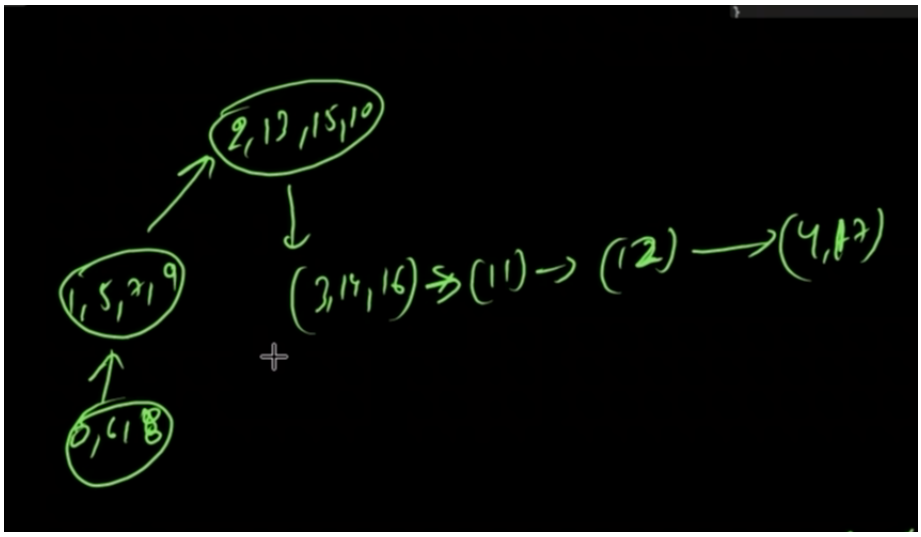


24. Level in Kahn's algo

```
// ! Important Point : What does the level indicate in the kahn's algo ???

// # Ek Level ke andar jo elements hai ye indicate karta hai wo sari process
// # jinko aap parallely kar sakte ho to save time and money because they are
// # independent of each other.

// So if we consider them machine, so the all the machine that come in the same
// level, they can be run parallel(simultaneously) with each other since they
// are independent of each other considering that their dependencies are
// resolved in the previous level. Hence we can save time for the whole process.

// ! Very important point, many questions can be resolved with this.

''
```







```java
//b <=============Kahn's Algo with Level importance ==================>

public static void kahnsAlgo_01(int n, ArrayList<Edge>[] graph) {
    int[] indegree = new int[n];
    for (ArrayList<Edge> edgesList : graph) {
        for (Edge e : edgesList) {
            indegree[e.v]++;
        }
    }

    LinkedList<Integer> que = new LinkedList<>();
    ArrayList<ArrayList<Integer>> ans = new ArrayList<>();

    for (int i = 0; i < n; i++)
        if (indegree[i] == 0)
            que.addLast(i);

    int level = 0;
    while (que.size() != 0) {
        int size = que.size();
        ArrayList<Integer> smallAns = new ArrayList<>();
        while (size-- > 0) {
            int vtx = que.removeFirst();
            smallAns.add(vtx);

            for (Edge e : graph[vtx]) {
                if (--indegree[e.v] == 0)
                    que.addLast(e.v);
            }

        }
        ans.add(smallAns);
        level++;
    }
```

```
        if (ans.size() != n) {
            System.out.println(x: "Topological Order is not possible due to Cycle");
            ans.clear();
        }
    }
```

25. Course Schedule

207. Course Schedule

Medium 11235 438 Add to List Share

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

Example 2:

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should
also have finished course 1. So it is impossible.
```

Constraints:

- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= 5000`
- `prerequisites[i].length == 2`
- `0 <= a_i, b_i < numCourses`
- All the pairs prerequisites[i] are **unique**.

```
// Here used the same kahn's algo
public boolean canFinish(int n, int[][] prerequisites) {

    ArrayList<Integer>[] graph = new ArrayList[n];
    for (int i = 0; i < n; i++) {
        graph[i] = new ArrayList<>();
    }

    int[] inDegree = new int[n];
    for (int[] e : prerequisites) {
        graph[e[0]].add(e[1]);
        inDegree[e[1]]++;
    }

    ArrayList<Integer> ans = new ArrayList<>();// This is used for more explanation. Rather than this, use a vertex
                                               // count variable to count the number of vertex that are resolved.
    LinkedList<Integer> que = new LinkedList<>();
    for (int i = 0; i < n; i++) {
        if (inDegree[i] == 0) {
            que.addLast(i);
        }
    }
    int vertexCount = 0;

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int rvtx = que.removeFirst();
            ans.add(rvtx);
            vertexCount++;

            for (int e : graph[rvtx]) {
                if (--inDegree[e] == 0) {
```

```
                que.addLast(e);
            }
        }
    }
}
// return vertexCount==n;
if (ans.size() != n) {
    return false;
}
return true;
}
```

26. Course Schedule II

210. Course Schedule II

Medium 7824 263 Add to List Share

There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a_i, b_i]` indicates that you **must** take course `b_i` first if you want to take course `a_i`.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return *the ordering of courses you should take to finish all courses*. If there are many valid answers, return **any** of them. If it is impossible to finish all courses, return **an empty array**.

Example 1:

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: [0,1]
Explanation: There are a total of 2 courses to take. To take course 1 you should have
finished course 0. So the correct course order is [0,1].
```

Example 2:

```
Input: numCourses = 4, prerequisites = [[1,0],[2,0],[3,1],[3,2]]
Output: [0,2,1,3]
Explanation: There are a total of 4 courses to take. To take course 3 you should have
finished both courses 1 and 2. Both courses 1 and 2 should be taken after you finished
course 0.
So one correct course order is [0,1,2,3]. Another correct ordering is [0,2,1,3].
```

Example 3:

```
Input: numCourses = 1, prerequisites = []
Output: [0]
```

Constraints:

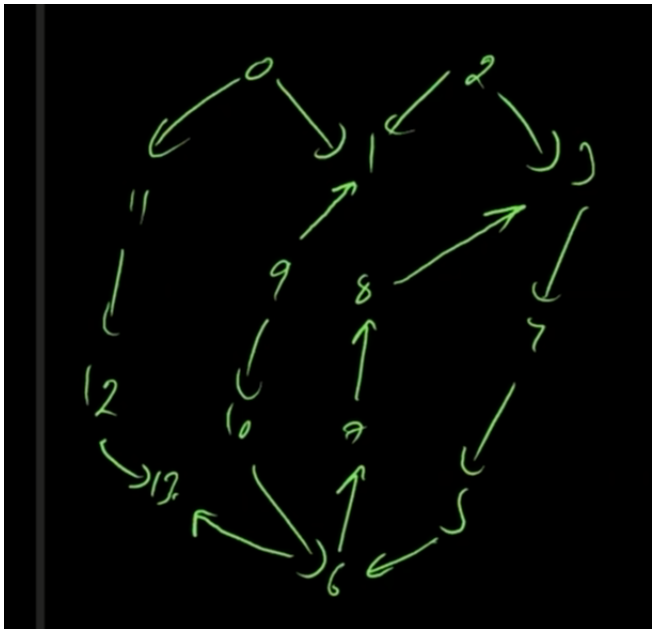
- `1 <= numCourses <= 2000`
- `0 <= prerequisites.length <= numCourses * (numCourses - 1)`
- `prerequisites[i].length == 2`
- `0 <= a_i, b_i < numCourses`
- `a_i != b_i`
- All the pairs `[a_i, b_i]` are **distinct**.

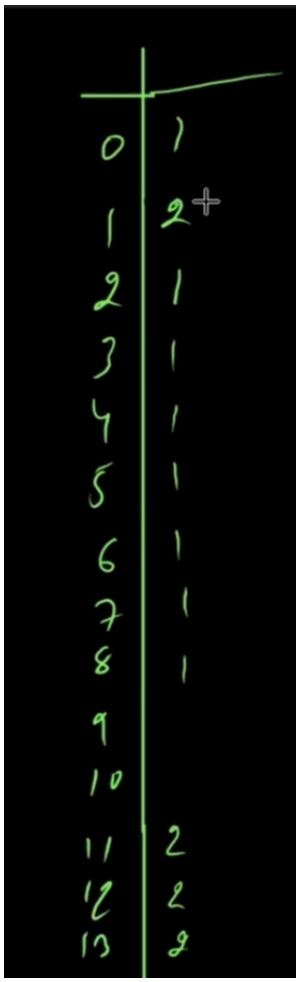
```
// b <<========== Course Schedule II ==================>
// https://leetcode.com/problems/course-schedule-ii/

// Here used Kahn's algo

public int[] findOrder(int n, int[][] prerequisites) {
    ArrayList<Integer>[] graph = new ArrayList[n];

    for (int i = 0; i < n; i++) {
        graph[i] = new ArrayList<>();
    }

    int[] inDegree = new int[n];
```

```
        for (int[] e : prerequisites) {
            graph[e[0]].add(e[1]);
            inDegree[e[1]]++;
        }

        // Rather than reversing the answer at the end, we can change the edges to get
        // the right answer

        // for (int[] e : prerequisites) {
        // graph[e[1]].add(e[0]);
        // inDegree[e[0]]++;
        // }

        LinkedList<Integer> que = new LinkedList<>();
        ArrayList<Integer> ans = new ArrayList<>();

        for (int i = 0; i < n; i++) {
            if (inDegree[i] == 0) {
                que.addLast(i);
            }
        }
```

```
    int level = 0;

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int rvtx = que.removeFirst();
            ans.add(rvtx);

            for (int v : graph[rvtx]) {
                if (--inDegree[v] == 0) {
                    que.addLast(v);
                }
            }
        }
        level++;
    }

    if (ans.size() != n) {
        ans.clear();
    }

    int[] myans = new int[ans.size()];
    int count = 0;
    for (int i = ans.size() - 1; i >= 0; i--) {
        myans[count++] = ans.get(i);
    }

    return myans;
}
```

27. Cycle Detection using dfs






```
// b<---------- dfs cycle detection ---------->
```

```
// D(---------- dfs cycle detection ----------)

// Simply hum ek path ka array bhi use kar sakte the along with visited, to
// indicate that the vertex is the part of the current path. But the same thing
// can be handled using the integer array which we can mark 0, 1, 2 values.

// 0 for unvisited
// 1 for is part of the current path and tells us that the vertex has been
// visited
// 2 for after backtrack indicating that the vertex has been visited earlier

// Jaise he mai kisi vertex mai aaya maine us 1 se mark kiya indicating ki wo
// mere current path ka part hai. Ab jaise he mai usse backtrack karunga, mai
// ab use 2 se mark karunga indicating that ki ye vertex visit ho chuki hai per
// ye ab mere current path ka part he nhi hai
```

```
public static boolean dfs_findOrder_isCyclePresent(ArrayList<Integer>[] graph, int src, ArrayList<Integer> ans,
        int[] vis) {

    vis[src] = 1;

    boolean res = false;
    for (int v : graph[src]) {
        if (vis[v] == 1) { // Agar jis vertex pe jaa rha hai wo already path ka part hai to matlab wo cycle
                           // hai
            return true;
        } else if (vis[v] == 0) {
            res = res || dfs_findOrder_isCyclePresent(graph, v, ans, vis);
        }
    }

    ans.add(src);
    vis[src] = 2;
    return res;
}
```

```
// Leetcode 210
public int[] findOrder_(int n, int[][] prerequisites) {

    ArrayList<Integer>[] graph = new ArrayList[n];
    for (int i = 0; i < n; i++) {
        graph[i] = new ArrayList<>();
    }

    for (int[] e : prerequisites) {
        graph[e[0]].add(e[1]);
    }

    ArrayList<Integer> ans = new ArrayList<>();
    int[] vis = new int[n];

    boolean isCycle = false;
    for (int i = 0; i < n; i++) {
        if (vis[i] == 0) {
            isCycle = isCycle || dfs_findOrder_isCyclePresent(graph, i, ans, vis);
        }
    }

    if (isCycle)
        return new int[] {};

    int[] myans = new int[ans.size()];
    int count = 0;
    for (int i = 0; i < ans.size(); i++) {
        myans[count++] = ans.get(i);
    }

    return myans;
}
```

28. Kahn's Algo in 2-d Matrix (329. Longest Increasing Path in a Matrix)

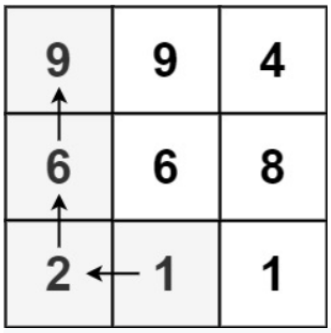
> 329. Longest Increasing Path in a Matrix

Hard 7139 109 Add to List Share

Given an `m x n` integers `matrix`, return *the length of the longest increasing path in* `matrix`.

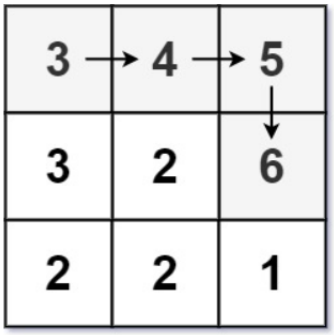
From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

Example 1:




```
Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:




```
Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not
allowed.
```

Example 3:

```
Input: matrix = [[1]]
Output: 1
```

Constraints:

- `m == matrix.length`
- `n == matrix[i].length`
- `1 <= m, n <= 200`
- `0 <= matrix[i][j] <= 2^31 - 1`

Treat a cell in the matrix as a node,
a directed edge from node x to node y if x and y are adjacent and x's value < y's value
Then a graph is formed.

No cycles can exist in the graph, i.e. a DAG is formed.

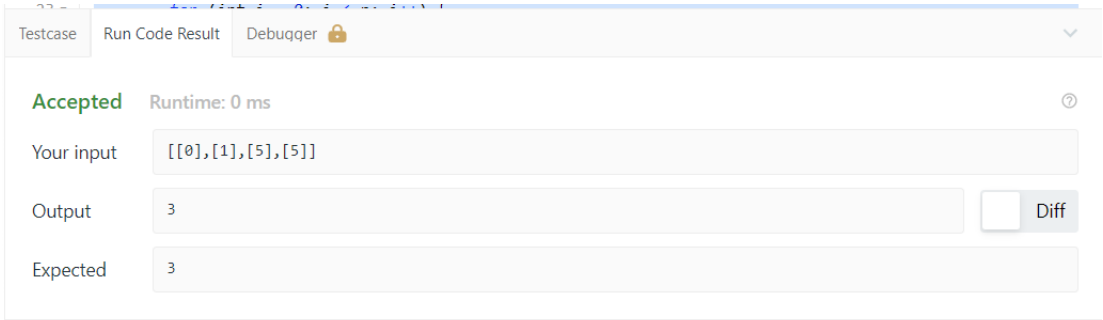
The problem becomes to get the longest path in the DAG.

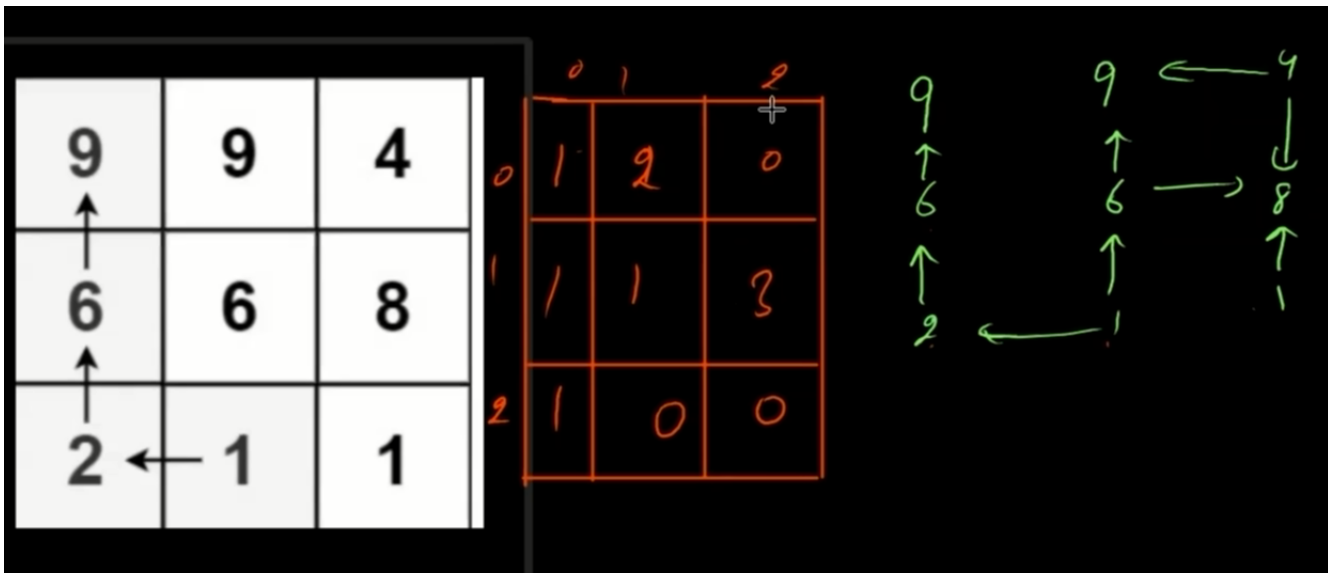
Topological sort can iterate the vertices of a DAG in the linear ordering.

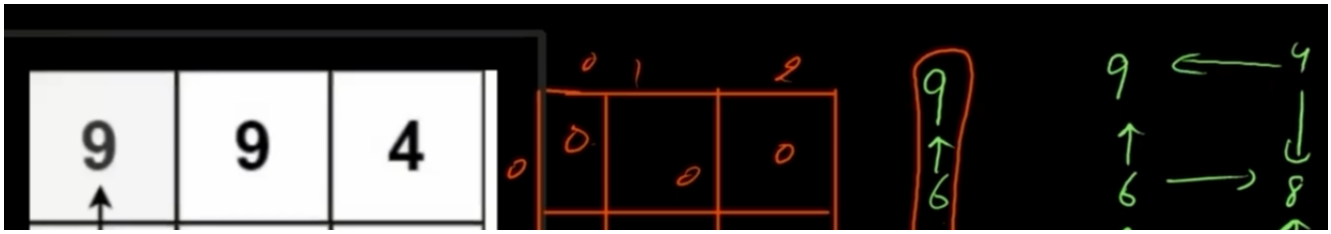
Using Kahn's algorithm(BFS) to implement topological sort while counting the levels can give us the longest chain of nodes in the DAG.

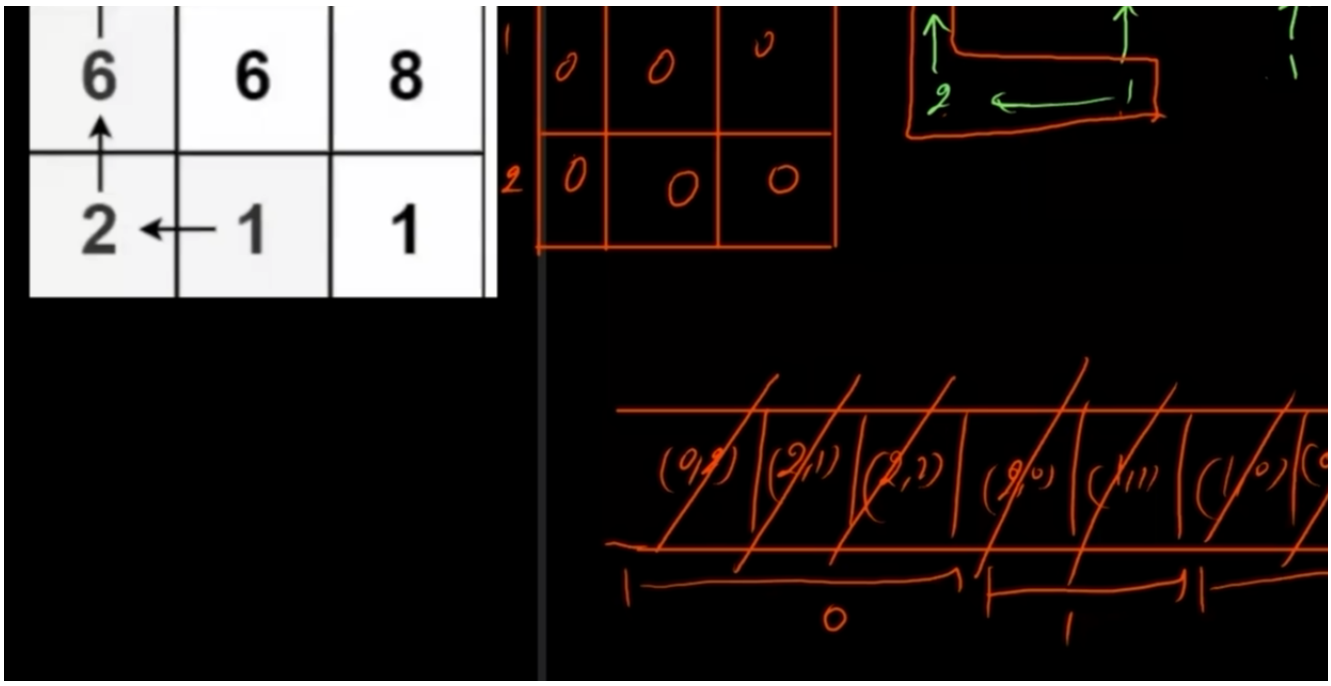
```
// b <= Kahn's Algo in 2-Matrix (329. Longest Increasing Path in a Matrix)==>
// https://leetcode.com/problems/longest-increasing-path-in-a-matrix/

// ? Treat a cell in the matrix as a node,
// ? a directed edge from node x to node y if x and y are adjacent and
// ? x's value < y's value Then a graph is formed.

// No cycles can exist in the graph, i.e. a DAG is formed.
// # DAG : Directed acyclic graph

// The problem becomes to get the longest path in the DAG.

// Topological sort can iterate the vertices of a DAG in the linear ordering.

// Here Using Kahn's algorithm(BFS) to implement topological sort while counting
// the levels can give us the longest chain of nodes in the DAG.

// ! To yahan pe kiya kya??

// Yahan pe basically humne har ek cell ko vertex ki tarah treat kiya hai aur ek
// graph mai convert kiya hai. Ab ye jo graph hoga, usme hum edges define
// karenge. Kyunki hume increasing path nikalna hai to agar koi meri adjacent
// value mujse kam hai, to wo edge uuse merepe banega.

// Aisa karke humne indegree calculate kiya hai. Ab humne kahn's algo lagayi
// simple to calculate the longest path since jo end mai level aayega wo uska
// longest path denote karege in terms of node/vertex.
```

Test case to dry run on :







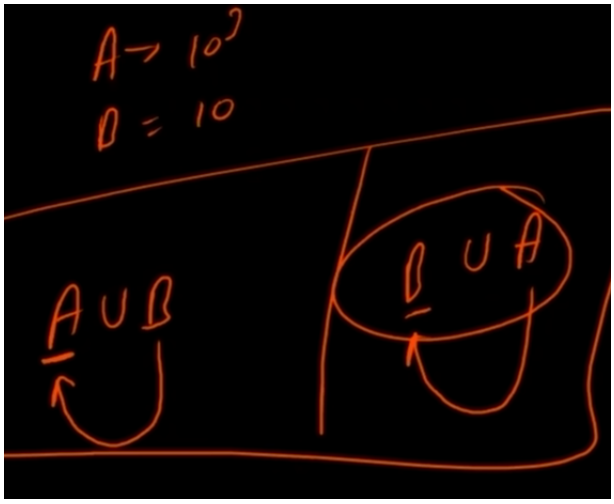



```java
public int longestIncreasingPath(int[][] matrix) {

    int n = matrix.length, m = matrix[0].length;
    int[][] indegree = new int[n][m];

    int[][] dir = { { -1, 0 }, { 0, 1 }, { 1, 0 }, { 0, -1 } };
    LinkedList<Integer> que = new LinkedList<>();
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            for (int d = 0; d < dir.length; d++) {
                int r = i + dir[d][0];
                int c = j + dir[d][1];
                if (r >= 0 && c >= 0 && r < n && c < m) {
                    if (matrix[r][c] < matrix[i][j]) // Jahan mai jaa raha hun agar uski value mujhe kam hai to meri
                                                     // indegree++ hogi kyunki wo edge usse merepe aayega. Pointer
                                                     // meri taraf aayega
                        indegree[i][j]++;
                }
            }

            if (indegree[i][j] == 0)
                que.addLast(i * m + j);
        }
    }
```

```java
    int level = 0;
    while (que.size() != 0) {
        int size = que.size();

        while (size-- > 0) {
            int rvtx = que.removeFirst();
            int sr = rvtx / m, sc = rvtx % m;

            for (int d = 0; d < dir.length; d++) {
                int r = sr + dir[d][0];
                int c = sc + dir[d][1];

                // Hum sirf wahin jayenge jiski value humse badi hogi
                if (r >= 0 && c >= 0 && r < n && c < m && matrix[r][c] > matrix[sr][sc]) { // Dry run on test case
                                                                                           // [[0],[1],[5],[5]] to
                                                                                           // figure out the last
                                                                                           // check
                    if (--indegree[r][c] == 0) {
                        que.addLast(r * m + c);
                    }
                }
            }
        }

        level++;
    }
```

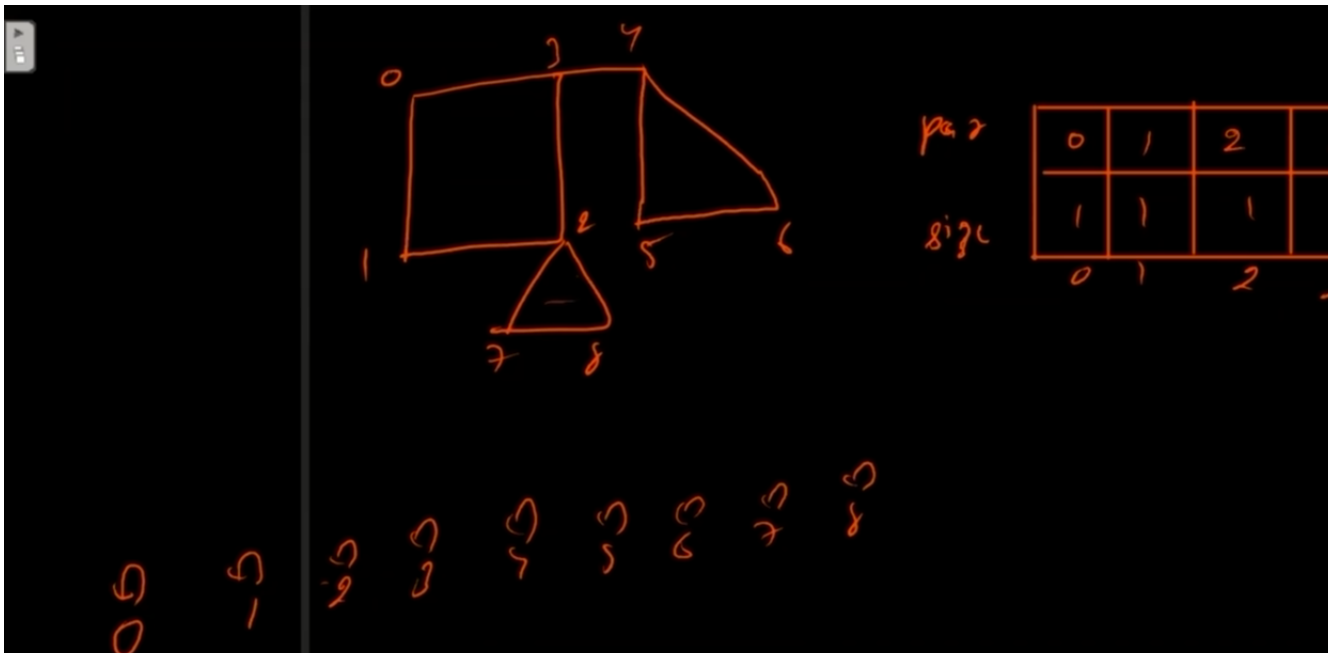
```
    return level;
}
```

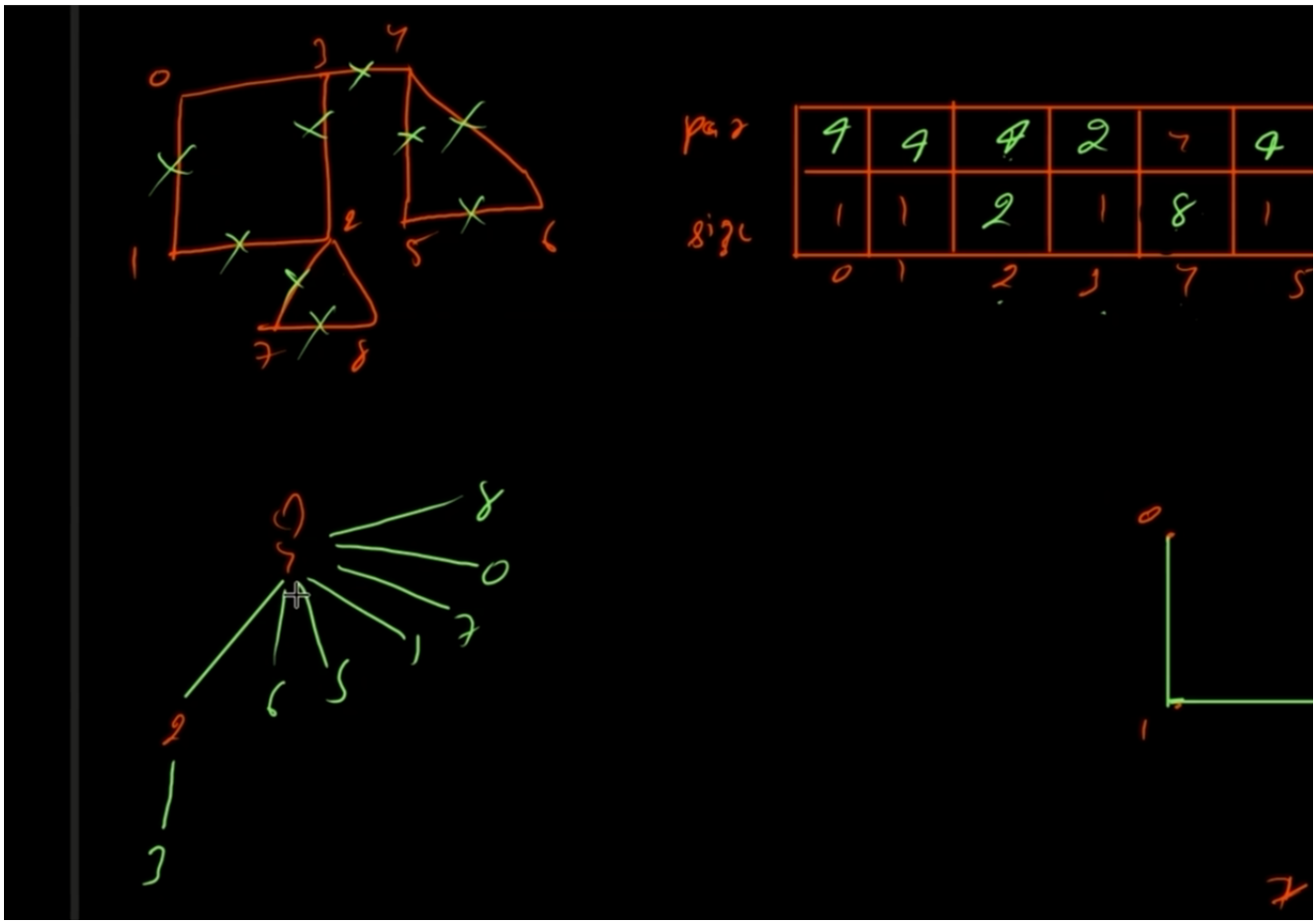
29. DSU (Disjoint set Union) {Union Find Algo}




```
// b <============== Union Find Algo ==============>

// ? It works better when the edges are gives. The whole graph is not required.

// Here AUB(A union B) != BUA (B union A)

// # But why ?
// Consider A to be size of 10³ and B to be size of 10. Then adding 10³ elements
// in B is is a big operation and is of high complexity. So the answer remains
// the same but the complexity are different.
```

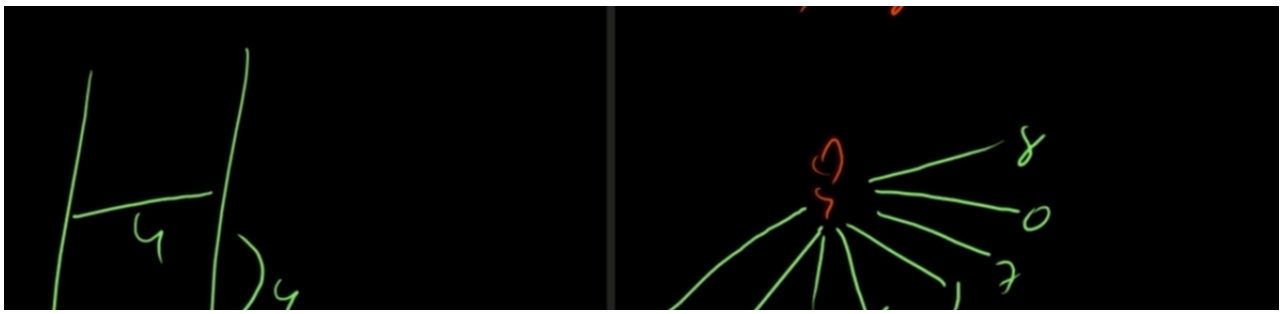
```
// # Ab union hoga kaise. Iske liye mai har ek set ka leader bana raha hun aur
// # jo bhi query hogi us set ke elements ki, wo leader se hogi

// So agar mai ek set ko country se denote karta hun to agar mai us country ka
// size puchta hun to mujhe leader answer karega. Aur koi ni.

// ! Dono ke leader alag alag hain aur unke beech mai edge exist karta hai, to
// ! matlab wo dono same country(set) ko belong karte honge.

// ! Aur agar dono ka leader same hai aur unke beech mai edge exist karti hai to
// ! cycle hai aur hum cycle wali edge ko add he nhi karte
```

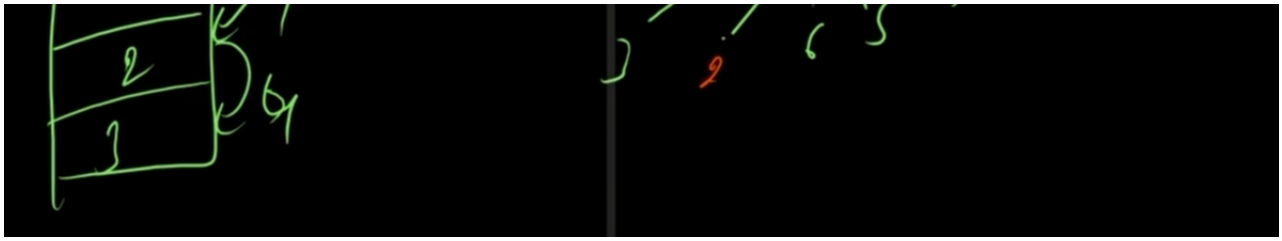


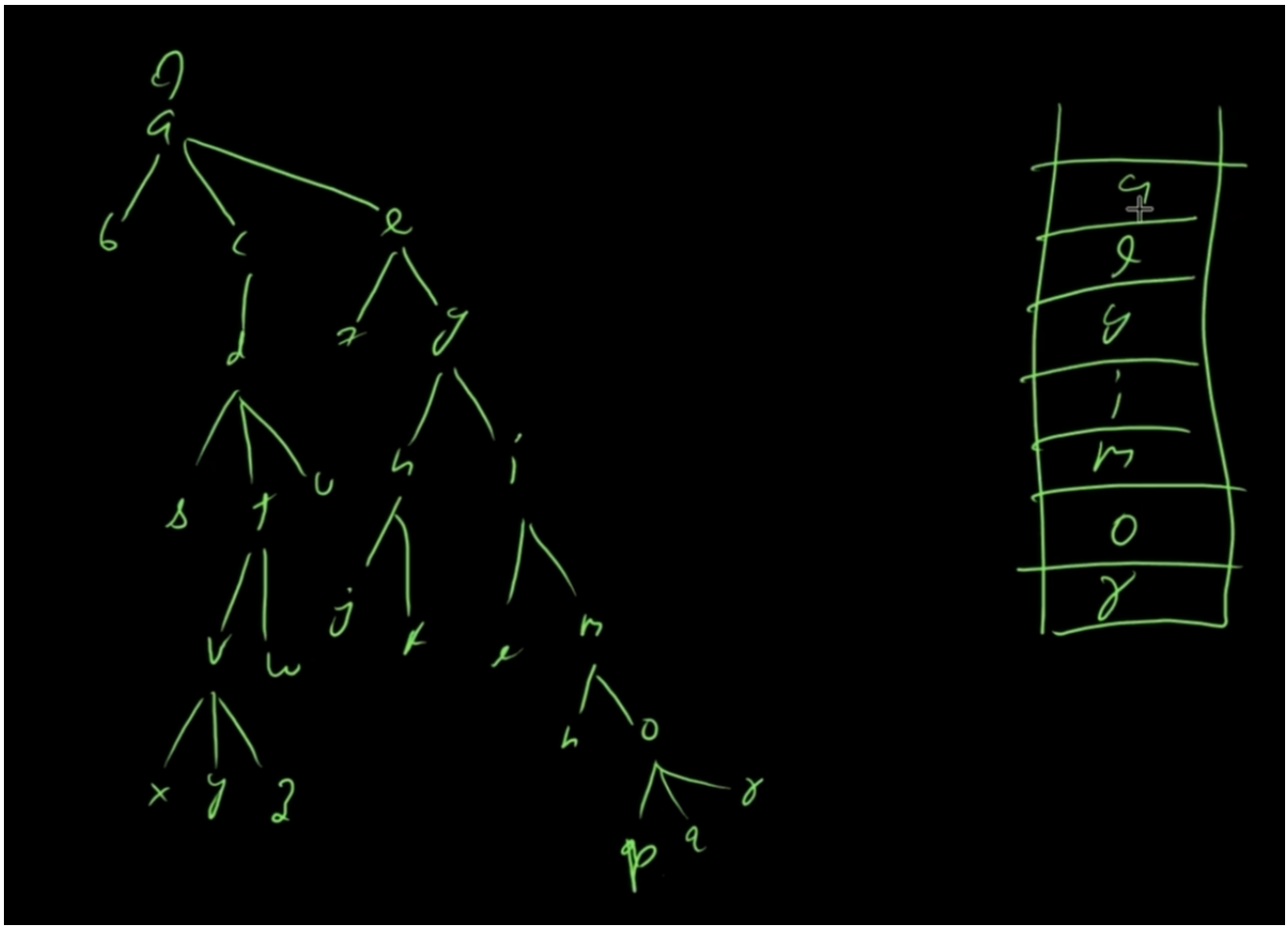


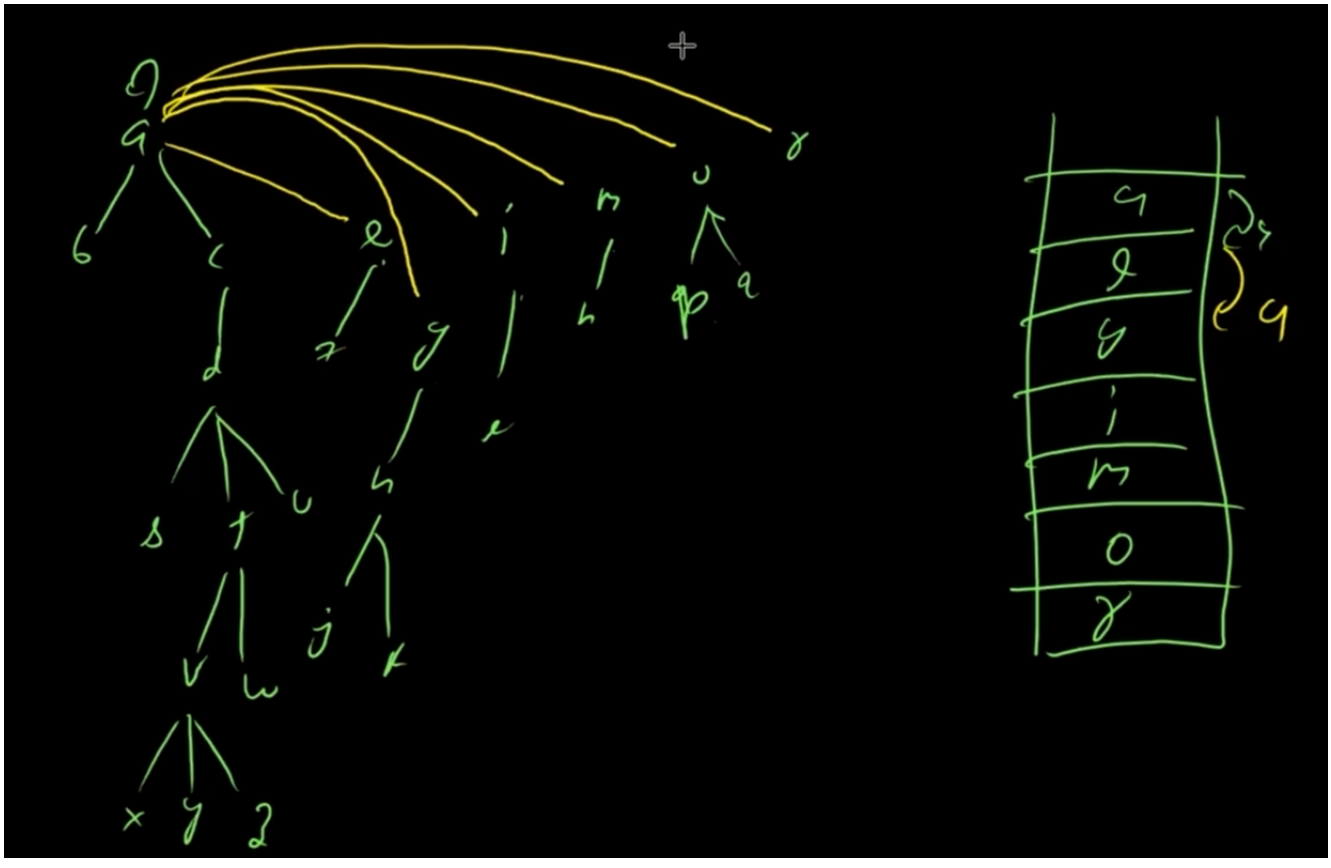
Path Compression

```
// # Now what we do is use path compression for to find the leader of the
// # childs.

// Manle ki h height ka tree hai aur us tree ka leader root node hai. Ab
// agar leaf se pucha leader ke bare mai to wo apne parent se puchega and same
// goes on till the leader is reached. Ab jaise he leader mila, aate samay sare
// root to node path ke parents honge unka leader set karte hue aayenge hum.
// Isse kya hoga ki humari searching optimize ho jayegi kyuki ab us path mai jo
// `bhi node hoga uska leader nikalna humare liye just O(1) ka operation hoga
// kunki humne aate samay leader set karte hue aaye the.

// Ab isse ek aur cheez hui ki tree ki height reduce ho gyi(taki leader jaldi
// mil jaye)

// ! Isi cheez ko path compression bolte hain. Jo humari complexity high hone se
// ! bacha leta hai.
```

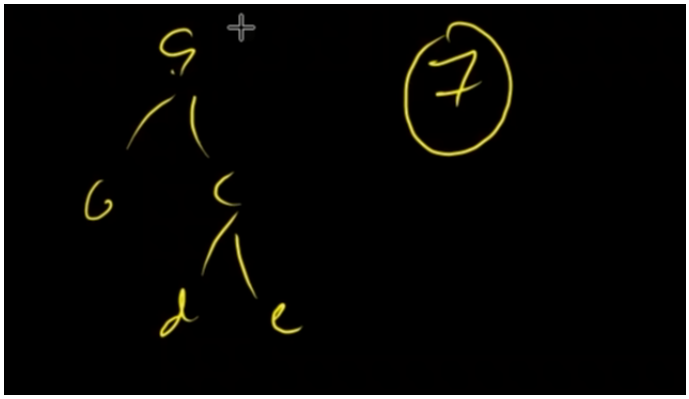








```
// # Ab agar maine parent find time parent pointers ko update nhi kiya aur maine
// # size ka bhi koi check ni lagaya (matlab tree ki height ko control karne ke
// # liye), to ek find ki complexity jati hai O(n).

// Height control karne ka matlab ye hai. Agar manle do tree hai ek ki height H
// hai aur ek single node ka hai. Aur dono ke apne apne leader hain. To join
// karte time, agar single node ke leader ko pure ka leader banayenge to combine
// tree ki height increase ho jayegi. Per agar Jyada height wale ke leader ko
// combine ka leader banaya, to height increase nhi hogi.
```



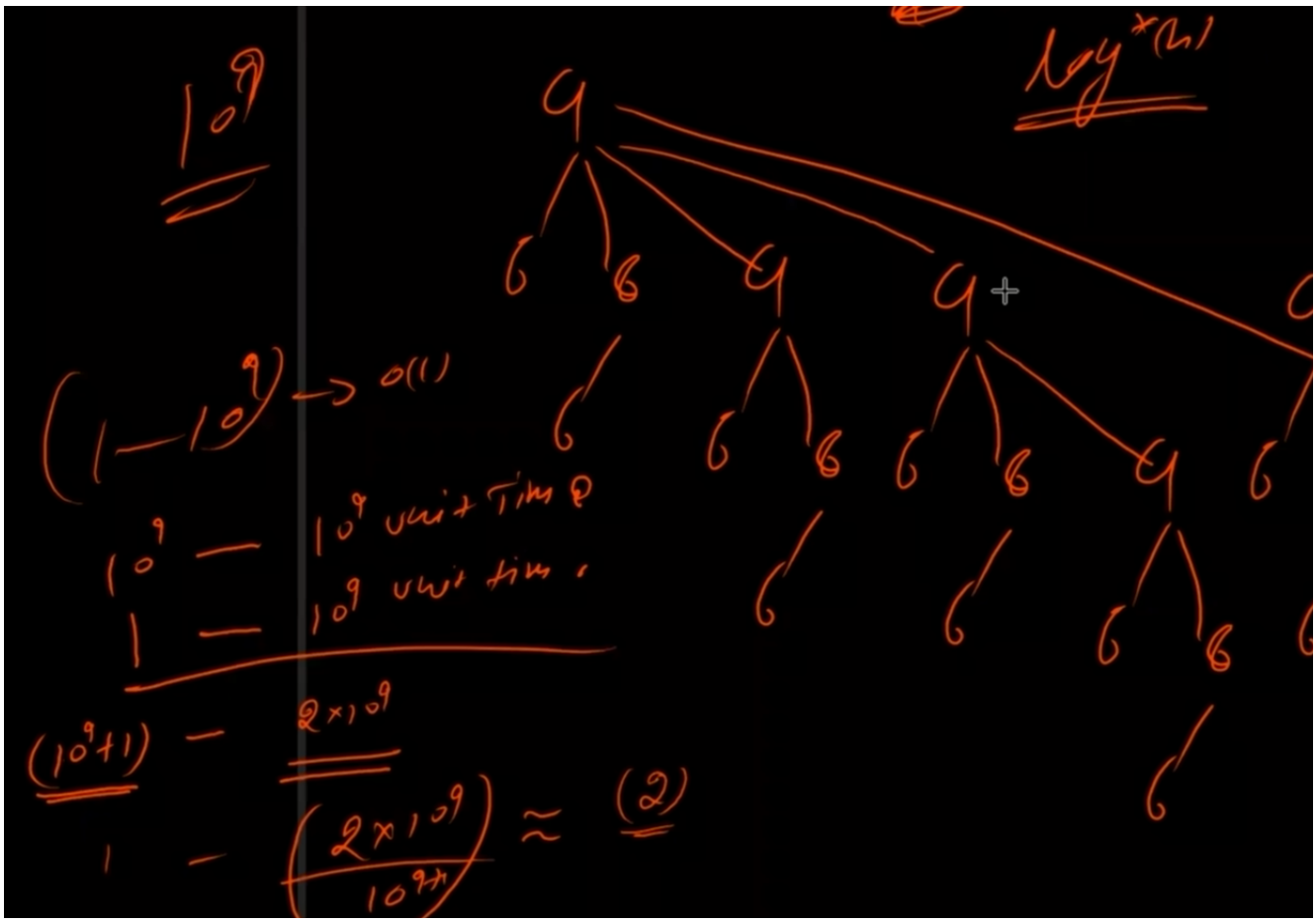
Time complexity [edit]

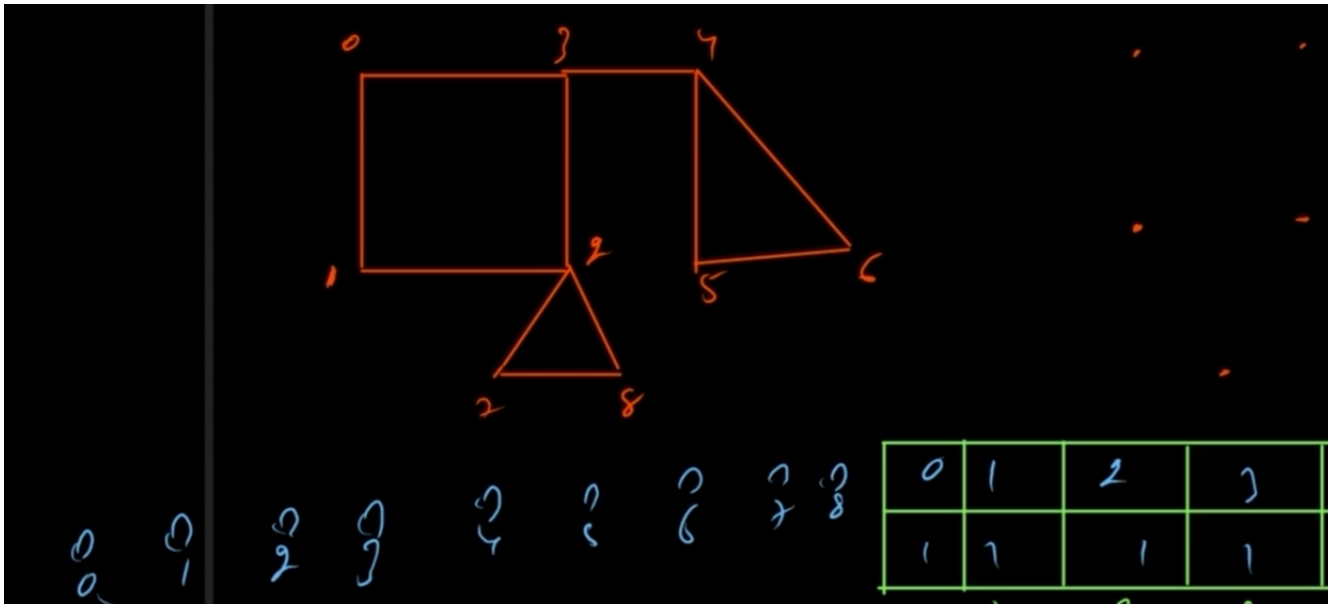
A disjoint-set forest implementation in which `Find` does not update parent pointers, and in which `Union` does not attempt to control tree heights, can have `Find` and `Union` operations require $O(n)$ time.

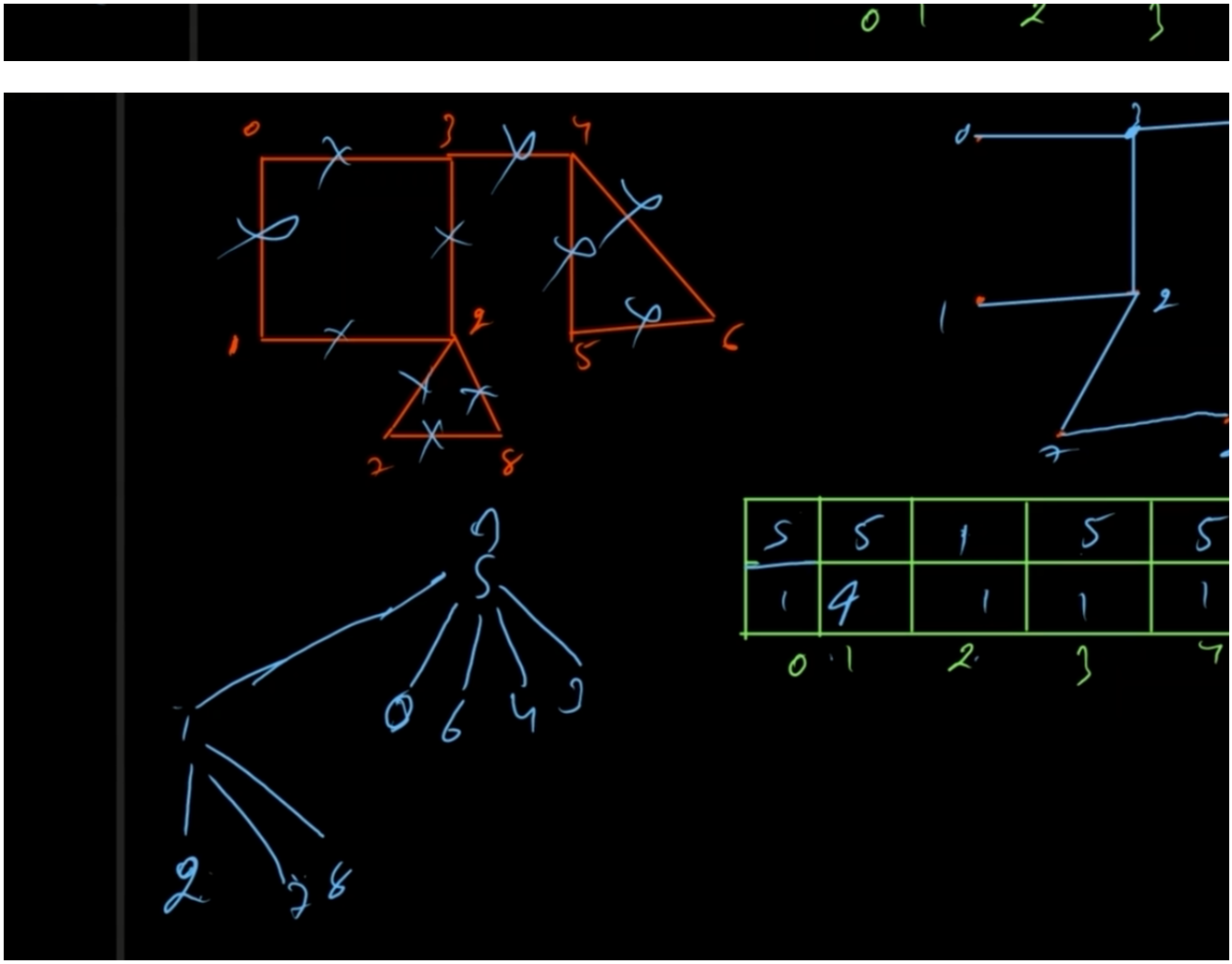
The combination of path compression, splitting, or halving, with union by size or by rank, reduces the running time for m operations of any type, up to n of which This makes the amortized running time of each operation $\Theta(\alpha(n))$. This is asymptotically optimal, meaning that every disjoint set data structure must use $\Omega(\alpha$ function $\alpha(n)$ is the inverse Ackermann function. The inverse Ackermann function grows extraordinarily slowly, so this factor is 4 or less for any n that can actua makes disjoint-set operations practically amortized constant time.

```
// ! Important Point
// Aagr hum path compression bhi use karte hain aur sath mai union karte time
// size of the tree ko bhi factor mai lete hain(matlab tree ki height control
// karne ke liye), to jo find ki complexity jati hai wo hoti hai log*(N) means
```

```
// amortized logN. and this will be always less that O(4).

// # How less than 4 . Assuming the worst case here.

// Ab manle ki ek tree hai aue use similar tree mila, to humne jab uska union
// kiya to simple leader ko join kar diya or ye sab O(1) mai hua. Aise he karte
// karte agar height 10^9 ko reach kari to humko 10^9 unit of time laga tree
// ko banane mai. Ab agar kisi leaf pe find parent ka operation hua to 10^9 unit
// of time lagega parent ko find karne mai. To agar hum total operation dekhein,
// 10^9 + 1 aur total time lage 2 * 10^9. Therefore ek operation ke liye jo time
// laga hai wo hoga O(2) which is less than 4.

// ? The find complexity will be log*(N){with size union and path compression}
// ? and log*(N) will be always less the O(4)

// ? If we ignore the path compression, the complexity becomes log(N)

// # The graph formed at the end of union find algo is called spanning tree(a
// # graph with no cycle).
```



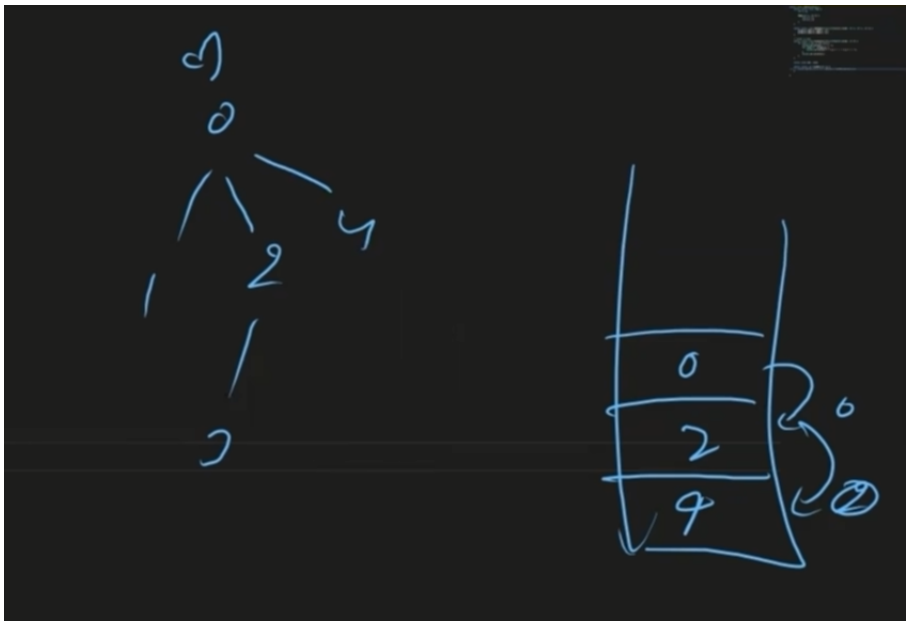




```
// ! Union Find algo basic steps :

// 1. Find the leader of the set. (To do it, use path compression is necessary)
// 2. Then while doing the union of the two sets, use size check in order to
// identify who will the new leader of the set.
// 3. If two vertices have same leader and a edge exist between them, then there
// is a cycle.
// 4. If two vertices have different leader and an edge exist between them, then
// ` both of the vertices belong to the same country(same set).
```

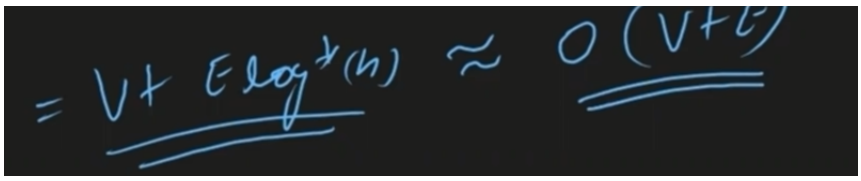
```java
// {u,v,w}
public static void unionFind(int[][] Edges, int N) {
    ArrayList<Edge>[] graph = new ArrayList[N];
    for (int i = 0; i < N; i++) {
        graph[i] = new ArrayList<>();
    }

    int[] par = new int[N];
    int[] size = new int[N];

    for (int i = 0; i < N; i++) { // Initially mera parent mai khud he hun aur mera size 1 hai
        par[i] = i;
        size[i] = 1;
    }

    for (int[] edge : Edges) {
        int u = edge[0], v = edge[1], w = edge[2];

        int p1 = findLeaderParent(u);
        int p2 = findLeaderParent(v);

        if (p1 != p2) { // dono ke leader alag hai per dono ke beech mai edge exist karti hai to dono
                        // same set ko belong karte hain

            merge_Union(p1, p2); // Sari jo questioning hogi wo leader se hogi
            addEdge(graph, u, v, w); // To create a union set graph
        }
    }
}
```

```java
static int[] par, size;

// find parent with path compression.
public static int findLeaderParent(int u) {
    return par[u] == u ? u : (par[u] = findLeaderParent(par[u])); // (par[u] = findParent(par[u])); This here used
                                                                  // is the
    // path compression.
}

public static void merge_Union(int p1, int p2) { // To merge(union) the two sets.

    if (size[p1] > size[p2]) {
        par[p2] = p1;
        size[p1] += size[p2];
    } else {
        par[p1] = p2;
        size[p2] += size[p1];
    }
}
```




```java
// {{u,v,w}}
public static void unionFind(int[][] Edges, int N) {
    ArrayList<Edge>[] graph = new ArrayList[N];
    for (int i = 0; i < N; i++)
        graph[i] = new ArrayList<>();

    par = new int[N];
    size = new int[N];
    for (int i = 0; i < N; i++) {
        par[i] = i;
        size[i] = 1;
    }

    for (int[] e : Edges) {
        int u = e[0], v = e[1], w = e[2];
        int p1 = findPar(u);
        int p2 = findPar(v);
        if (p1 != p2) {
            merge(p1, p2);
            addEdge(graph, u, v, w);
        }
    }
}
```

V + E O(4) = V + E




```
// It is a linear algo. Complexity is V + E log(N)== V + E O(4)==V + E
```

30. 684. Redundant Connection

684. Redundant Connection

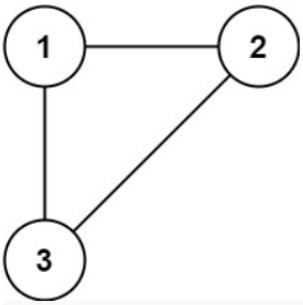
Medium 4452 317 Add to List Share

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

You are given a graph that started as a tree with `n` nodes labeled from `1` to `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 graph is represented as an array `edges` of length `n` where `edges[i] = [a_i, b_i]` indicates that there is an edge between nodes `a_i` and `b_i` in the graph.

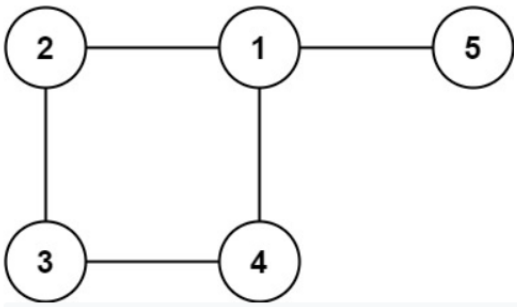
Return *an edge that can be removed so that the resulting graph is a tree of* `n` *nodes*. If there are multiple answers, return the answer that occurs last in the input.

Example 1:




```
Input: edges = [[1,2],[1,3],[2,3]]
Output: [2,3]
```

Example 2:




```
Input: edges = [[1,2],[2,3],[3,4],[1,4],[1,5]]
Output: [1,4]
```

Constraints:

- `n == edges.length`
- `3 <= n <= 1000`
- `edges[i].length == 2`
- `1 <= a_i < b_i <= edges.length`
- `a_i != b_i`

- There are no repeated edges.
- The given graph is connected.

```java
// b <===============684. Redundant Connection ====================>
// https://leetcode.com/problems/redundant-connection/

// Same union find algo lagayi. Bas merege karte time size ka check nhi lagaya,
// aise hemerge kar diya.

static int[] par;

public static int findLeaderParent(int u) { // Will find the leader of the set.
    return par[u] == u ? u : (par[u] = findLeaderParent(par[u]));
}

public int[] findRedundantConnection(int[][] edges) {
    int N = edges.length + 1;
    par = new int[N];

    for (int i = 0; i < N; i++) {
        par[i] = i;
    }

    for (int[] edge : edges) {
        int u = edge[0], v = edge[1];

        int p1 = findLeaderParent(u);
        int p2 = findLeaderParent(v);

        if (p1 != p2) {
            par[p2] = p1; // Humne union merge ki jagah pe kisi ko bhi leader set kar diya. Size ko check
                          // ni kiya. Isse complexity O(6 ya 7) pahunchti hai max. par[p1] = p2; bhi likh
                          // dete to code same he run karta.
        } else {
            return new int[] { u, v };
        }
    }

    return new int[] {};
}
```

31. Lexicographically Smallest Equivalent String

1061. Lexicographically Smallest Equivalent String

Medium 150 8 Add to List Share

You are given two strings of the same length `s1` and `s2` and a string `baseStr`.

We say `s1[i]` and `s2[i]` are equivalent characters.

- For example, if `s1 = "abc"` and `s2 = "cde"`, then we have `'a' == 'c'`, `'b' == 'd'`, and `'c' == 'e'`.

Equivalent characters follow the usual rules of any equivalence relation:

- **Reflexivity:** `'a' == 'a'`.
- **Symmetry:** `'a' == 'b'` implies `'b' == 'a'`.
- **Transitivity:** `'a' == 'b'` and `'b' == 'c'` implies `'a' == 'c'`.

For example, given the equivalency information from `s1 = "abc"` and `s2 = "cde"`, `"acd"` and `"aab"` are equivalent strings of `baseStr = "eed"`, and `"aab"` is the lexicographically smallest equivalent string of `baseStr`.

Return *the lexicographically smallest equivalent string of* `baseStr` *by using the equivalency information from* `s1` *and* `s2`.

Example 1:

```
Input: s1 = "parker", s2 = "morris", baseStr = "parser"
Output: "makkek"
Explanation: Based on the equivalency information in s1 and s2, we can group their
characters as [m,p], [a,o], [k,r,s], [e,i].
The characters in each group are equivalent and sorted in lexicographical order.
So the answer is "makkek".
```

Example 2:

```
Input: s1 = "hello", s2 = "world", baseStr = "hold"
Output: "hdld"
Explanation: Based on the equivalency information in s1 and s2, we can group their
characters as [h,w], [d,e,o], [l,r].
So only the second letter 'o' in baseStr is changed to 'd', the answer is "hdld".
```

Example 3:

```
Input: s1 = "leetcode", s2 = "programs", baseStr = "sourcecode"
Output: "aauaaaaada"
Explanation: We group the equivalent characters in s1 and s2 as [a,o,e,r,s,c], [l,p],
[g,t] and [d,m], thus all letters in baseStr except 'u' and 'd' are transformed to 'a',
the answer is "aauaaaaada".
```

Constraints:

- `1 <= s1.length, s2.length, baseStr <= 1000`
- `s1.length == s2.length`
- `s1`, `s2`, and `baseStr` consist of lowercase English letters.

```
// b<===========1061. Lexicographically Smallest Equivalent String =====>
// https://leetcode.ca/all/1061.html

// So kiya ye ki pattern find kiya.

// Agar hum edges create karlein since A[i] and B[i] are equivalent characters.
// To jinke bhi beech mai edge hogi wo sab same country ko belong karte honge.
// Aur country mai jo sabse chota hoga, use uska leader set kar denge to
// lexographically order mil jayega kyunki sabse chota he chahiye humesha humko.

// Taki easy ho jaye humne integer mai convert kar rahe hai taki array use kar
// payein aur fast rahe. Baki logic same union find ka hai.
```

```
public static int findLeader_(int u) {
    return par[u] == u ? u : (par[u] = findLeader_(par[u]));
}

public String smallestEquivalentString(String s1, String s2, String baseStr) {

    ArrayList<int[]> edges = new ArrayList<>();
    for (int i = 0; i < s1.length(); i++) {
        char ch1 = s1.charAt(i);
        char ch2 = s2.charAt(i);

        edges.add(new int[] { ch1 - 'a', ch2 - 'a' });
    }

    par = new int[26];
    for (int i = 0; i < 26; i++) {
        par[i] = i;
    }

    for (int[] edge : edges) {
        int u = edge[0], v = edge[1];
        int p1 = findLeader_(u);
        int p2 = findLeader_(v);

        if (p1 != p2) {
            if (p1 < p2) {
                par[p2] = p1;
            } else {
                par[p1] = p2;
            }
        }
    }
```

```
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < baseStr.length(); i++) {
        char ch = baseStr.charAt(i);
        int parent = findLeader_(ch - 'a');
        sb.append((char) ('a' + parent));
    }

    return sb.toString();
```

```
}
```

```
// Bhaiya Method :

public String smallestEquivalentString_(String s1, String s2, String baseStr) { // Kiya same hai meri tarah
    for (int i = 0; i < 26; i++)
        par[i] = i;
    for (int i = 0; i < s1.length(); i++) { // Maine bas edges create kari thi more understanding ke liye aur taki
                                            // code same rahe, baki same jo tune kiya hai. Edges na create karke
                                            // direct aise bhi kar sakte hain, jyada better way hai
        char c1 = s1.charAt(i), c2 = s1.charAt(i);
        int p1 = findLeader_(c1 - 'a'), p2 = findLeader_(c2 - 'a');
        par[p1] = Math.min(p1, p2);
        par[p2] = Math.min(p1, p2);
    }

    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < baseStr.length(); i++) {
        char ch = baseStr.charAt(i);
        int parent = findLeader_(ch - 'a');
        sb.append((char) ('a' + parent));
    }

    return sb.toString();
}
```

32. Similar String Groups

839. Similar String Groups

Hard 955 178 Add to List Share

Two strings `X` and `Y` are similar if we can swap two letters (in different positions) of `X`, so that it equals `Y`. Also two strings `X` and `Y` are similar if they are equal.

For example, `"tars"` and `"rats"` are similar (swapping at positions `0` and `2`), and `"rats"` and `"arts"` are similar, but `"star"` is not similar to `"tars"`, `"rats"`, or `"arts"`.

Together, these form two connected groups by similarity: `{"tars", "rats", "arts"}` and `{"star"}`. Notice that `"tars"` and `"arts"` are in the same group even though they are not similar. Formally, each group is such that a word is in the group if and only if it is similar to at least one other word in the group.

We are given a list `strs` of strings where every string in `strs` is an anagram of every other string in `strs`. How many groups are there?

Example 1:

```
Input: strs = ["tars","rats","arts","star"]
Output: 2
```

Example 2:

```
Input: strs = ["omv","ovm"]
Output: 1
```

Constraints:

- `1 <= strs.length <= 300`
- `1 <= strs[i].length <= 300`
- `strs[i]` consists of lowercase letters only.
- All words in `strs` have the same length and are anagrams of each other.

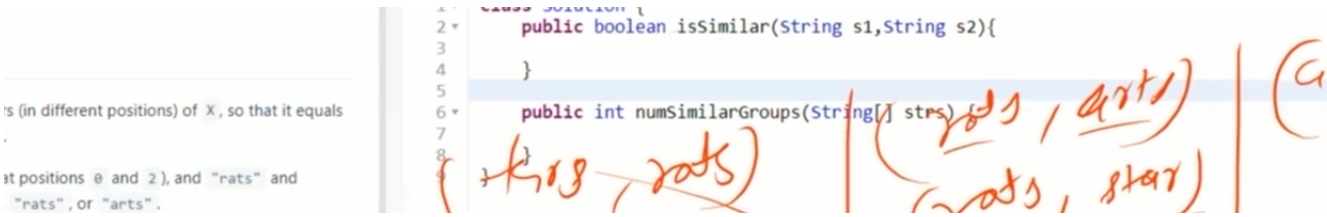
```
// b <===============839. Similar String Groups ============>
// https://leetcode.com/problems/similar-string-groups/

// Jab bhi hum union find lagane ki sochte hain, to hum edges dhundhte hain
// To ab jitne bhi combination ho sakte the, humne unko edge manke chalaya.
// Ab lekin jo similar hai unko he group karna tha, to bas tabhi he group kiya.
```

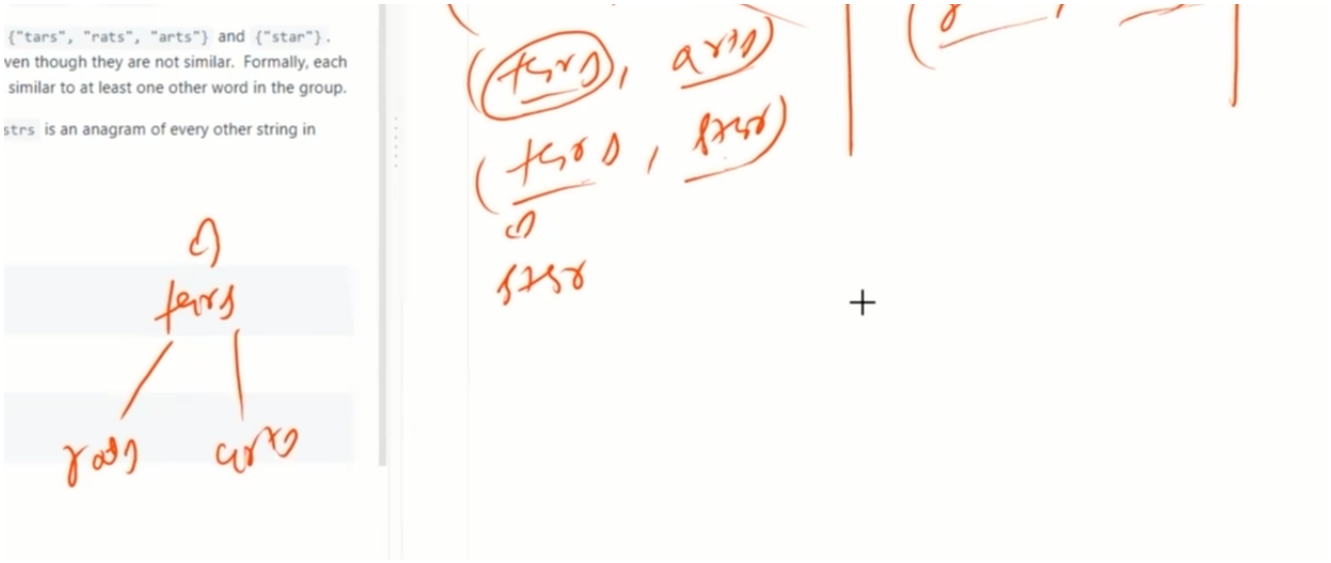
```
// To Baki pura same hai union find algo ki tarah. Same concept.

// Parent ka array bhi integer rakha kyunki string ka role bas similar check
// karne mai hai jise hum given string ke array se nikal sakte hain easily.
// Therefore index he vertex(yahan pe string) ko denote kar raha hai.
```

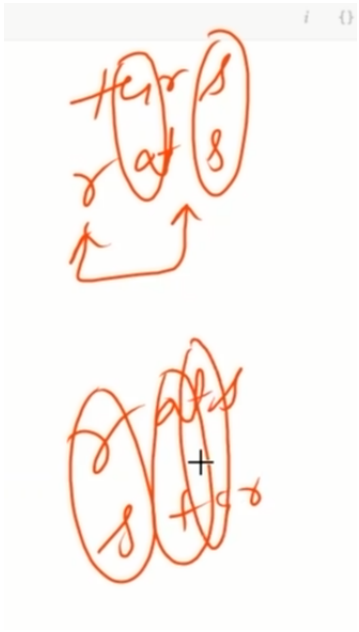
```
public int numSimilarGroups(String[] strs) {

    int n = strs.length;

    for (int i = 0; i < n; i++) {
        par[i] = i;
    }

    int numberOfSets = n;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (isSimilar(strs[i], strs[j])) {
                int p1 = findLeaderParent(i);
                int p2 = findLeaderParent(j);

                if (p1 != p2) { // Merge operation
                    par[p1] = p2;
                    numberOfSets--;
                }
            }
        }
    }

    return numberOfSets;
}
```

```
// ! Important Point :

// # Initially har koi apni country ka leader tha, to jo number of sets the wo n
// # the.

// # Ab hum jaise he merge karte gaye, to same country wale ek he set mai aa
// # gaye. To jitni bar merge perform hua, utni bar kisi already existing set
// # mai koi add hua. Therefore har merge mai ek set decrease hua.

// # So total sets end mai n - total merge operation
```

```
public static boolean isSimilar(String s1, String s2) {

    // Since it is given that the two strings are always anagram of each other.
    // Isiliye agar same index pe character same nhi hai aur uska count 2 se jyada
    // hai, to hum ek swap mai to dubara string to bana he nhi payenge, to similar
    // nhi hogi

    // t a r s
    // r a t s

    // 0th and 2nd index are not similar and its count is 2 therefore are similar.

    // t a r s
    // a r t s

    // 0th, 1st, and 2nd index are not similar, count =3, so not similar

    int count = 0;
    for (int i = 0; i < s1.length(); i++) {
        if (s1.charAt(i) != s2.charAt(i) && ++count > 2)
            return false;
    }

    return true;
}
```







33. Number of Islands(Union Find)

200. Number of Islands

Medium 16967 389 Add to List Share

Given an `m x n` 2D binary grid `grid` which represents a map of `'1'`s (land) and `'0'`s (water), return *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: grid = [
  ["1","1","1","1","0"],
  ["1","1","0","1","0"],
  ["1","1","0","0","0"],
  ["0","0","0","0","0"]
]
Output: 1
```

Example 2:

```
Input: grid = [
  ["1","1","0","0","0"],
  ["1","1","0","0","0"],
  ["0","0","1","0","0"],
  ["0","0","0","1","1"]
]
Output: 3
```

Constraints:

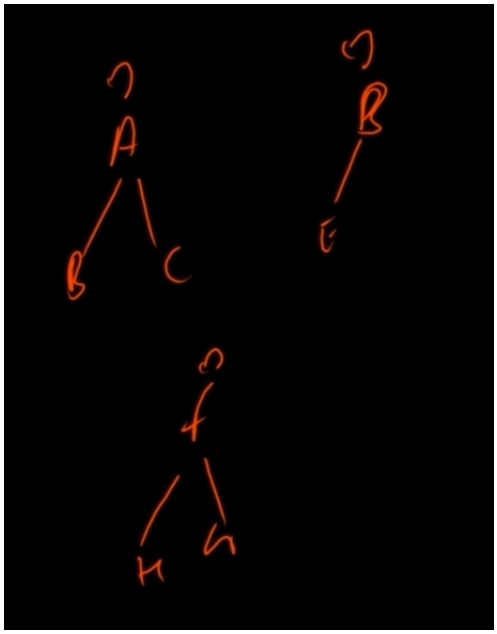
- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 300`
- `grid[i][j]` is `'0'` or `'1'`.

```
// b <==============Number of Islands ====================>
// https://leetcode.com/problems/number-of-islands/

// ? Humne yahan pe basically same 2-d to 1-d mai convert kiya.

// Jitne bhi 1 hain wo abhi independent island hain. To abhi wo khud ke he
// parent hain.

// Jaise he hum unhe group
// karte jayenge(ek he island mai merge karte jayenge), utne he single island
// kam honge

// # To end mai jitne set bachenge, utne he seperate island honge kyunki jo same
// # island ko belong karte honge unka alag se set ban gaya hoga.
```

```
public int numIslands(char[][] grid) {

    int n = grid.length;
    int m = grid[0].length;

    par = new int[n * m];

    for (int i = 0; i < n * m; i++) { // Har ek cell ko vertex man liya
        par[i] = i;
    }

    int noOfSets = 0;
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == '1') // Jitne 1 honge whi to island bana payenge. To Starting mai mana jitne 1 utne
                                   // island.
                noOfSets++;
        }
    }
```

```
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == '1') {
                int p1 = findLeaderParent(i * m + j);

                for (int d = 0; d < dir.length; d++) {
                    int r = i + dir[d][0];
                    int c = j + dir[d][1];

                    if (r >= 0 && c >= 0 && r < n && c < m && grid[r][c] == '1') {

                        int p2 = findLeaderParent(r * m + c);
                        if (p1 != p2) { // Agar p1=p2 aaya matlab wo pehle se same island ka part tha kyunki unka
                                        // parent same aaya hai. Aur p1!=p2 aaya matlab wo pehle mere island ka part
                                        // nhi tha(kyunki dono kar parent different hai), per kyunki wo mere
                                        // adjacent mai hai to mai use apne island ka part bana sakta hun. To paren
                                        // same kiya aur kyunki ab maine 2 sets ko ek mai merge kar diya hai to
                                        // number of sets to decrease to honge he by 1;
                            par[p2] = p1;
                            // Cannot write par[p1]=p2 since agar aise likha to pehle valid direction ke
                            // `baad set ka leader p2 ho jata. Agar aisa likhan hai to p1 ka duabra
```

```
                    // findLeaderParent call karna hoga update karne ke liye.

                    // par[p2] = p1 mai humeshaa global parent p1 he bana rahega.

                    // Dry run with three sets. you will understand.
                    noOfSets--;
                }
            }
        }
    }
}

return noOfSets;
```



34. Max Area of Island(Union Find)

695. Max Area of Island

Medium 7889 176 Add to List Share

You are given an `m x n` binary matrix `grid`. An island is a group of `1`'s (representing land) connected **4-directionally** (horizontal or vertical.) You may assume all four edges of the grid are surrounded by water.

The **area** of an island is the number of cells with a value `1` in the island.

Return *the maximum **area** of an island in* `grid`. If there is no island, return `0`.

Example 1:

0	0	1	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	0
0	1	0	0	1	1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0

Input: grid = [[0,0,1,0,0,0,0,1,0,0,0,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],
[0,1,1,0,1,0,0,0,0,0,0,0,0],[0,1,0,0,1,1,0,0,1,0,1,0,0],[0,1,0,0,1,1,0,0,1,1,1,0,0],
[0,0,0,0,0,0,0,0,0,0,1,0,0],[0,0,0,0,0,0,0,1,1,1,0,0,0],[0,0,0,0,0,0,0,1,1,0,0,0,0]]
Output: 6

Explanation: The answer is not 11, because the island must be connected 4-directionally.

Example 2:

```
Input: grid = [[0,0,0,0,0,0,0,0]]
Output: 0
```

Constraints:

- `m == grid.length`
- `n == grid[i].length`
- `1 <= m, n <= 50`
- `grid[i][j]` is either `0` or `1`.

```
// b <================695. Max Area of Island===============>
// https://leetcode.com/problems/max-area-of-island/

// Same as number of islands. Bas ek size ke array aur rakh diya taki size bhi
// calculate kar sakein.

// Kyunki humne initially size ko 1 rakha hai, to jaise he merge karte hain,
// leader ke size ko increase kar dete hain by the merging set size.

// To bas end mai maxSize he return karna hai
```

```
public int maxAreaOfIsland(int[][] grid) {
    int n = grid.length;
    int m = grid[0].length;

    par = new int[m * n];
    size = new int[m * n];
    for (int i = 0; i < m * n; i++) {
        par[i] = i;
        size[i] = 1;
    }

    int maxSize = 0;
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1) {
                int p1 = findLeaderParent(i * m + j);
                for (int d = 0; d < dir.length; d++) {
                    int r = i + dir[d][0];
                    int c = j + dir[d][1];

                    if (r >= 0 && c >= 0 && r < n && c < m && grid[r][c] == 1) {
                        int p2 = findLeaderParent(r * m + c);
                        if (p1 != p2) {
                            par[p2] = p1;
                            size[p1] += size[p2];
                        }
                    }
                }
                maxSize = Math.max(maxSize, size[p1]);
            }
        }
    }

    return maxSize;
}
```

35. Count Sub Islands

1905. Count Sub Islands

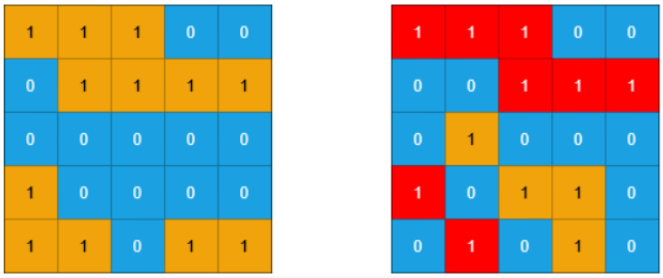
Medium 1350 42 Add to List Share

You are given two `m x n` binary matrices `grid1` and `grid2` containing only `0`'s (representing water) and `1`'s (representing land). An **island** is a group of `1`'s connected **4-directionally** (horizontal or vertical). Any cells outside of the grid are considered water cells.

An island in `grid2` is considered a **sub-island** if there is an island in `grid1` that contains **all** the cells that make up **this** island in `grid2`.

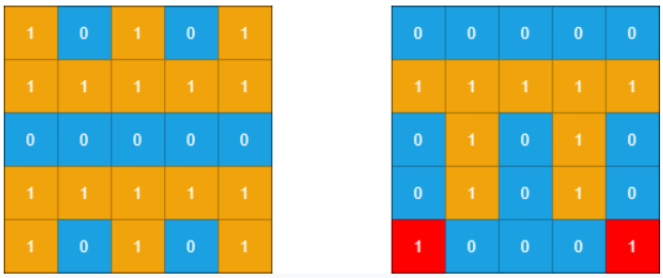
Return the ***number*** *of islands in* `grid2` *that are considered* ***sub-islands***.

Example 1:



Input: grid1 = [[1,1,1,0,0],[0,1,1,1,1],[0,0,0,0,0],[1,0,0,0,0],[1,1,0,1,1]], grid2 = [[1,1,1,0,0],[0,0,1,1,1],[0,1,0,0,0],[1,0,1,1,0],[0,1,0,1,0]]
Output: 3
Explanation: In the picture above, the grid on the left is grid1 and the grid on the right is grid2.
The 1s colored red in grid2 are those considered to be part of a sub-island. There are three sub-islands.

Example 2:



Input: grid1 = [[1,0,1,0,1],[1,1,1,1,1],[0,0,0,0,0],[1,1,1,1,1],[1,0,1,0,1]], grid2 = [[0,0,0,0,0],[1,1,1,1,1],[0,1,0,1,0],[0,1,0,1,0],[1,0,0,0,1]]
Output: 2
Explanation: In the picture above, the grid on the left is grid1 and the grid on the right is grid2.
The 1s colored red in grid2 are those considered to be part of a sub-island. There are two sub-islands.

Constraints:

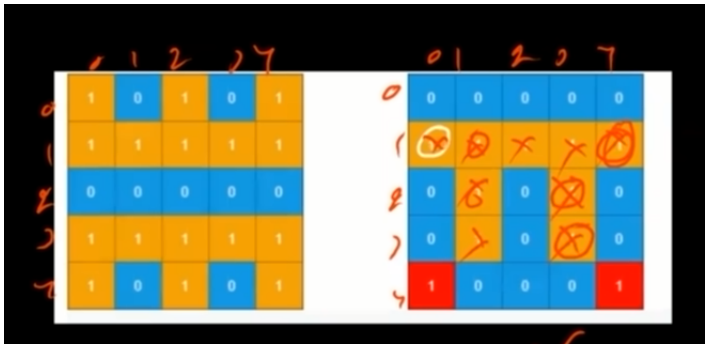
- `m == grid1.length == grid2.length`
- `n == grid1[i].length == grid2[i].length`
- `1 <= m, n <= 500`
- `grid1[i][j]` and `grid2[i][j]` are either `0` or `1`.

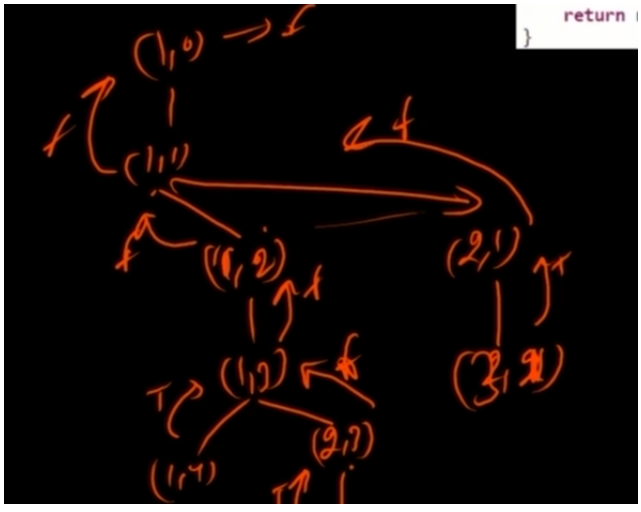
```
// b <================ 1905. Count Sub Islands ======================>
// https://leetcode.com/problems/count-sub-islands/

// Humko yahan pe basically recursion ko rokna nhi tha. Chalne dena tha taki hum
// grid 2 mai pura island ek mark karlein aur pata karlein island ka.

// ? To bas ek bar jab mark karke aa gaye, to check kiya ki mere 1 ke liye grid
// ? 1 mai koi 1 hai ki nhi. Agar grid 1 mai 0 mila to iska matlab mai jis
// ? island ko traverse kar raha hun, wo grid1 ke island ka part nhi ban sakta.
```

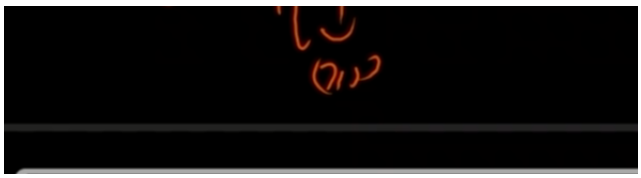
```
    // ? Ab kyunki mujhe grid2 wla island to pura mark karna he hoga, isiliye mai
    // ? recursion ko pura chalne dunga.

    // Aur agar kahin se bhi false milta hai to whi return karwaunga
```

```
    public int countSubIslands(int[][] grid1, int[][] grid2) {

        int n = grid2.length, m = grid2[0].length;
        int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

        int noOfSubIslands = 0;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid2[i][j] == 1) {
                    boolean res = dfs_countSubIslands(grid1, grid2, dir, i, j);
                    if (res)
                        noOfSubIslands++;
                }
            }
        }

        return noOfSubIslands;
    }
}
```

```
    public static boolean dfs_countSubIslands(int[][] grid1, int[][] grid2, int[][] dir, int sr, int sc) {
        grid2[sr][sc] = 2; // Mark kiya

        boolean res = true;
        for (int d = 0; d < dir.length; d++) {
            int r = sr + dir[d][0];
            int c = sc + dir[d][1];

            if (r >= 0 && c >= 0 && r < grid2.length && c < grid2[0].length && grid2[r][c] == 1) {
                res = dfs_countSubIslands(grid1, grid2, dir, r, c) && res; // Agar ek direction se false aaya aur agle
                                                                           // direction se true aay, to hume to fales he
                                                                           // return karwana hai, per recursion ko rokna
                                                                           // nhi hai
            }
        }

        return res && grid1[sr][sc] == 1;
    }
```







36. Number of Islands II

305. Number of Islands II

Hard 1132 31 Add to List Share

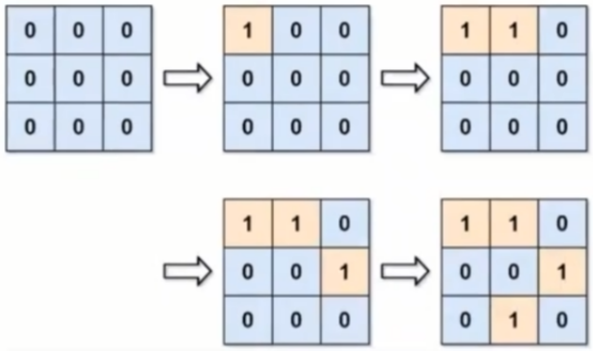
You are given an empty 2D binary grid `grid` of size `m x n`. The grid represents a map where `0`'s represent water and `1`'s represent land. Initially, all the cells of `grid` are water cells (i.e., all the cells are `0`'s).

We may perform an add land operation which turns the water at position into a land. You are given an array `positions` where `positions[i] = [r_i, c_i]` is the position `(r_i, c_i)` at which we should operate the `i^th` operation.

Return *an array of integers* `answer` *where* `answer[i]` *is the number of islands after turning the cell* `(r_i, c_i)` *into a land.*

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: m = 3, n = 3, positions = [[0,0],[0,1],[1,2],[2,1]]
Output: [1,1,2,3]
Explanation:
Initially, the 2d grid is filled with water.
- Operation #1: addLand(0, 0) turns the water at grid[0][0] into a land. We have 1
island.
- Operation #2: addLand(0, 1) turns the water at grid[0][1] into a land. We still have
1 island.
- Operation #3: addLand(1, 2) turns the water at grid[1][2] into a land. We have 2
islands.
- Operation #4: addLand(2, 1) turns the water at grid[2][1] into a land. We have 3
islands.
```

Example 2:

```
Input: m = 1, n = 1, positions = [[0,0]]
Output: [1]
```

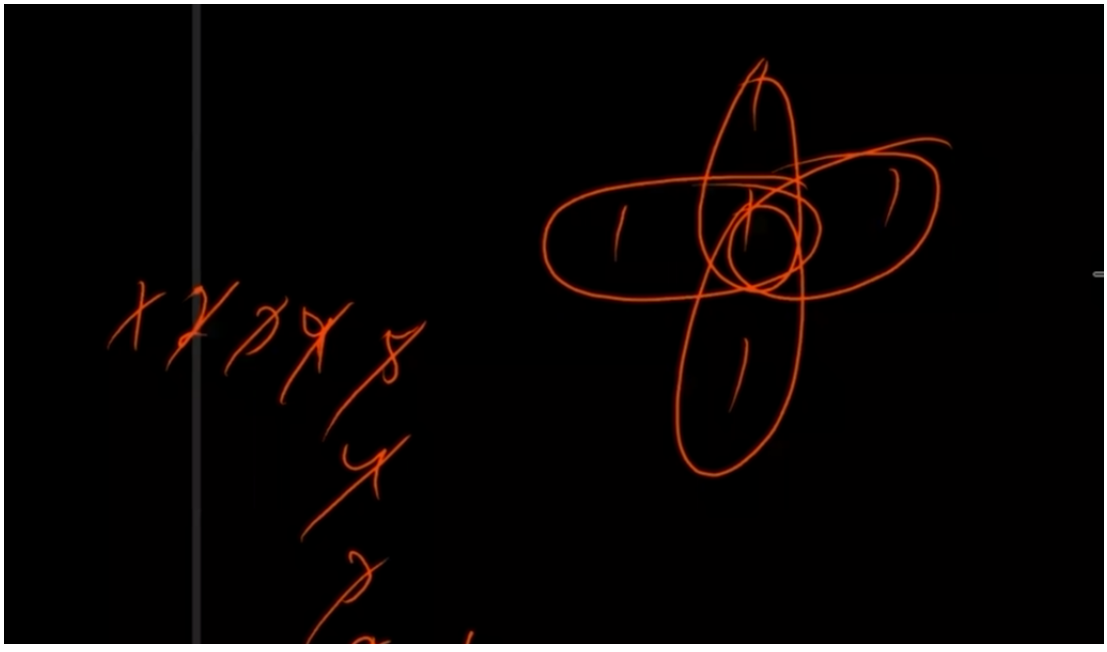
Constraints:

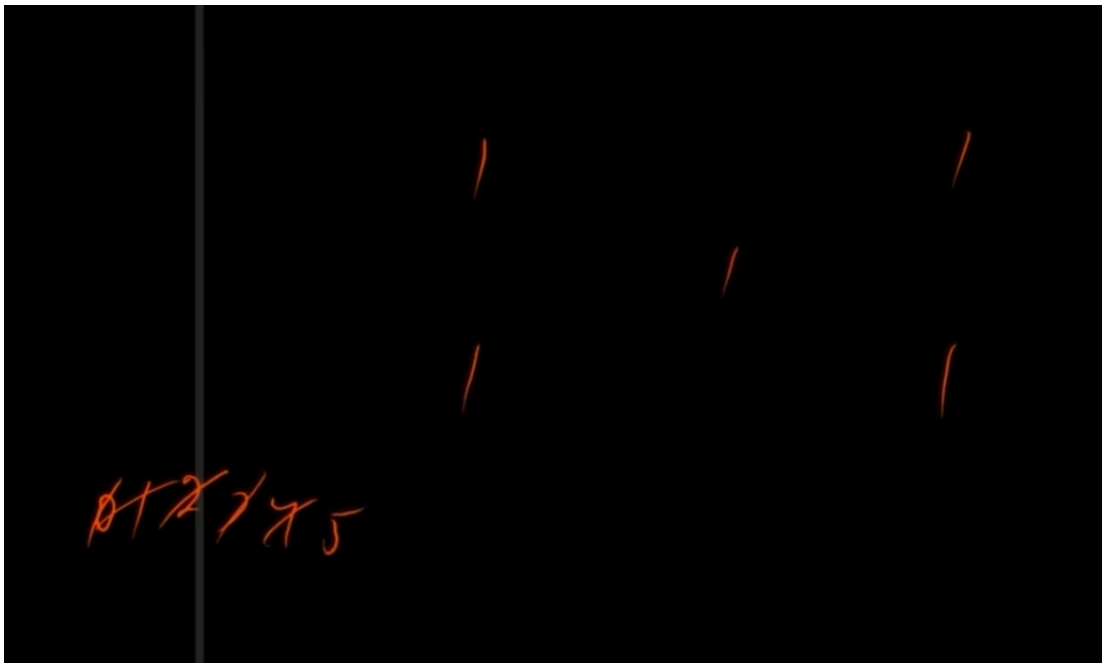
- `1 <= m, n, positions.length <= 10^4`
- `1 <= m * n <= 10^4`
- `positions[i].length == 2`
- `0 <= r_i < m`
- `0 <= c_i < n`

Follow up: Could you solve it in time complexity `O(k log(mn))`, where `k == positions.length`?

```
// b <================== Number of Islands II ==================>
// https://www.lintcode.com/problem/434/

// ? Yahan pe kiya kya ??

// Jaise he hum koi edge nikal rahe hain(means cell jo land mai convert hogi),
// hun count mai +1 kar rahe hain indicating ki ek nya island add hua hai. Ab
// hum us island ke surrounding mai dekehnge, agar wo kisi aur island ka part
// ` ban sakta hai ki nhi. Agar ban payega to use merge karke count mai -1 kar
// denge.

// Jaise he hum edge nikalte jayenge, waise he hum apni grid mai update bhi
// karte jayenge. Taki hume pata rahe ki 1 kahan kahan pe hai.

// Ek edge case dhyan rakhna, agar koi already land hai aur use dubara land mai
// convert karna hai, to koi bhi changes nhi honge.
```

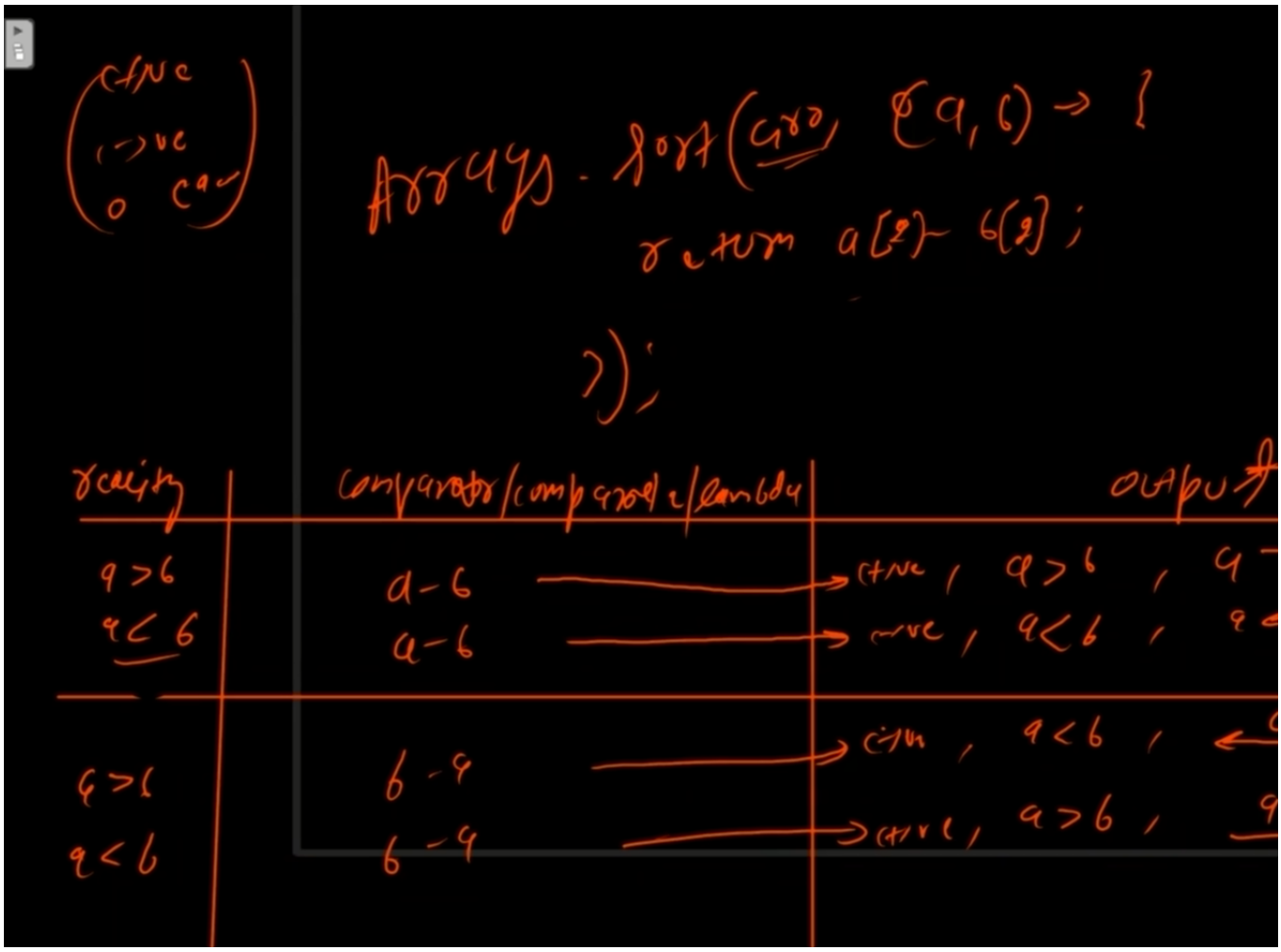
```
// # This is using grid
public List<Integer> numIslands2(int n, int m, int[][] operators) {

    List<Integer> ans = new ArrayList<>();
    int[][] grid = new int[n][m];

    par = new int[n * m];
    for (int i = 0; i < n * m; i++) {
        par[i] = i;
    }

    int count = 0;
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    for (int[] p : operators) {
        int i = p[0], j = p[1];

        if (grid[i][j] == 1) { // if there is already a land where we are going then nothing should be updated.
            ans.add(count);
            continue;
        }
        grid[i][j] = 1;

        count++;
        int p1 = findLeaderParent(i * m + j);
        for (int d = 0; d < dir.length; d++) {
            int r = i + dir[d][0];
            int c = j + dir[d][1];

            if (r >= 0 && c >= 0 && r < n && c < m && grid[r][c] == 1) {
                int p2 = findLeaderParent(r * m + c);

                if (p1 != p2) {
                    par[p2] = p1;
                    count--;
                }
            }
        }
        ans.add(count);
    }
    return ans;
}
```





```java
// # Without using the grid space.

// kyunki ab sare check parent ke array se lagene honge, to initially parent
// array ko -1 se fill kiya
public List<Integer> numIslands2_(int n, int m, int[][] operators) {

    List<Integer> ans = new ArrayList<>();

    par = new int[n * m];
    for (int i = 0; i < n * m; i++) {
        par[i] = -1;
    }

    int count = 0;
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    for (int[] p : operators) {
        int i = p[0], j = p[1];

        if (par[i * m + j] != -1) { // Agar parent array pehle se he visited ha us point mai, matlab wahan pe pehle
                                    // se he land hai
            ans.add(count);
            continue;
        }

        par[i * m + j] = i * m + j;

        count++;
        int p1 = i * m + j; // Jo aaya humne use he apna global parent bana diya agar uske aa pass 1 hai to
        for (int d = 0; d < dir.length; d++) {
            int r = i + dir[d][0];
            int c = j + dir[d][1];

            if (r >= 0 && c >= 0 && r < n && c < m && par[r * m + c] != -1) {
                int p2 = findLeaderParent(r * m + c);
                if (p1 != p2) {
                    par[p2] = p1;
                    count--;
                }
            }
        }
        ans.add(count);
    }

    return ans;
}
```

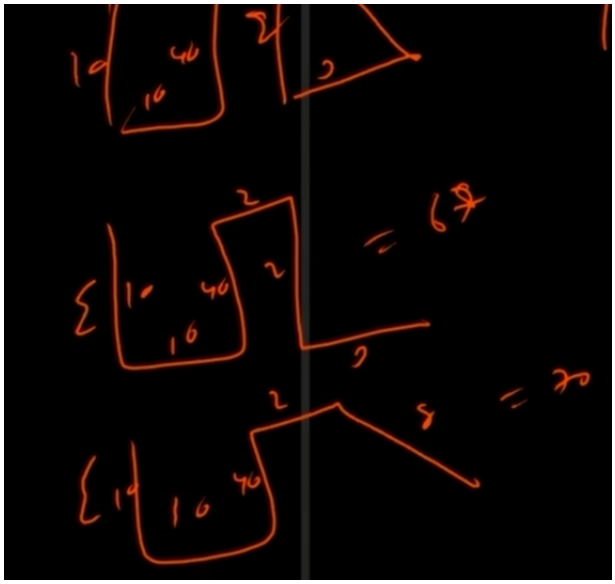
37. Sort an Array (this - other concept)

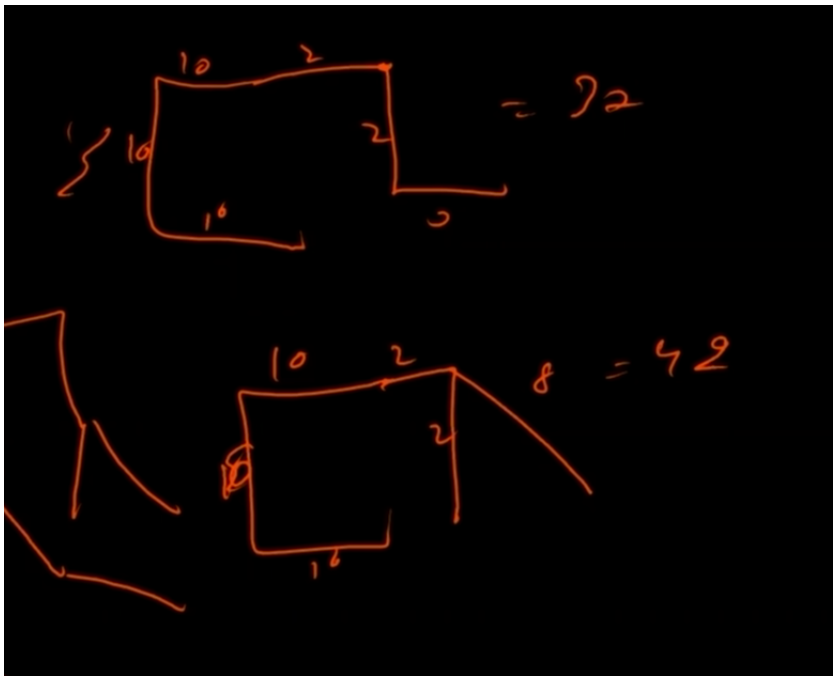



```java
public static void sortArray(int[][] arr) {
    // {{A,B,C}}, sort on the basis of index 2
    Arrays.sort(arr, (a, b) -> {
        // return a[2] - b[2]; // this- other,default beahviour
        return b[2] - a[2]; // other - this,reverse of default beahviour
    });

    display2D(arr);
}

public static void main(String[] args) {
    int[][] arr = { { 1, 2, 3 }, { 1, 2, 2 }, { 4, 2, -3 }, { 10, 22, 23 } };
    sortArray(arr);
}
```

38. Minimum Spanning Tree

```java
// B <==============MST(Minimum Spanning Tree) ================>

// Aisa spanning tree, jiska edges ki weight ka sum mimimum ho, use bolte hain
// minumum spanning tree.

// Ek graph ke to bahut sare spanning tree ban sakte hain, to jo miminum wala
// hoga with edges weight is called MST.

// ! Condition ek ye hain ki graph connected he rehna chahiye.
```

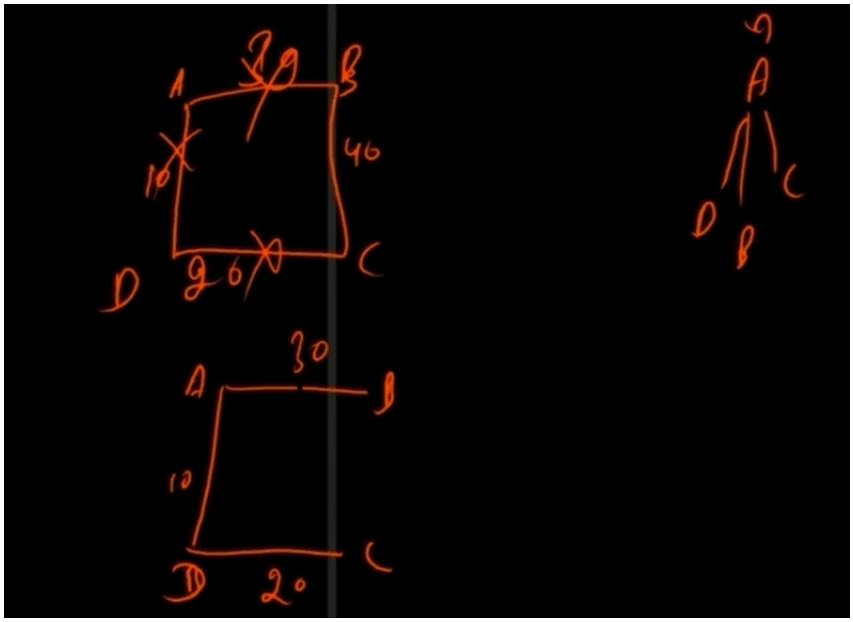




So 37 is the one with minimum weight, so therefore called MST

```
// B <==============MST(Minimum Spanning Tree) ================>

// # Graph connected with no cycle is called spanning tree.
// # Ab agar spanning tree mai edges ke weight ka sum minimum hua to MST

// ! Remember the number of components remain same in MST.
// The components in the original graph and the components after the union find
// algo

// Aisa spanning tree, jiska edges ki weight ka sum mimimum ho, use bolte hain
// minumum spanning tree.

// Ek graph ke to bahut sare spanning tree ban sakte hain, to jo miminum wala
// hoga with edges weight is called MST.

// ! Condition ek ye hain ki graph connected he rehna chahiye.

/*
 * Ab Mera union find algo is baat ki guarentee to deta he hai ki spanning tree
 * banega. Aur agar mai edges ko sorted by weight pass karun, to wo minimum
 * spanning tree dega. Per aisa kyun ??????
 */

// ? Aisa isiliye kyunki jab bhi mai union find algo lagata hun aur mai, tab mai
// ? jo edges cycle ke liye resposible hoti hai, mai use include he nhi karta
// ? hun. Ab agar maine weight wise sorted kiya, to jo sari heavy edges hai wo
// ? baad mai aayengei aur kyunki wo cycle ke liye responsible ho sakti hain to
```

```
// ? hum unhe include he nhi karenge. Isi reason ki wajah se MST Create Ho jata
// ? hai edeges ko sorted rakhne se.
```



39. Kruskal Algorithm

```
// b <============== Kruskal Algo ==================>

// It finds the minimum spanning tree.
// How ? Same by passing the sorted edges array for the reason mentioned above.

public static void kruskalAlgo(int[][] edges, int N) {
    // {{u,v,w}}
    Arrays.sort(edges, (a, b) -> {
        return a[2] - b[2];
    });
}
```

Kruskal's algorithm[1] finds a minimum spanning forest of an undirected edge-weighted graph. If the graph is connected, it finds a minimum spanning tree. (A minimum spanning tree of a connected graph is a subset of the edges that forms a tree that includes every vertex, where the sum of the weights of all the edges in the tree is minimized. For a disconnected graph, a minimum spanning forest is composed of a minimum spanning tree for each connected component.) It is a greedy algorithm in graph theory as in each step it adds the next lowest-weight edge that will not form a cycle to the minimum spanning forest.[2]

This is the kruskal algo to find the mst(minimum spanning tree)

40. Optimize Water Distribution in a Village

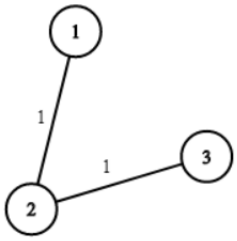
1168. Optimize Water Distribution in a Village

There are `n` houses in a village. We want to supply water for all the houses by building wells and laying pipes.

For each house `i`, we can either build a well inside it directly with cost `wells[i]`, or pipe in water from another well to it. The costs to lay pipes between houses are given by the array `pipes`, where each `pipes[i] = [house1, house2, cost]` represents the cost to connect `house1` and `house2` together using a pipe. Connections are bidirectional.

Find the minimum total cost to supply water to all houses.

Example 1:




```
Input: n = 3, wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]
Output: 3
Explanation:
The image shows the costs of connecting houses using pipes.
The best strategy is to build a well in the first house with cost 1 and connect the other
```

```
connect the other houses to it with cost 2 so the total cost is 3.
```

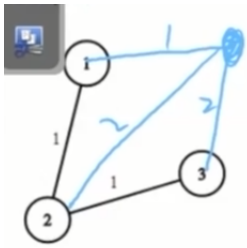
Constraints:

- `1 <= n <= 10000`
- `wells.length == n`
- `0 <= wells[i] <= 10^5`
- `1 <= pipes.length <= 10000`
- `1 <= pipes[i][0], pipes[i][1] <= n`
- `0 <= pipes[i][2] <= 10^5`
- `pipes[i][0] != pipes[i][1]`

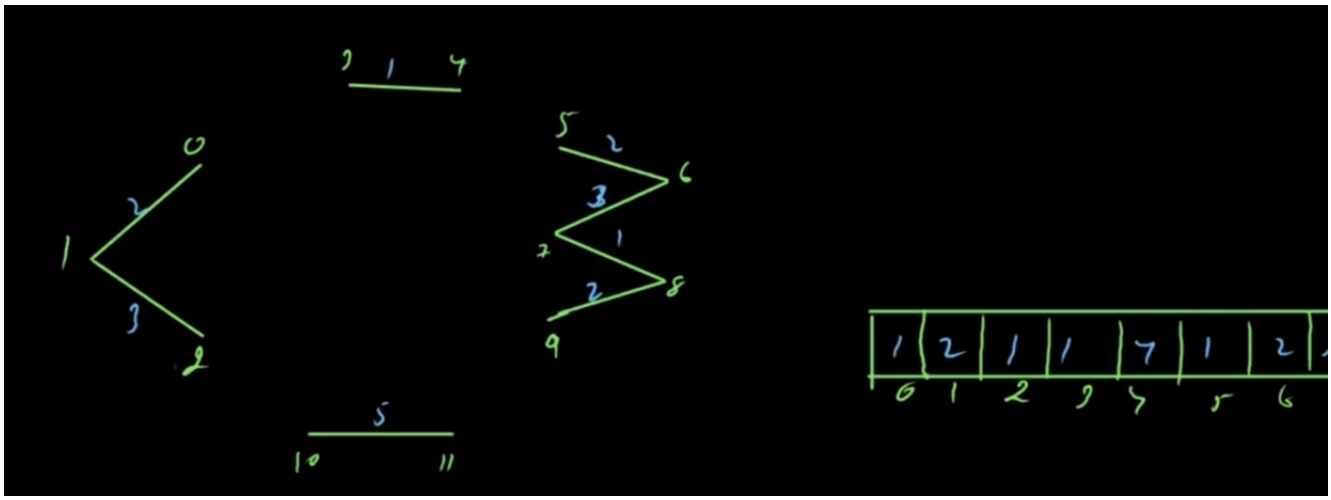
```
// b <========= 1168. Optimize Water Distribution in a Village==========>
// https://leetcode.ca/all/1168.html
// https://www.pepcoding.com/resources/data-structures-and-algorithms-in-java-levelup/g

// ! To yahan pe kiya kya ???

// Yahan pe hume basically min cost nikalni hai water ko har ek ghar tak
// pahunchane ke liye.

// Humpe do option hain ya to well bana lete hain ya pipe ka use karke kisi aur
// ke well se pani le lenge

// To humko dikh raha hai ki hume min cost nikalni hai. To hum yahan pe mst wla
// concept use kar sakte hain. Per mera graph to connected he nhi hai.

// To iske liye humne kiya ye ki ek aur vertex manli(a bigger well that will
// supply to other wells in the village)

// Ab humne sare edges jo us bigger well se sare houses ko ja sakte the, humne
// `unko bhi add kar diya. Humne ye mana ki bigger well se house tak pipe se
// transfer karne ki cost hai wells ke array mai.

// ? To ab ye ek pipes ki problem mai convert ho gyi aur humara graph connected
// ? ho gya. Ab agar hum isme mst lagayenge to hume humara answer mil jayega.
// ? Bas jab hum merge operation karenge, tab he us edge (pipe connection) ko
// ? add karne ki cost bhi add karte jayenge aur end mai whi return kar denge
```

```
public static int minCostToSupplyWater(int n, int[] wells, int[][] pipes) {

    ArrayList<int[]> edges = new ArrayList<>();
    for (int[] pipe : pipes) {
        edges.add(pipe);
    }

    for (int i = 0; i < wells.length; i++) {
```

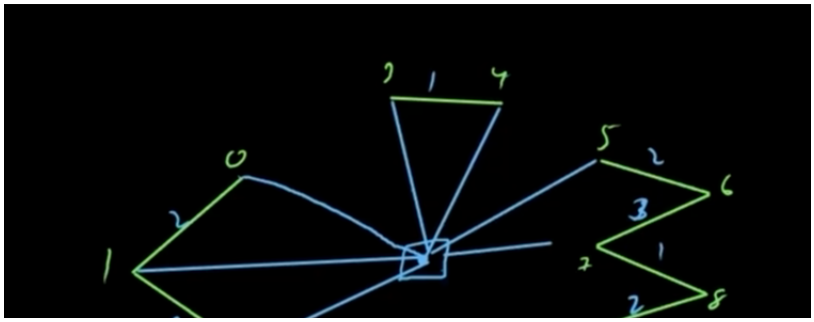
```
            edges.add(new int[] { 0, i + 1, wells[i] }); // Added the new pipe connection to all the houses from the
                                                         // ` bigger well.
        }

        edges.sort((a, b) -> { // Sorted the array for MST
            return a[2] - b[2];
        });

        // ! Baki pura union find algo hai
        par = new int[n + 1];
        for (int i = 0; i < n + 1; i++) {
            par[i] = i;
        }
        int minCost = 0;
        for (int[] edge : edges) {

            int u = edge[0], v = edge[1], w = edge[2];
            int p1 = findLeaderParent(u);
            int p2 = findLeaderParent(v);

            if (p1 != p2) {
                par[p1] = p2;
                minCost += w;
            }
        }

        return minCost;
    }
```



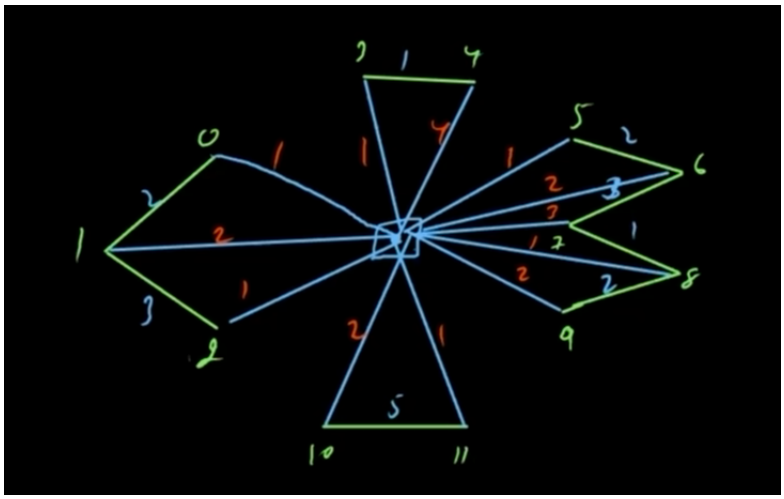
Input: n = 3, wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]
Output: 3
Explanation:
The image shows the costs of connecting houses using pipes.
The best strategy is to build a well in the first house with cost 1 and connect the other houses to it with cost 2 so the total cost is 3.



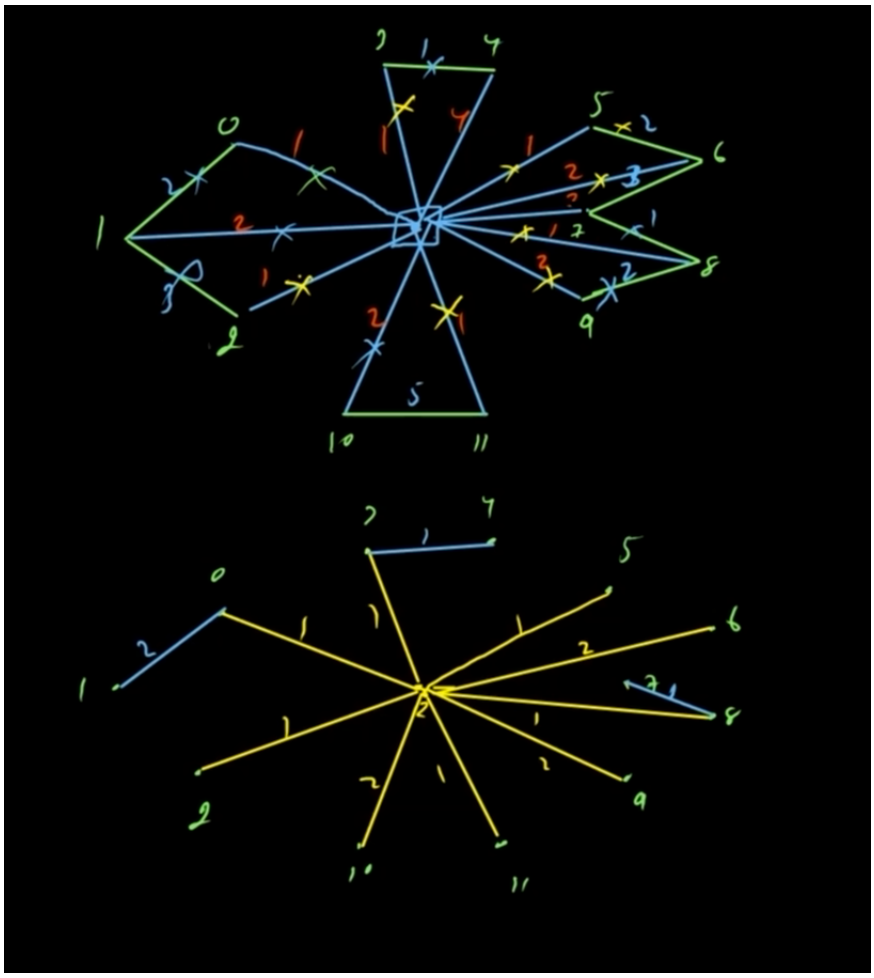
Pehle graph connected nhi tha



Ab connected hai to MST laga sakte hain



Bigger well at the center



Bottom one is the MST

41. Mr President

Problem

You have recently started playing a brand new computer game called "Mr. President". The game is about ruling a country, building infrastructures and developing it.

Your country consists of **N** cities and **M** bidirectional roads connecting them. Each road has assigned a cost of its maintenance. The greatest achievement in the game is called "Great administrator" and it is given to a player who manage to have all cities in the country connected by roads in such a way that it is possible to travel between any two cities and that the sum of maintenance costs of these roads is not greater than **K**.

This is very hard to accomplish, but you are very close to do it. More precisely, you have just discovered a new method of transforming standard roads into super roads, with cost of maintenance just **1**, due to their extreme durability.

The bad news is that it is very expensive to transform a standard road into a super road, but you are so excited that you are going to do it anyway.

In addition, because you have a lot of other expenses, you also want to first demolish as many roads as possible in order to safe some money on their maintenance first and then start working on getting the achievement. You can demolish any road in the country and that operation does not cost you anything.

Because you want to spend the absolutely minimum money in order to get the achievement, you are interested in the smallest number of transformations of standard roads into super roads in such a way that you can do that.

Input format:

In the first line there are 3 integers **N**, **M** and **K** denoting the number of cities in the country, the number of roads in it and the desired sum of costs of maintenance. M lines describing these roads follow. In each of them there are 3 integers **A**, **B** and **C**, where **A** and **B** denote the endpoints of the road while **C** denotes the cost of its maintenance.

Output:

In a single line, output the minimum number of roads which need to be transformed in order to get the achievement. If you cannot do it no matter what, output -1.

Constraints:

$2 <= N, M <= 10^6$
$0 <= K <= 10^{18}$
$1 <= A, B <= N$ and $A != B$
$1 <= C <= 10^6$

Sample Input	Sample Output
3 3 25 1 2 10 2 3 20 3 1 30	1

Time Limit: 2
Memory Limit: 256
Source Limit:

Explanation

You can transform to super a road either the road between cities 1 and 2 or the road between cities 2 and 3 in order to produce the desired road network of costs respectively 21 and 11. Doing that will cost you one transformation and it is optimal in this case.

Contributers:

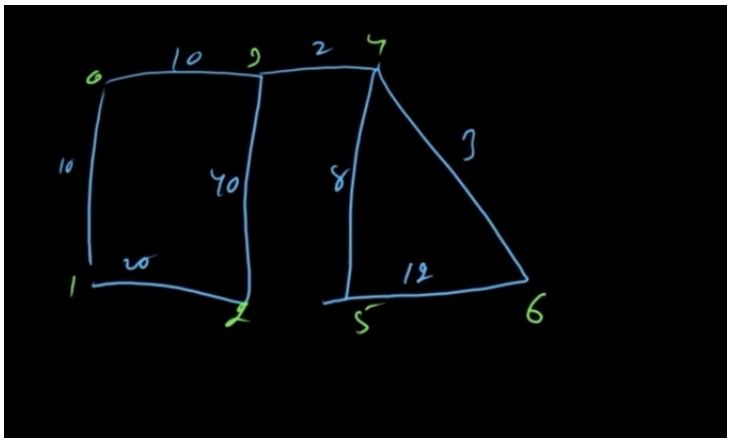

Pawe? Kacprzak

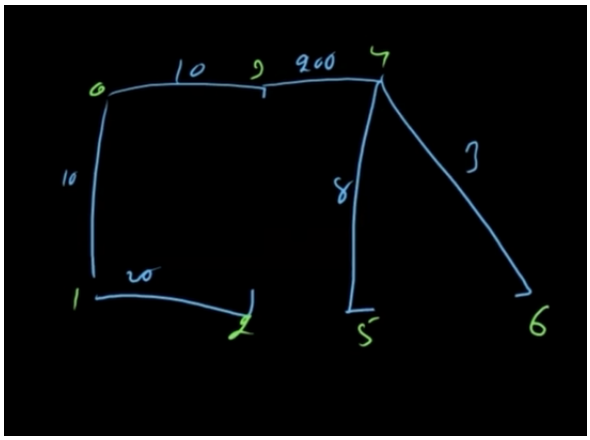

Bohdan Pryshchenko

```java
// b <============== Mr. President ================>
// https://www.hackerearth.com/practice/algorithms/graphs/minimum-spanning-tree/prac

// ! So humne kiya kya ?
// Humko ek graph de rakha tha cities ke beech mai. Wahan road bhi de rakh thi.
// Consider them like edges.

// To humne pehle ek spanning tree create kiya taki hume sari roads jinhe hum
// demolish kar sakte the unhe nikal dein. Ab hume K se kam karni hai cost of
// maintainence. To ek sorted arraylist humne create ki while creating the
// spanning tree. Ab agar K se kam value lani hai with minimum conversion, to
// pehle sabse badi value ko decrease karna chahiye. to bas whi kiya
```

```java
public static int mr_president() {

    // N = > Cities
    // M => bidirectional Roads
    // K => Maintainence cost of the road should be less than k (Achievement)
    // can convert a road to a super road( that has maintainence 1)
    // you can demolish roads
    // In order to get the achievement, find the minimum number of road
    // transformation of standard road to super road

    Scanner scn = new Scanner(System.in);
    int N = scn.nextInt();
    int M = scn.nextInt();
    long k = scn.nextLong();

    ArrayList<int[]> Edges = new ArrayList<>();
    for (int i = 0; i < M; i++) {
        int u = scn.nextInt(), v = scn.nextInt(), w = scn.nextInt();
        Edges.add(new int[] { u, v, w });
    }

    Collections.sort(Edges, (a, b) -> {
        return a[2] - b[2];
    });

    par = new int[N + 1];

    for (int i = 0; i < N + 1; i++) {
        par[i] = i;
    }
```

```java
    int numberOfComponents = N;
    long totalMaintainenceCost = 0;
    ArrayList<Integer> costArray = new ArrayList<>();
    for (int[] edge : Edges) {
        int u = edge[0], v = edge[1], w = edge[2];

        int p1 = findLeaderParent(u);
        int p2 = findLeaderParent(v);

        if (p1 != p2) {
            par[p1] = p2;
            numberOfComponents--;
            totalMaintainenceCost += w;
            costArray.add(w);
        }
    }

    if (numberOfComponents > 1) { // Agar graph mai he number of compoennts jyada hai to sari cities connected reh
                                  // he nhi payegi
        return -1;
    }
    int numberOfTransformation = 0;
    for (int i = costArray.size() - 1; i >= 0; i--) {
        if (totalMaintainenceCost > k) {
            totalMaintainenceCost = totalMaintainenceCost - costArray.get(i) + 1; // +1 for maintainence of the new
                                                                                  // super road.
            numberOfTransformation++;
        } else {
            break;
        }
    }

    return totalMaintainenceCost > k ? -1 : numberOfTransformation;
}
```





Manle ki spanning tree ki values ye aayi. To aur k ki value 30 hai. Dry run kar.

Why we need to store the weight of the cost of the created spanning tree?

```
// Why we need to store the weight of the cost of the created spanning tree?

// Let sau that the spannig trees values are 10,10,10,20,20,20,20.
// And the value of k is 30.

// Now dry run and figure out.
```

42. Min Cost to connect all the points

1584. Min Cost to Connect All Points

Medium 2952 78 Add to List Share

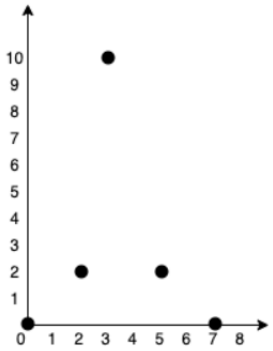
You are given an array `points` representing integer coordinates of some points on a 2D-plane, where `points[i] = [xi, yi]`.

The cost of connecting two points `[xi, yi]` and `[xj, yj]` is the **manhattan distance** between them: `|xi - xj| + |yi - yj|`, where `|val|` denotes the absolute value of `val`.

Return *the minimum cost to make all points connected.* All points are connected if there is **exactly one** simple path

between any two points.

Example 1:




```
Input: points = [[0,0],[2,2],[3,10],[5,2],[7,0]]
Output: 20
Explanation:
```


```
We can connect the points as shown above to get the minimum cost of 20.
Notice that there is a unique path between every pair of points.
```

Example 2:

```
Input: points = [[3,12],[-2,5],[-4,1]]
Output: 18
```

Constraints:

- `1 <= points.length <= 1000`
- `-10^6 <= x_i, y_i <= 10^6`
- All pairs `(x_i, y_i)` are distinct.

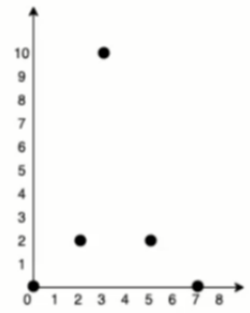
```java
// ! Remember a spanning tree has N-1 number of edges

// b <================ 1584. Min Cost to Connect All Points ============>
// https://leetcode.com/problems/min-cost-to-connect-all-points/

// So sabse pehle apna connected graph create kiya. Ab bas simple MST ka logic
// laga diya.

// ? Here mapped every point to index. So this makes it very easy so to keep
// ? track of the point.
```

```java
public int minCostConnectPoints(int[][] points) {
    int n = points.length;

    ArrayList<int[]> Edges = new ArrayList<>();
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
```

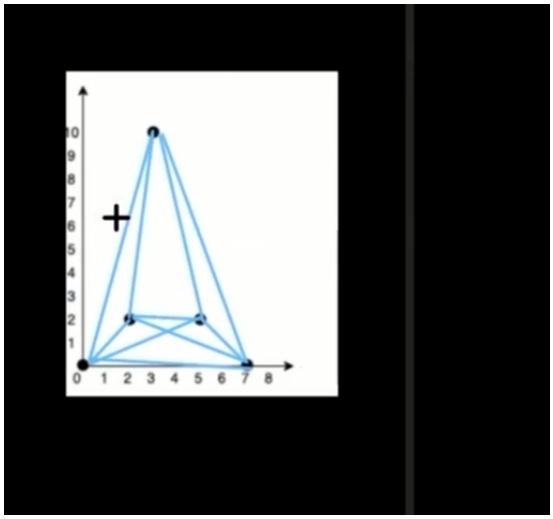
```
            int u = i, v = j, w = Math.abs(points[i][0] - points[j][0]) + Math.abs(points[i][1] - points[j][1]);
            // w is the distance between the two points. So i.e weight of the edge.
            Edges.add(new int[] { u, v, w }); // Sare points ko ek dusre se connect karke ek dense graph create
                                              // kiya.
        }
    }

    par = new int[n];
    for (int i = 0; i < n; i++) {
        par[i] = i;
    }
    Collections.sort(Edges, (a, b) -> {
        return a[2] - b[2];
    });

    int minCost = 0;
    for (int[] edge : Edges) {
        int u = edge[0], v = edge[1], w = edge[2];

        int p1 = findLeaderParent(u);
        int p2 = findLeaderParent(v);

        if (p1 != p2) {
            par[p2] = p1;
            minCost += w;
        }
    }

    return minCost;
}
```

Return *the minimum cost to make all points connected.* All points are connected if there is **exactly one** simple path between any two points.

Example 1:

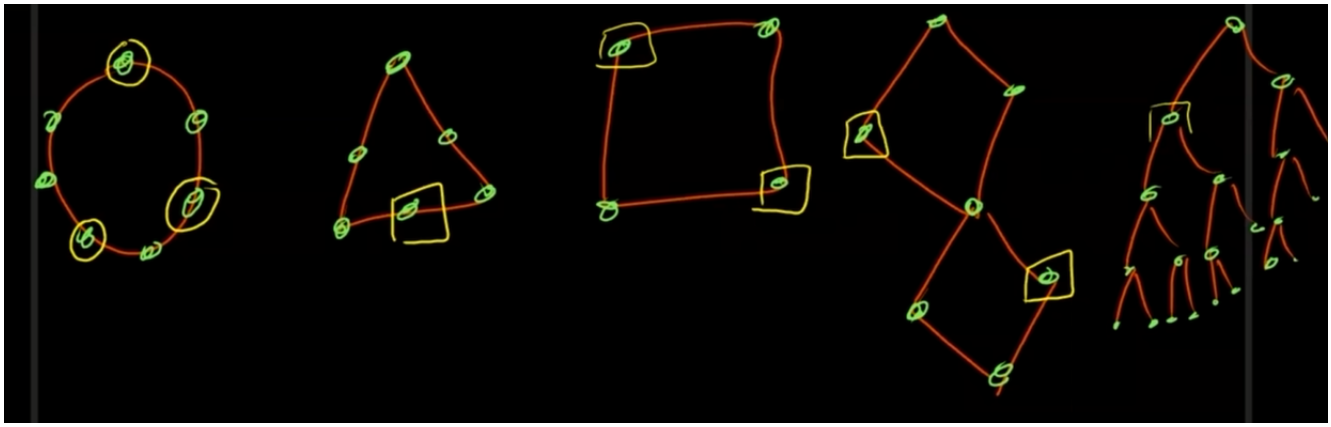



```
Input: points = [[0,0],[2,2],[3,10],[5,2],[7,0]]
Output: 20
Explanation:
```



The densely connected graph will look like this.

43. Minimize Malware Spread



```
// b <============== 924. Minimize Malware Spread ====================>
// https://leetcode.com/problems/minimize-malware-spread/

// As already discussed, Union find wahan pe ache se kaam karta hai jahan pe
// grouping hoti hai

// Ab is question ko aise soch ki bahut sari country hai aur har country mai
// kafi bahut se log reh rahe hain.

// Aur har country mai ek ya ek se jyada log initially infect ho sakta hain. To
// main kis country ko save karun ki mai max logon ki jaan bacha sakun

// Main unko save karna prefer karunga jinme sirf ek he infected hai, taki mai
// ` use remove karke puri country ko save kar paun.

// Aur agar aise bahut sari country hain jisme sirf ek infected person hai, to
// ` unme se jiski population jyada hogi mai usi ko to save karunga kyunki whin
// pe max logon ki jaan bach payegi.
```

```
// ? Aap sirf ek he node ko cure kar sakte ho
```

```
static int[] population; // Yahan pe har country ka size store kar rahe hai

public int minMalwareSpread(int[][] graph, int[] initial) {
    // M(initial) == > final nodes that got infected by the malwaare
    // We have to minimize M(initial)
    // Only we can remove one initial infected node.
    // return the node on which removal M(initial) is minimized.
    // In question given the matrix representation of the graph

    int n = graph.length, m = graph[0].length;

    par = new int[n];
    population = new int[n];

    for (int i = 0; i < n; i++) {
        par[i] = i;
        population[i] = 1;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (i == j || graph[i][j] == 0)
                continue;

            int p1 = findLeaderParent(i);
            int p2 = findLeaderParent(j);

            if (p1 != p2) { // Yahan pe same country ke logon ko ek sath karke unka ek similar leader bana
                            // diya
                par[p2] = p1;
                population[p1] += population[p2];
            }
        }
    }
```

```
        // Here have found the infected count of every country.
        int[] infectedCount = new int[n];
        for (int i = 0; i < initial.length; i++) {
            int parent = findLeaderParent(initial[i]);
            infectedCount[parent]++;
        }

        Arrays.sort(initial); // Agar merepe aisa case aaya ki sari country ka size same hai aur sabke
                              // infected log bhi same hain, to mai usko return karunga jiska index small ho.

        int ans = initial[0];
        int maxPopulation = 0;
        for (int i = 0; i < initial.length; i++) {
            int ele = initial[i];
            int parent = findLeaderParent(ele);
            if (infectedCount[parent] == 1 && population[parent] > maxPopulation) { // Yahan pe compare kiya ki agar
                                                                                    // ` bahut sari country hai jinme
                                                                                    // sirf ek he infected hai, to
                                                                                    // mai us country ko bachaunga
                                                                                    // jiski population jyada hogi
                ans = ele;
                maxPopulation = population[parent];
            }
        }

        return ans;
    }
```

44. Region Cut by Slashes

959. Regions Cut By Slashes

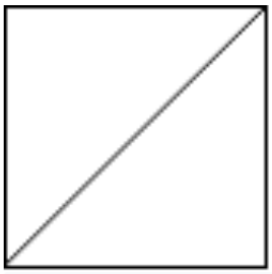
Medium 2431 465 Add to List Share

An `n x n` grid is composed of `1 x 1` squares where each `1 x 1` square consists of a `'/'`, `'\'`, or blank space `' '`. These characters divide the square into contiguous regions.

Given the grid `grid` represented as a string array, return *the number of regions*.

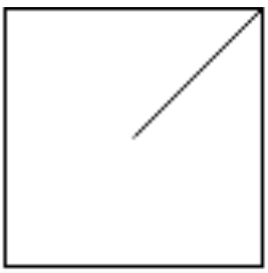
Note that backslash characters are escaped, so a `'\'` is represented as `'\\'`.

Example 1:



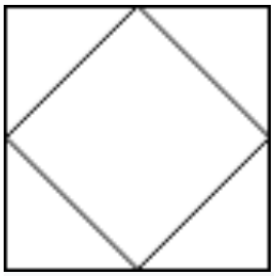
```
Input: grid = [" /","/ "]
Output: 2
```

Example 2:



```
Input: grid = [" /","  "]
Output: 1
```

Example 3:



```
Input: grid = ["/\\","\\/"]
Output: 5
Explanation: Recall that because \ characters are escaped, "\\/" refers to \/, and "/\\" refers to /\.
```

Constraints:

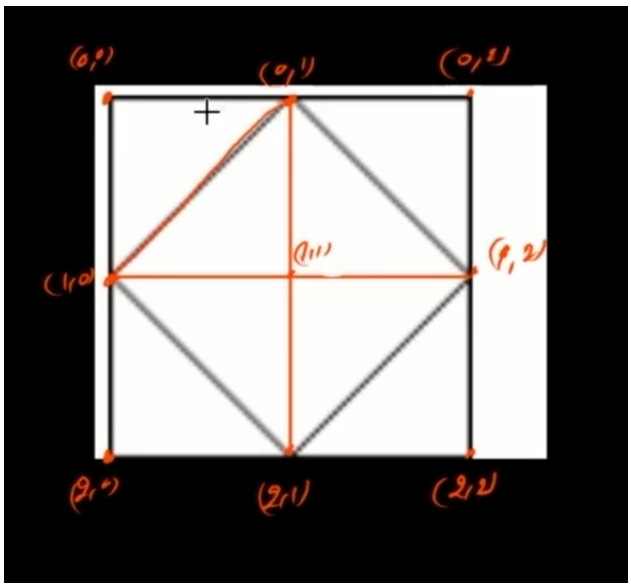
- `n == grid.length == grid[i].length`
- `1 <= n <= 30`
- `grid[i][j]` is either `'/'`, `'\'`, or `' '`.

```
// b <============= 959. Regions Cut By Slashes ================>
// https://leetcode.com/problems/regions-cut-by-slashes/

// # Bfs and DSU can find number of distinct cycles, not the overlapping one.

// ! If I get two points, then only I can create a line .
// ! So Basically I would be needing two points.
// ! Ek line banane ke liye do points to chahiye he honge.

// # So Agar tu number of distincts cycles nikal de, to tereko wo number of
// # regions mil jayenge jo slashes ne cut kiya hoga

// ? Per ab isko karein kaise ????

// Kuch aise grid de rakhi hai grid = ["/\\","\\/"]
// Now consider this grid as a 2-d matrix. With string as an array stroring some
// values in 0th and 1st index.

// Ab forward slash and backward slash kuch region cut kar rahe hain.
// let n=grid.length
// ` Unko show karne ke liye ek +1 size ki matrix ki jaroorat hogi

// Let N=n+1
// To humne N*N ki matrix consider ki aur usme points mark kiye.

// Agar grid mai (0,0) mai / hai to wo (x,y+1) and (x+1,y) points pe draw hoga
// Agar grid mai (0,1) mai \\ hai to wo (x,y) and (x+1,y+1) points pe draw hoga

// Ab bas same union find algo lagaya aur grid se ek ek edge nikalte rahe.
// Aur number of cycles count kar li.

// # Initially cycle ka count 1 hoga kyunki boundary bhi to ek cycle bana rahi
// # hogi
```

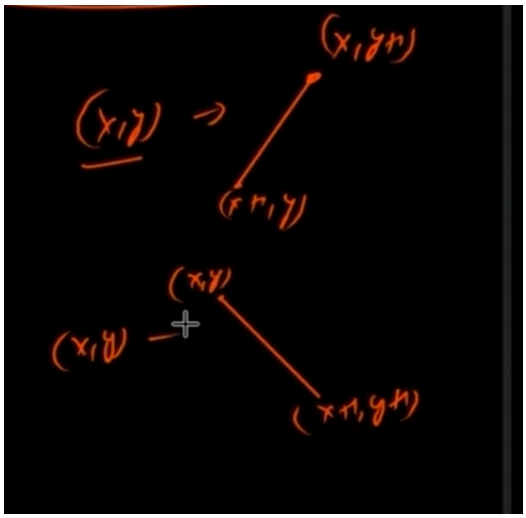
```
// Ab kyunki boundary ek cycle hai to aur sare edges ek dusre se connected hain
// to unka parent ek he hoga initially.

public static int mergeUnion(int p1, int p2) {
    if (p1 != p2) {
        par[p2] = p1;
        return 0;
    }
    return 1; // Agar cycle hui to 1 return karega
}

public int regionsBySlashes(String[] grid) {
    int n = grid.length;
    int N = n + 1;

    par = new int[N * N];
    for (int i = 0; i < N * N; i++) {
```

```
        if (i / N == 0 || i % N == 0 || i / N == N - 1 || i % N == N - 1) { // Sare boundary elements ka parent 0 ko
                                                                            // maan liya kyunki wo ek sath connected
                                                                            // hain
            par[i] = 0;
        } else {
            par[i] = i;
        }
    }
```

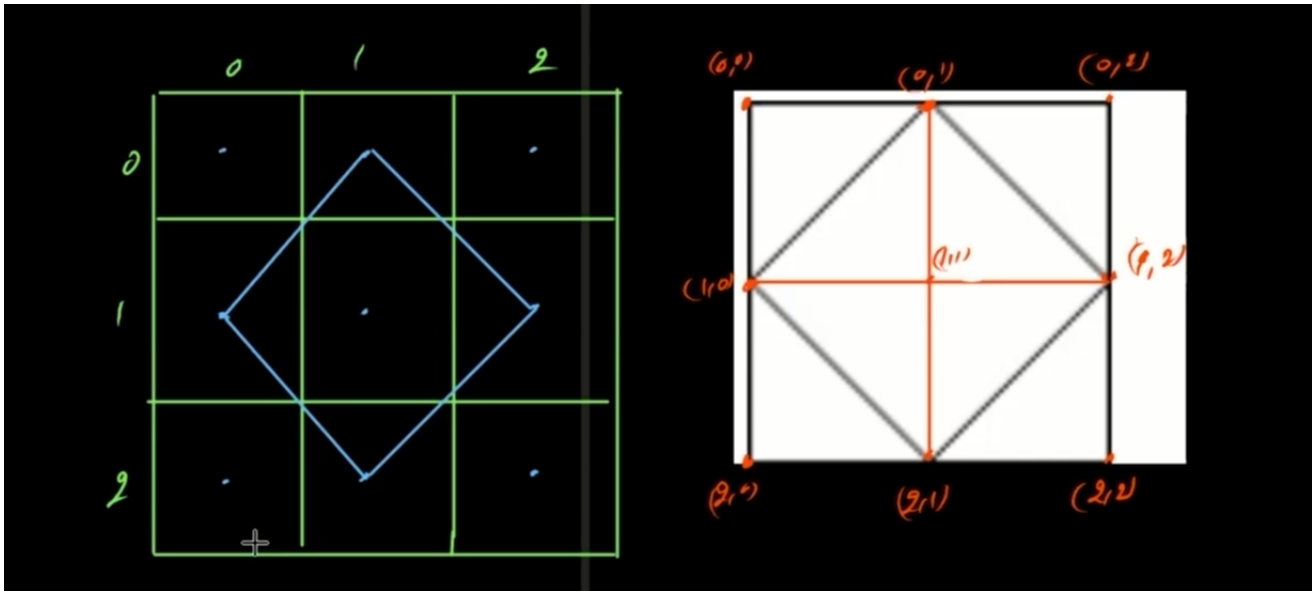
```
public int regionsBySlashes(String[] grid) {
    int n = grid.length;
    int N = n + 1;

    par = new int[N * N];
    for (int i = 0; i < N * N; i++) {
        if (i / N == 0 || i % N == 0 || i / N == N - 1 || i % N == N - 1) { // Sare boundary elements ka parent 0 ko
                                                                            // maan liya kyunki wo ek sath connected
                                                                            // hain
            par[i] = 0;
        } else {
            par[i] = i;
        }
    }

    int numberOfCycles = 1; // 1 se isiliye start kiya kyunki ek cycle to already hai jo boundary hai
    for (int i = 0; i < n; i++) {
        String str = grid[i];
        for (int j = 0; j < str.length(); j++) {
            char ch = str.charAt(j);
            if (ch == '/') {
                // Dono points nikale ki kahan pe line draw hogi aur tab merge kara
                int p1 = findLeaderParent(i * N + j + 1);
                int p2 = findLeaderParent((i + 1) * N + j);
                numberOfCycles += mergeUnion(p1, p2);
            } else if (ch == '\\') {
                int p1 = findLeaderParent(i * N + j);
                int p2 = findLeaderParent((i + 1) * N + j + 1);
                numberOfCycles += mergeUnion(p1, p2);
            }
        }
    }

    return numberOfCycles;
}
```

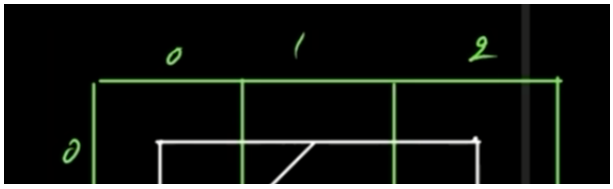


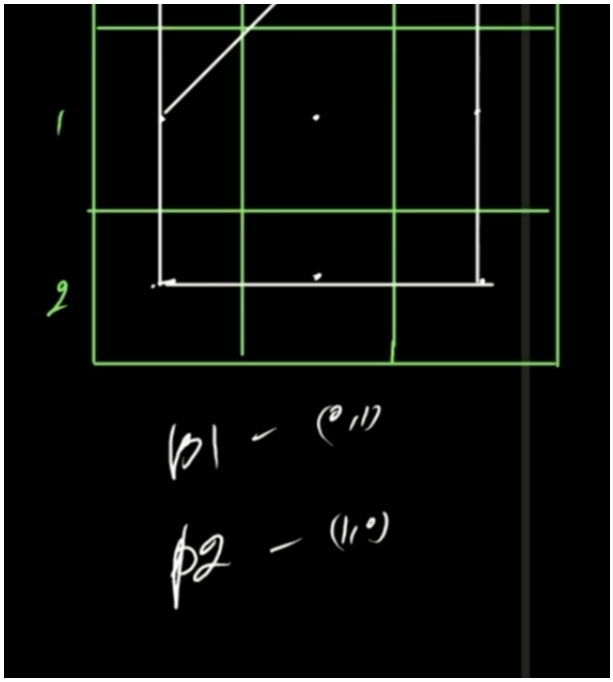

(1,0)
(1,1)
(1,2)
(2,0)
(2,1)
(2,2)

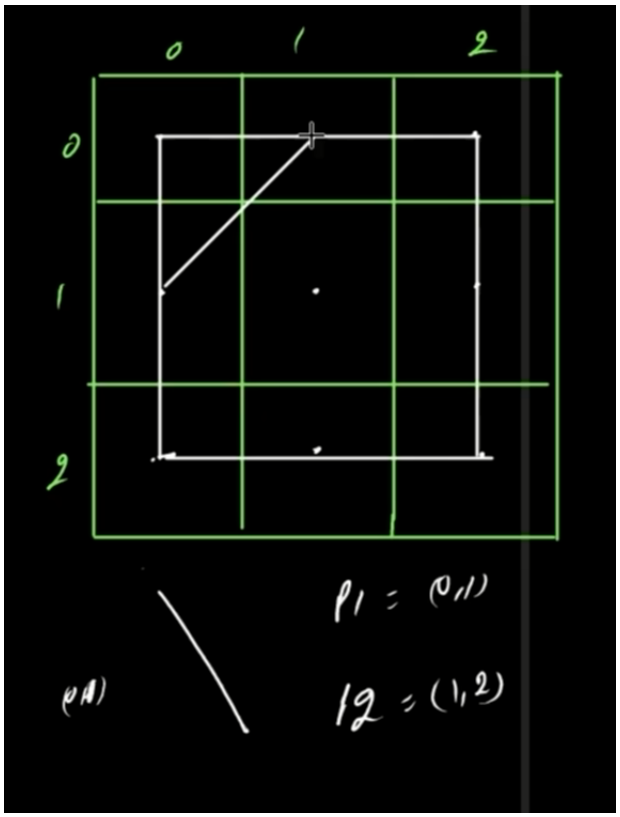

(x,y)
(x+1,y)
(x,y)
(x,y+1)
(x,y)
(x+1,y+1)

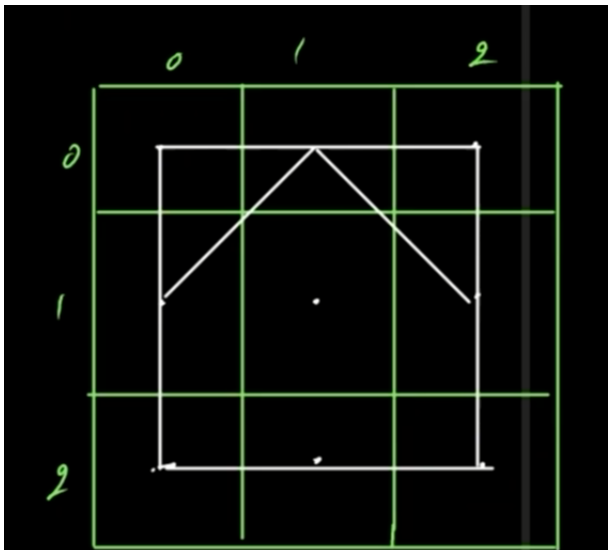
```
grid = ["/\\","\\/"]
```

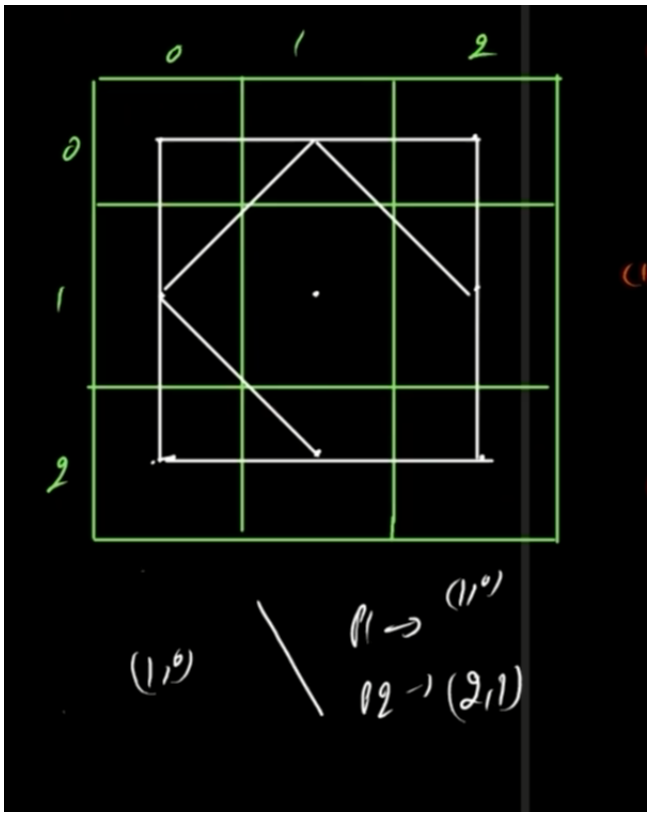

0
1
2
0
1
2
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
(2,0)
(2,1)
(2,2)

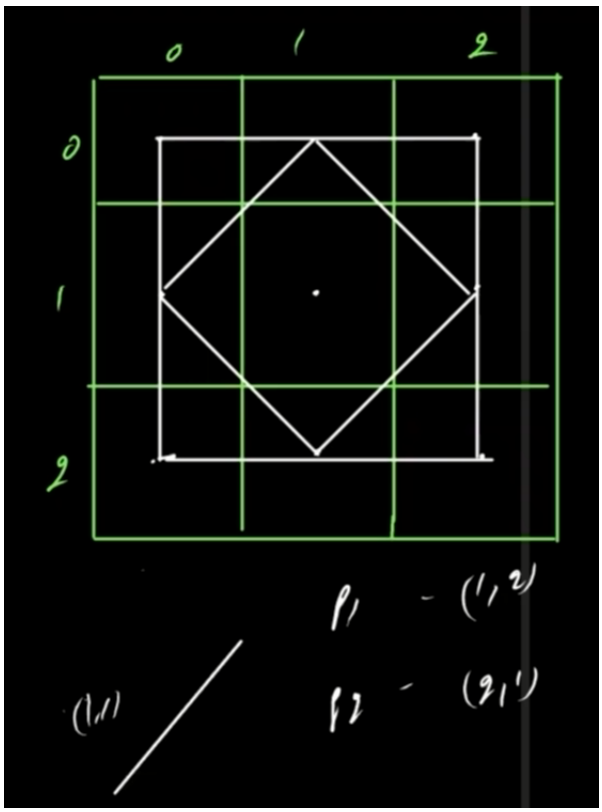

0
1
2
0

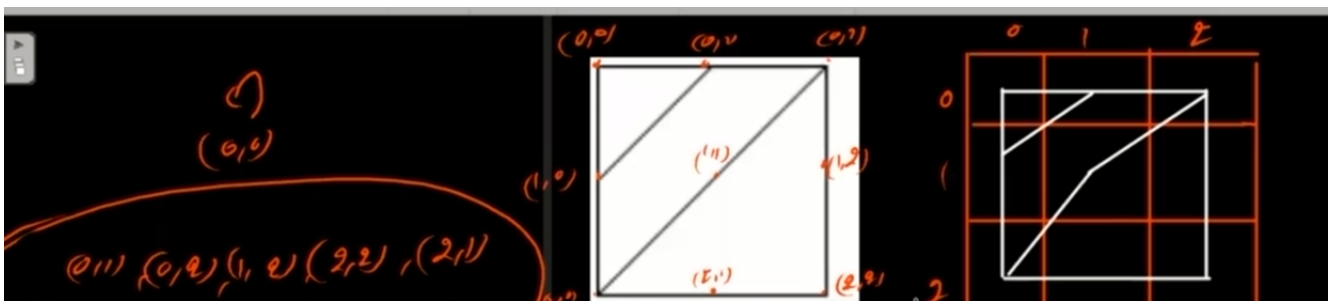

1
2
p1 - (0,1)
p2 - (1,0)

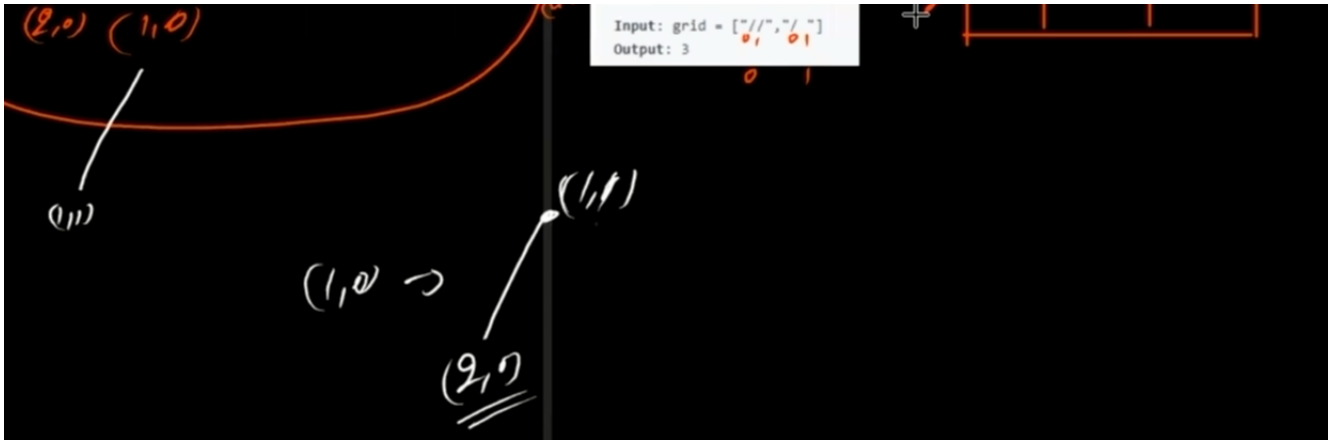

0
1
2
0
1
2
p1 = (0,1)
p2 = (1,2)
(0,1)


0
1
2
0
1
2


0
1
2
0
1
2
(1,0)
(1,1)
(2,1)


0
1
2
0
1
2
(1,1)


(0,0)
(1,0)
(2,0)
(0,1)
(1,1)
(2,1)
(0,2)
(1,2)
(2,2)
0
1
2



45. Satisfiability of Equality Equations

990. Satisfiability of Equality Equations

Medium 1834 23 Add to List Share

You are given an array of strings `equations` that represent relationships between variables where each string `equations[i]` is of length `4` and takes one of two different forms: `"xi==yi"` or `"xi!=yi"`.Here, `xi` and `yi` are lowercase letters (not necessarily different) that represent one-letter variable names.

Return `true` *if it is possible to assign integers to variable names so as to satisfy all the given equations, or* `false` *otherwise.*

Example 1:

```
Input: equations = ["a==b","b!=a"]
Output: false
Explanation: If we assign say, a = 1 and b = 1, then the first equation is satisfied, but not the second.
There is no way to assign the variables to satisfy both equations.
```

Example 2:

```
Input: equations = ["b==a","a==b"]
Output: true
Explanation: We could assign a = 1 and b = 1 to satisfy both equations.
```

Constraints:

- `1 <= equations.length <= 500`
- `equations[i].length == 4`
- `equations[i][0]` is a lowercase letter.
- `equations[i][1]` is either `'='` or `'!'`.
- `equations[i][2]` is `'='`.
- `equations[i][3]` is a lowercase letter.

```java
// b <============= 990. Satisfiability·of·Equality·Equations ============>
// https://leetcode.com/problems/satisfiability-of-equality-equations/

public boolean equationsPossible(String[] equations) {

    par = new int[26];
    for (int i = 0; i < 26; i++) {
        par[i] = i;
    }

    int[] notArray = new int[26];
```

```
        for (String str : equations) { // Pehle jo bhi equal hai un sabko same set ke andar dala
            if (str.charAt(index: 1) == '=') {
                int p1 = findLeaderParent(str.charAt(index: 0) - 'a');
                int p2 = findLeaderParent(str.charAt(index: 3) - 'a');

                if (p1 != p2) {
                    par[p2] = p1;
                }
            }
        }
```

```
        for (String str : equations) {// Ab agar maine jo sabhi equal unko ek set mai rakh diya hai to agar ab mujhe
                                      // koi bhi milega jiske parent same honge(reason because they have == between
                                      // them), aur umse ! ka sign aaya, to wo equation false ho jayegi.
            if (str.charAt(index: 1) == '!') {
                int p1 = findLeaderParent(str.charAt(index: 0) - 'a');
                int p2 = findLeaderParent(str.charAt(index: 3) - 'a');

                if (p1 == p2) {
                    return false;
                }
            }
        }
        return true;
    }
```

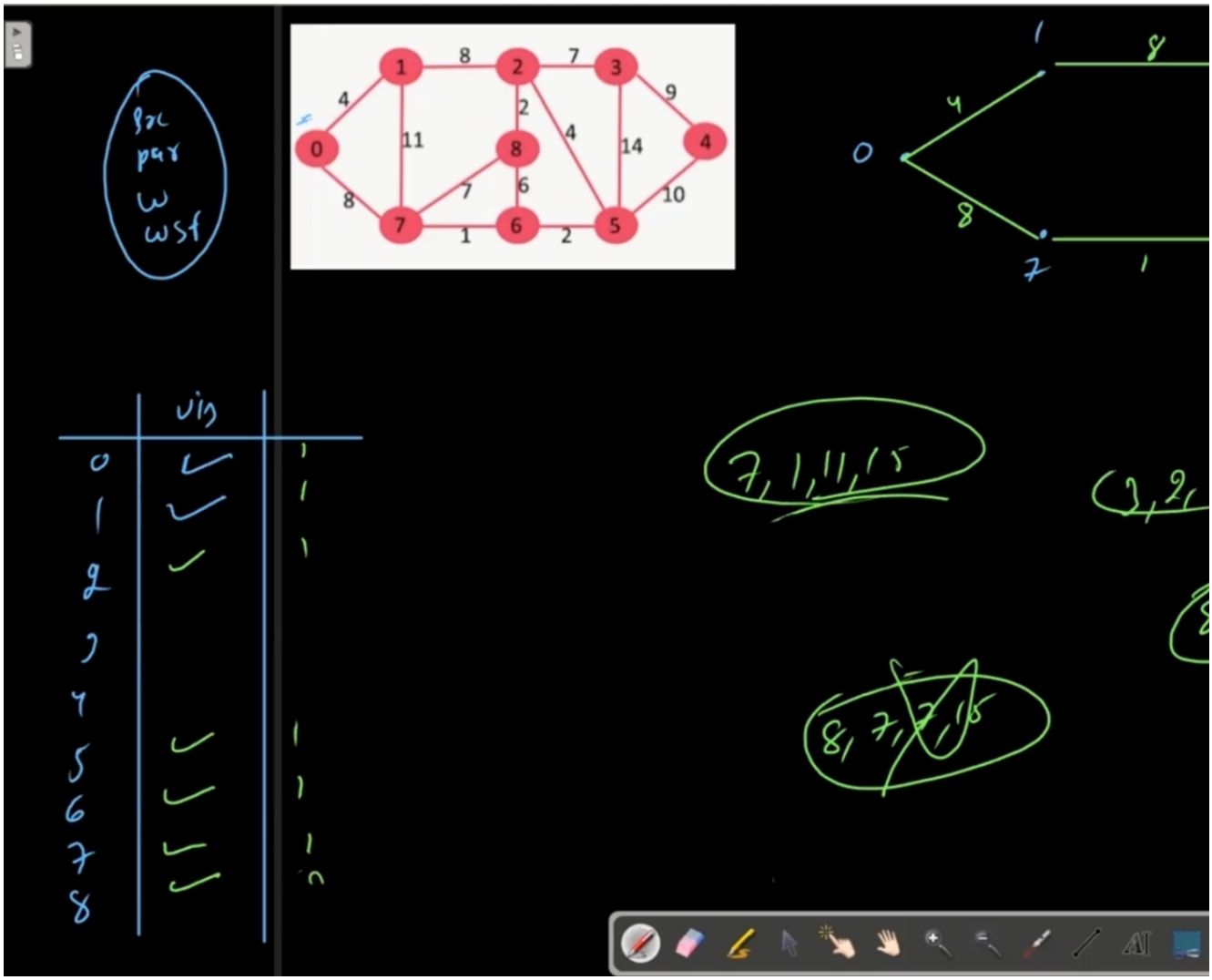
46. Dijikstra

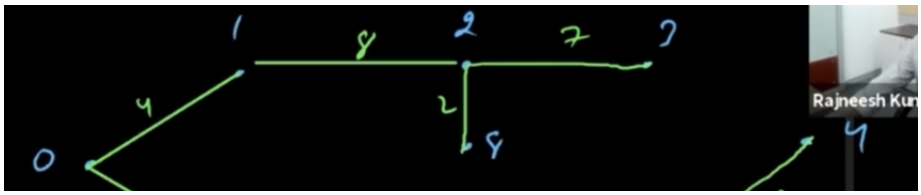
```
    // b <<<<============== Dijikstra(Normal One) ======================>>>>

    // MST pure graph ke weight ko minimise karne ki try karta hai. Ye(MST) sabhi
    // edges ke sum ko minimise karne ki try karta hai

    // ! It(Dijikstra) is a single source algorithm.
    // # So it gives the lightest(minimum) weight path from source to every vertex.

    // It does not work for -ve edges.

    // Simple hai wsf(weight so far) ke terms mai priority queue se bahar nikalte
    // rehna hai simple bfs mai. Bas isse hume ek specific source se kisi vertex tak
    // ka shhortest path(in terms of weight) mil jata hai. Preferred to say the
    // lightest path
```

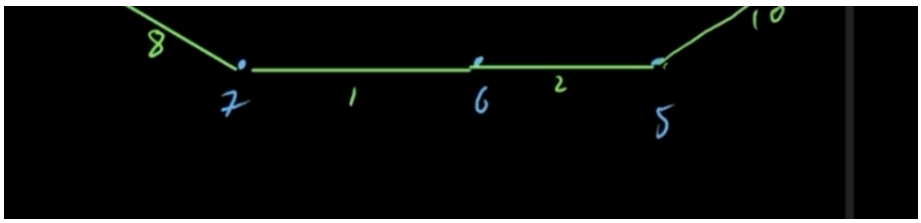
```
    public static void dijikstra(ArrayList<Edge>[] graph, int V, int src) {
        ArrayList<Edge>[] mygraph = new ArrayList[V];
        for (int i = 0; i < V; i++)
            graph[i] = new ArrayList<>();

        boolean[] vis = new boolean[V];
        PriorityQueue<pair> pq = new PriorityQueue<>((a, b) -> {
            return a.wsf - b.wsf; // This - other concept
        });

        pq.add(new pair(src, -1, w: 0, wsf: 0));
        while (pq.size() != 0) {
            pair p = pq.remove();
            if (vis[p.src])
                continue;

            if (p.par != -1) // To create the new(The lightest path from the source to every vertex)
                             // graph(generally not needed).
                addEdge(mygraph, p.src, p.par, p.w);

            vis[p.src] = true;
            for (Edge e : graph[p.src]) {
                if (!vis[e.v])
                    pq.add(new pair(e.v, p.src, e.w, p.wsf + e.w));
            }
        }
    }
```

```
    public static class Edge {
        int v, w;
```

```
    Edge(int v, int w) {
        this.v = v;
        this.w = w;
    }
}

public static void addEdge(ArrayList<Edge>[] graph, int u, int v, int w) {
    graph[u].add(new Edge(v, w));
    graph[v].add(new Edge(u, w));
}

public static class pair {
    int src = 0;
    int par = 0;
    int w = 0;
    int wsf = 0;

    // dijikstra
    pair(int src, int par, int w, int wsf) {
        this.src = src;
        this.par = par;
        this.w = w;
        this.wsf = wsf;
    }

    // This one to be used in prims algo
    pair(int src, int par, int w) {
        this(src, par, w, wsf: 0); // constructor chaining
    }
```







47. Network Delay Time

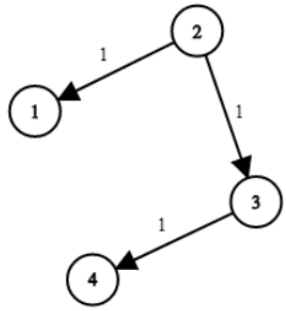
743. Network Delay Time

Medium 5591 318 Add to List Share

You are given a network of `n` nodes, labeled from `1` to `n`. You are also given `times`, a list of travel times as directed edges `times[i] = (ui, vi, wi)`, where `ui` is the source node, `vi` is the target node, and `wi` is the time it takes for a signal to travel from source to target.

We will send a signal from a given node `k`. Return *the* ***minimum*** *time it takes for all the* `n` *nodes to receive the signal*. If it is impossible for all the `n` nodes to receive the signal, return `-1`.

Example 1:




```
Input: times = [[2,1,1],[2,3,1],[3,4,1]], n = 4, k = 2
Output: 2
```

Example 2:

```
Input: times = [[1,2,1]], n = 2, k = 1
Output: 1
```

Example 3:

```
Input: times = [[1,2,1]], n = 2, k = 2
Output: -1
```

Constraints:

- `1 <= k <= n <= 100`
- `1 <= times.length <= 6000`
- `times[i].length == 3`
- `1 <= ui, vi <= n`
- `ui != vi`
- `0 <= wi <= 100`
- All the pairs `(ui, vi)` are **unique**. (i.e., no multiple edges.)

```
// b <============= 743. Network Delay Time ==================>
// https://leetcode.com/problems/network-delay-time/

// # Here created edge and the pair of both integer array. Good for online test.
// # But create a class in front of interviewer.
```

```java
    // ! Also try to break it in function.

    public int networkDelayTime(int[][] times, int n, int k) {

        // Edge : {v,w}
        ArrayList<int[]>[] graph = new ArrayList[n + 1];

        for (int i = 0; i < n + 1; i++)
            graph[i] = new ArrayList<>();

        for (int[] time : times) {
            // times[i] = (ui, vi, wi)
            graph[time[0]].add(new int[] { time[1], time[2] }); // Apna graph create kiya
        }

        // ! Baki simple Dijistra lagaya. niche ka pura hum ek function mai likh sakte
        // ! hain.

        // pair : {v,wsf}
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> {
            return a[1] - b[1];
        });
```

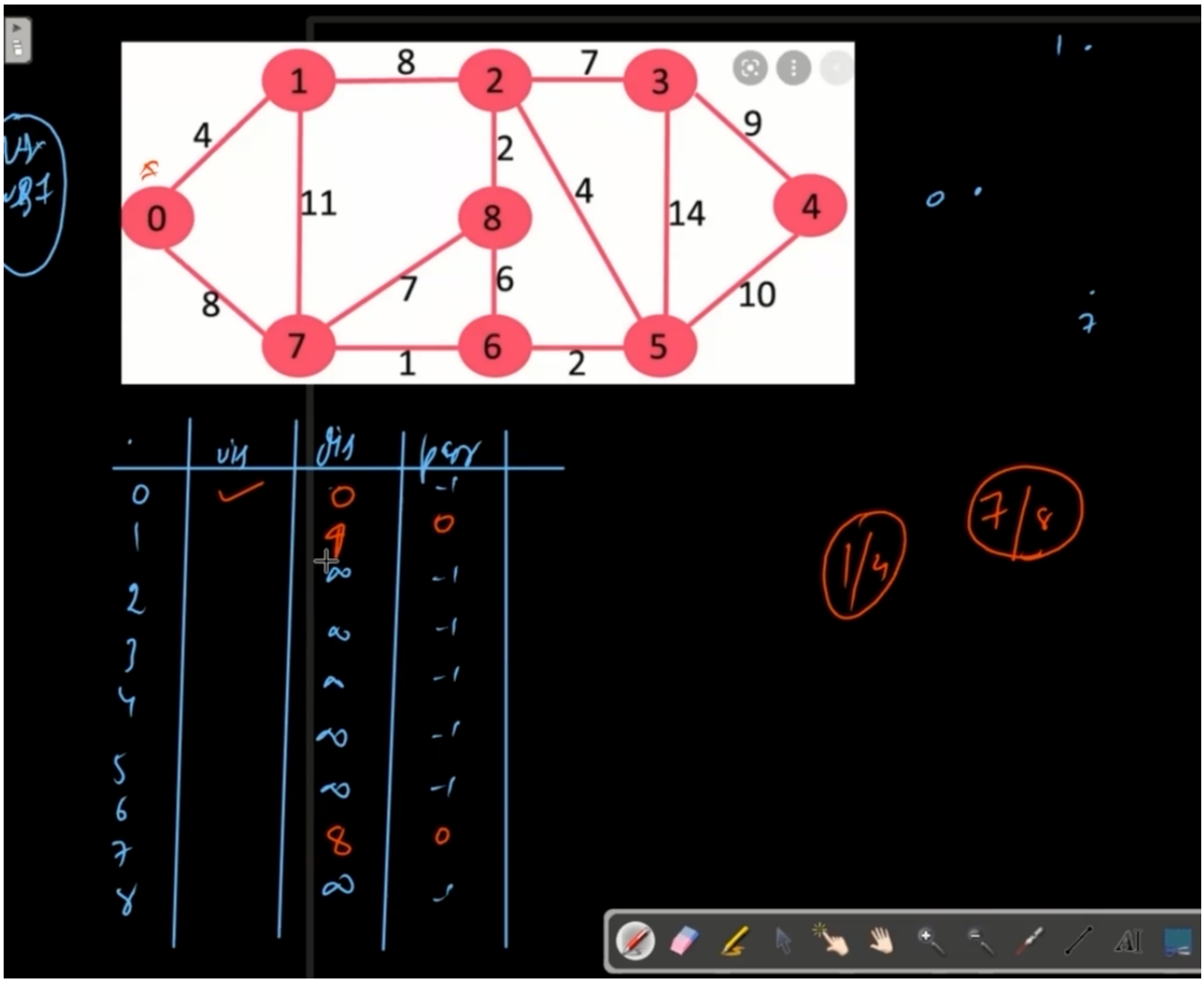
```java
        boolean[] vis = new boolean[n + 1];
        int[] dis = new int[n + 1]; // Keeping a distance array to store the min distance(with weight) of every
                                    // vertex.
        Arrays.fill(dis, (int) 1e9);

        pq.add(new int[] { k, 0 });

        while (pq.size() != 0) {
            int[] rn = pq.remove();
            int u = rn[0], wsf = rn[1];
            if (vis[u])
                continue;

            vis[u] = true;
            dis[u] = wsf;

            for (int[] e : graph[u]) {
                int wt = e[1], v = e[0];
                if (!vis[v]) {
                    pq.add(new int[] { v, wsf + wt });
                }
            }
        }

        int maxDistance = 0;
        for (int i = 1; i < n + 1; i++) {
            if (dis[i] == (int) 1e9) // if any value is still (int)1e9, it means that that vertex was never reached.
                return -1;

            maxDistance = Math.max(maxDistance, dis[i]);
        }

        return maxDistance;
    }
```

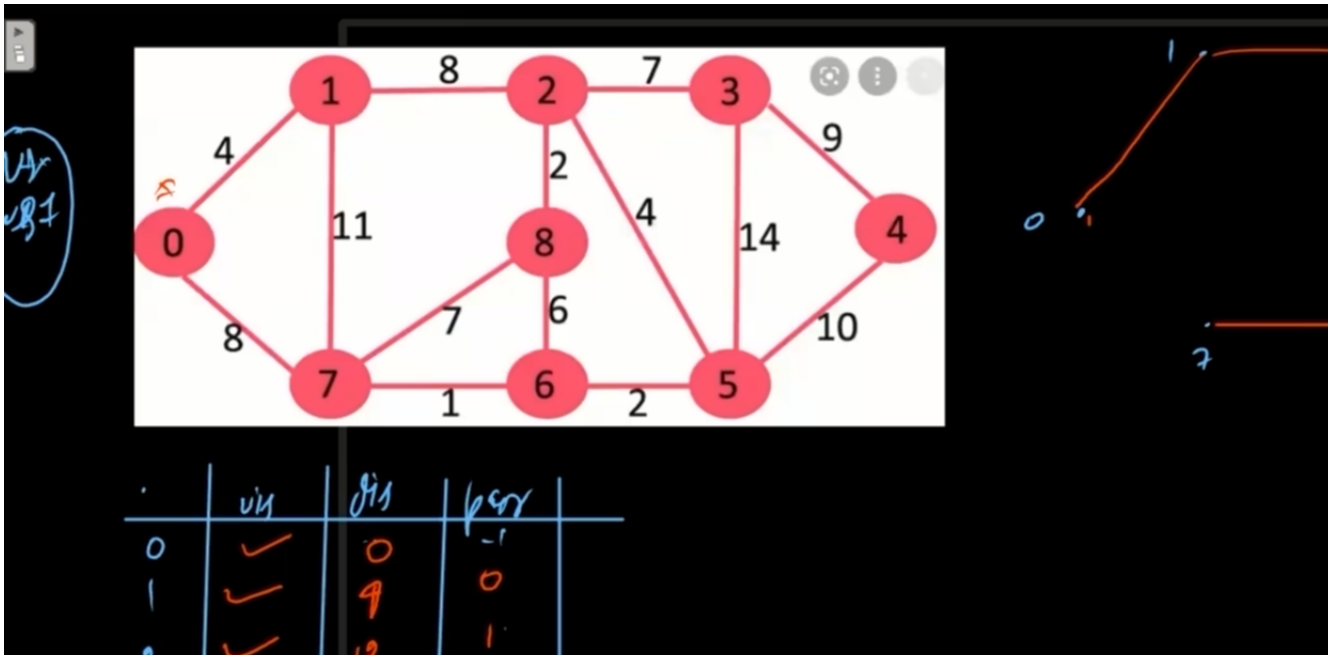
48. Dijkstra Better

```java
    // b <============ Dijikstra Better ==============>
    // Jaise he mai elements ko queue mai dalunga, waise he mai apna distance ka
    // array update marunga

    // Intially mera distance ka array ∞ hoga. Ab jaise he koi element aaya, mai ye
    // check karunga ki wo element jis wsf ke sath aaya hai wo distance array mai jo
    // value rakhi hai wo usse better hai ki nhi

    // ? So basically ye distance ka array kuch elements jinki need nhi hai unko
    // ? dalne ne nhi deta priority queue mai. Isse iski complexity thodi si better
    // ? ho jati hai.

    // Per iski jo Best complexity aati hai wo AVL Tree/ BST ko use karke he aati
    // hai

    // # Parent naam ka array mujhe shortest path in terms of weight nikalne
```

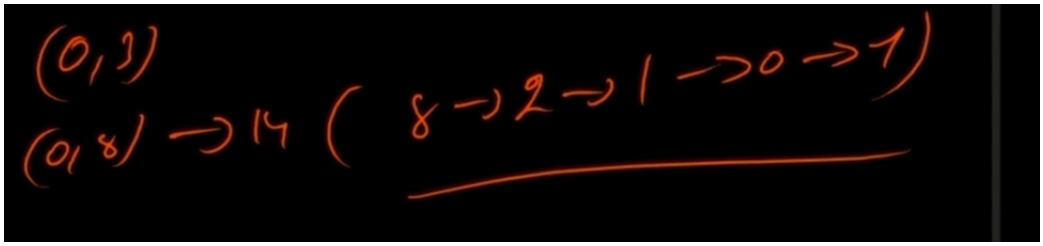
```
// # mai help karta hai.

// Agar mujhe kisi vertex tak ka path chahiye, to mai usse puchunga ki tera
// parent kaun hai.Tab wo jo answer dega mai usse dubara question karrunga ki
// tera parent kaun hai. Aisa mai tab tak karta rahunga jab tak mai source ke
// parent tak nhi pahunch jata. joki hai -1. Aise muhje pura path mil jayega.

// # Distance naam ka array mujhe O(1) mai min distance(lightest path ka
// # distance) nikal ke deta hai.
```

Same Bfs lagana hai bas thoda sa change karke++++++++++++++++++++++++++





```
// Per iski jo Best complexity aati hai wo AVL Tree/ BST ko use karke he aati
// hai ya fir priority queue aur hashmap ko use karke. Per kyunki hashmap ki
// complexity already he bekar hoti hai to, to pehla wali jyada achi complexity
// deta hai. Per worst case mai dono ki same hoti hai.
```



Finding path using the parent array.

```
public static void dijikstra_Btr(ArrayList<Edge>[] graph, int V, int src) {
    ArrayList<Edge>[] mygraph = new ArrayList[V];
    for (int i = 0; i < V; i++)
        graph[i] = new ArrayList<>();

    boolean[] vis = new boolean[V];

    // {vtx, wsf} Pair used
    int[] dis = new int[V];
    int[] par = new int[V];

    Arrays.fill(dis, (int) (1e9));
    Arrays.fill(par, -1);

    PriorityQueue<pair> pq = new PriorityQueue<>((a, b) -> {
        return a.wsf - b.wsf;
    });
```

```
    pq.add(new pair(src, wsf: 0));
    par[src] = -1; // initially source ka parent -1
    dis[src] = 0; // source ka kudh se distance 0.

    while (pq.size() != 0) {

        pair rn = pq.remove();
        int vtx = rn.src, wsf = rn.wsf;

        if (vis[vtx]) // wsf>=dis[vtx] == > basically agar mera wsf dis[vtx] se bada ya equal hai to
                      // mujhe distance array ko update karne ki jaroorat he nhi hai to continue kar
                      // do. To matlab cycle hai
            continue;

        vis[vtx] = true;
        for (Edge e : graph[vtx]) {
            int v = e.v, w = e.w;
            if (!vis[v] && wsf + w < dis[v]) { // Agar mera distance wsf + w se jyada hai, iska matlab mujhe koi
                                               // aisa mila hai jo meretak current dis[v] se bhi lightest path ke
                                               // sath pahunch jayego, to maine tab apne distance ko mara update;
                dis[v] = wsf + w;
                par[v] = vtx; // Aur parent ko bhi sath mai update mar diya.
                pq.add(new pair(v, wsf + w));
            }
```

```
    }
}
```

49. Network Delay Time(Better Solution)

```
// b <============== 743. Network Delay Time(with better complexity)===========>
// https://leetcode.com/problems/network-delay-time/

public int networkDelayTime_(int[][] times, int n, int k) {

    // Edge : {v,w}
    ArrayList<int[]>[] graph = new ArrayList[n + 1];

    for (int i = 0; i < n + 1; i++)
        graph[i] = new ArrayList<>();

    for (int[] time : times) {
        graph[time[0]].add(new int[] { time[1], time[2] });
    }
    PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> {
        return a[1] - b[1];
    });

    boolean[] vis = new boolean[n + 1];
    int[] dis = new int[n + 1];

    Arrays.fill(dis, (int) 1e9);
```

```
    pq.add(new int[] { k, 0 });
    dis[k] = 0;
    while (pq.size() != 0) {
        int[] rn = pq.remove();
        int u = rn[0], wsf = rn[1];
        if (vis[u])
            continue;

        vis[u] = true;

        for (int[] e : graph[u]) {
            int wt = e[1], v = e[0];
            if (!vis[v] && wt + wsf < dis[v]) {
                dis[v] = wt + wsf;
                pq.add(new int[] { v, wsf + wt });
            }
        }
    }

    int maxDistance = 0;
    for (int i = 1; i < n + 1; i++) {
        if (dis[i] == (int) 1e9)
            return -1;

        maxDistance = Math.max(maxDistance, dis[i]);
    }
    return maxDistance;
}
```

50. Cheapest Flights Within K Stops

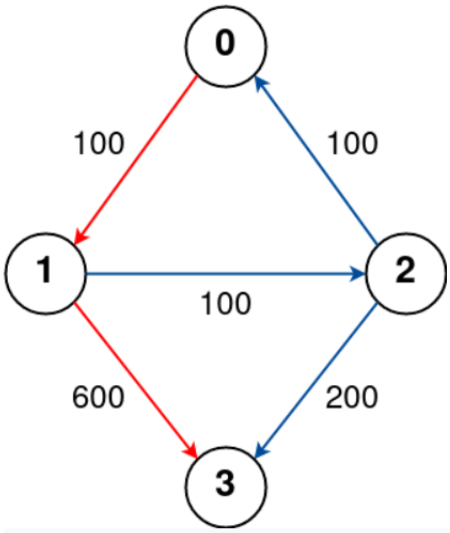
787. Cheapest Flights Within K Stops

Medium 5888 269 Add to List Share

There are `n` cities connected by some number of flights. You are given an array `flights` where `flights[i] = [from_i, to_i, price_i]` indicates that there is a flight from city `from_i` to city `to_i` with cost `price_i`.

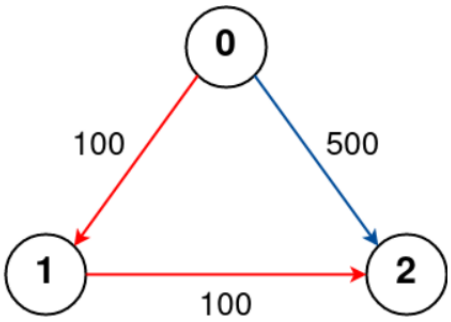
You are also given three integers `src`, `dst`, and `k`, return ***the cheapest price*** *from* `src` *to* `dst` *with at most* `k` *stops.* If there is no such route, return `-1`.

Example 1:



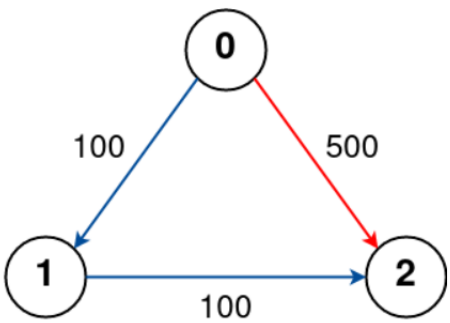

```
Input: n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3,
k = 1
Output: 700
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 =
700.
```

Example 2:




```
Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1
Output: 200
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 =
200.
```

Example 3:




```
Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 0
Output: 500
Explanation:
The graph is shown above.
The optimal path with no stops from city 0 to 2 is marked in red and has cost 500.
```

Constraints:

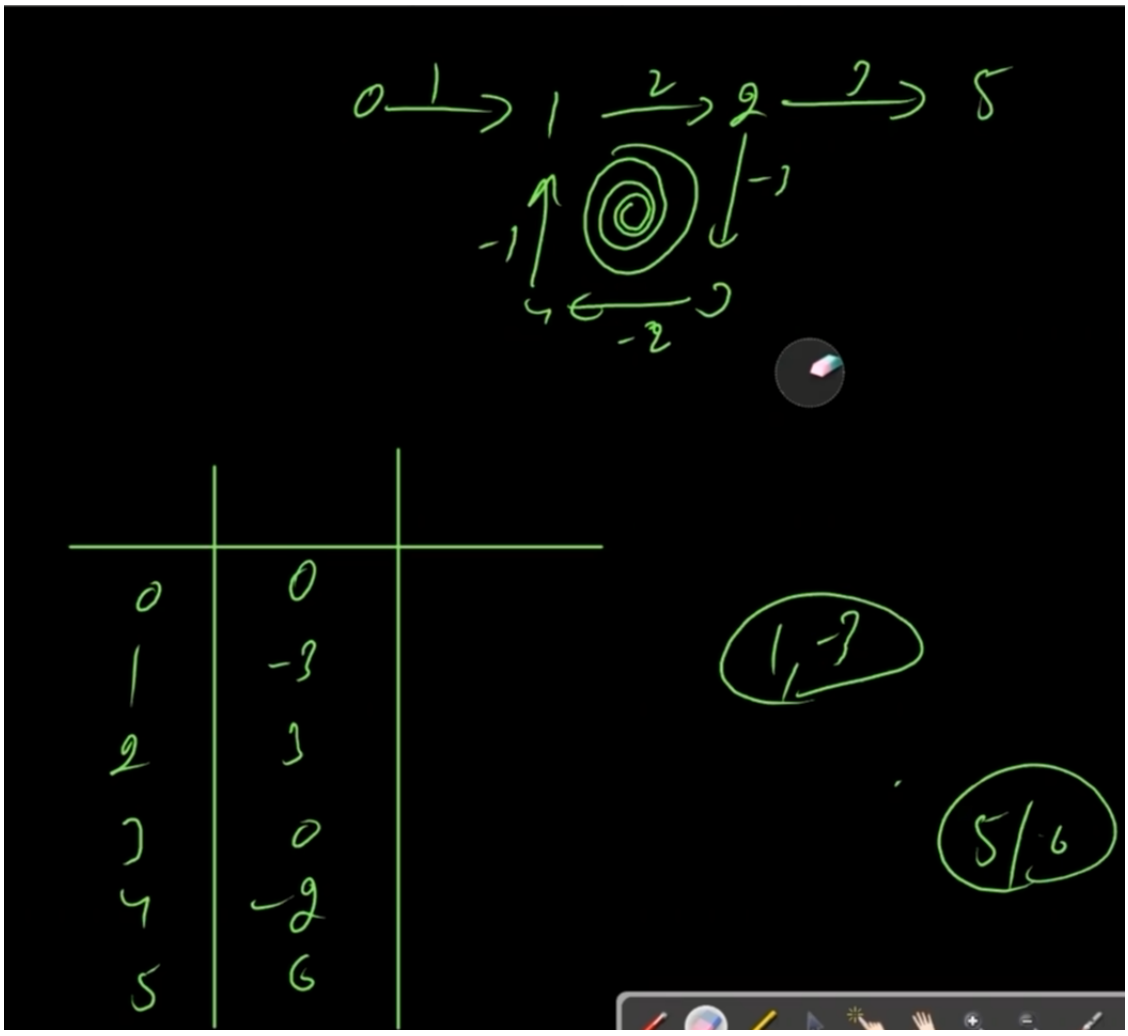
- `1 <= n <= 100`
- `0 <= flights.length <= (n * (n - 1) / 2)`
- `flights[i].length == 3`
- `0 <= from_i, to_i < n`
- `from_i != to_i`
- `1 <= price_i <= 10^4`
- There will not be any multiple flights between two cities.
- `0 <= src, dst, k < n`
- `src != dst`

```
// b <============== 787. Cheapest Flights Within K Stops ====>
// https://leetcode.com/problems/cheapest-flights-within-k-stops

// Good test case to dry run

// 4
// [[0,1,1],[0,2,5],[1,2,1],[2,3,1]]
// 0
// 3
// 1

// Jo uper test case hai uspe dry run kar. Usse tereko samaj aayega ki hum na
// distance array ko use kar sakte hai aur na he visited ko. To agar isko simply
// he karna start kar denge to TLE aaya.

// vis aur distance use karne se ye hoga ki jo mere edges jo shi wt aur steps ke
// sath badme aa rahe the, unhe wo queue mai dalne he nhi dega. Jisse mujhe
// answer kabhi mil he nhi payega

// To aisa kya karein ab ??

// ? The code below will give TLE since the number of comparison have incereased
// ? since their is no boundation of which edge can enter the queue. All the
// ? edge are entering the queue.
```

```
public int findCheapestPrice(int n, int[][] flights, int src, int dst, int k) {
    ArrayList<int[]>[] graph = new ArrayList[n];
    for (int i = 0; i < n; i++)
        graph[i] = new ArrayList<>();

    // Edge {v,w}
    for (int[] flight : flights) {
        int u = flight[0], v = flight[1], w = flight[2];
        graph[u].add(new int[] { v, w });
    }

    // Pair {vtx,wsf,steps}
    PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> {
        return a[1] - b[1];
    });

    boolean[] vis = new boolean[n];

    pq.add(new int[] { src, 0, -1 });

    int cheapPrice = (int) 1e9;
    while (pq.size() != 0) {
        int[] rn = pq.remove();
        int vtx = rn[0], wsf = rn[1], steps = rn[2];

        if (vtx == dst && steps <= k && wsf < cheapPrice) {
            cheapPrice = wsf;
        }
        for (int[] Edge : graph[vtx]) {
            int v = Edge[0], w = Edge[1];
            pq.add(new int[] { v, wsf + w, k + 1 });
        }
    }

    return cheapPrice == (int) 1e9 ? -1 : cheapPrice;
}
```

51. BellMan Ford Algo

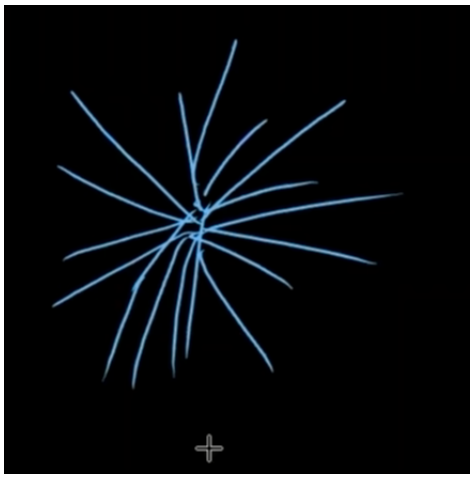
```
// b <============== Belman Ford Algo ================>

// Dijikastra hume agar -ve edges hote hain to shi answer nikal ke nhi deta hai.
// Auur agar dijkstra mai -ve weight ki cycle milti hai to ∞ loop chal jata hai.
// Jiskko hum apne aap rok to sakte hai pr tabhi bhi wo shi answer nhi nikal ke
// deta.

// # Dijkstra may aur may not correct answer dede in -ve edges.
// This may be true when the cycle overall weight will be +ve. Sirf correct
// answer aane ki possibility.

// Belman ford ye bataata hai ki graph ke andar -ve weight ka cycle hai ki nhi.
// Iske liye Graph edges ki form mai he hona chahiye.

// ? Agar mere graph mai -ve cycle hai (overall cycle weight is -ve) to answer
// ? possible he nhi hai. Matlab jo bhi answer aayega wo galat aayega.

// Dijkstra humesha ek spannig tree nikal ke deta hai aur ek spanning tree mai
// V - 1 edges ki requirement hoti hai to reach every vertex. Matlab mai at most
// V -1 edges ka use karke har vertex tak jaa sakta hun.

// So agar V-1 ke baad koi bhi update hua, iska matlab ye hai ki -ve weight ki
// cycle hai.
```



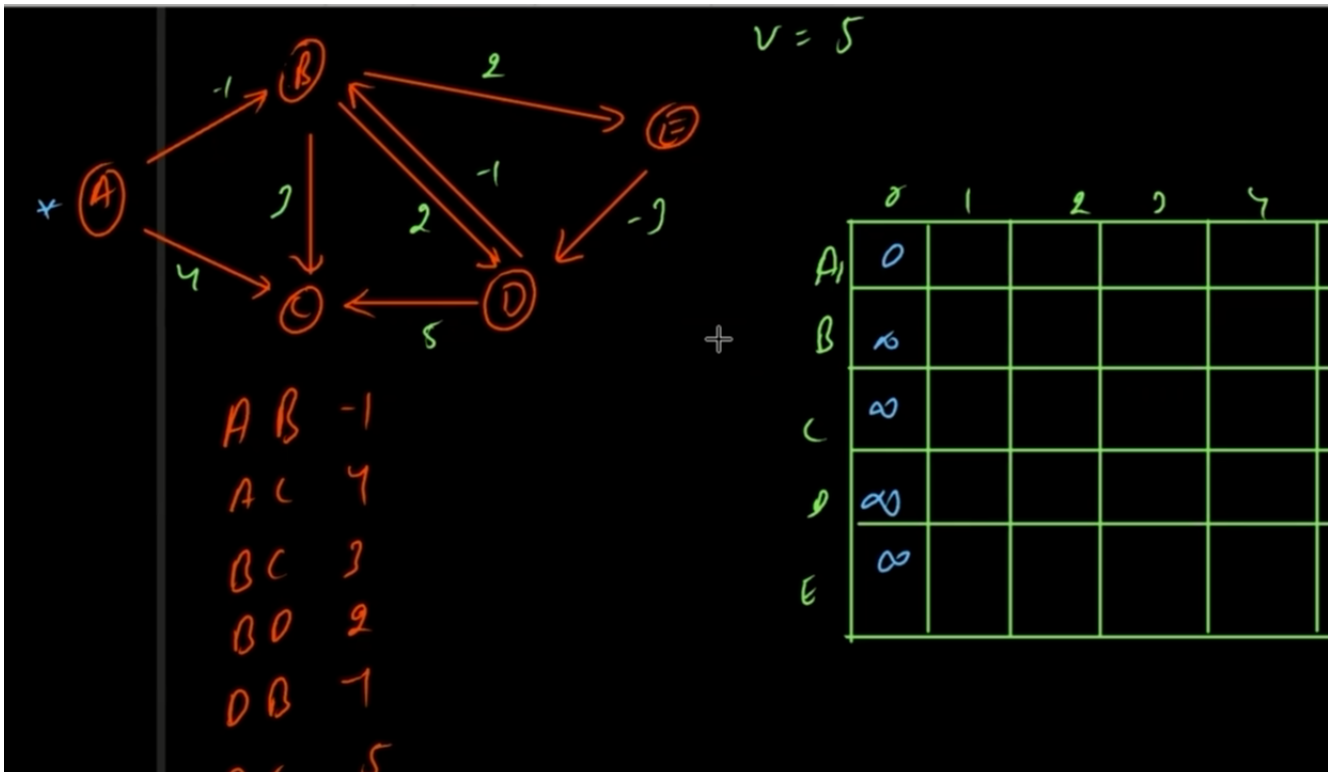

```
// ! Belmanford is also a single source algo.

// Yahan pe edges chahiye hoti hain.

// Yahan pe do array use honge. Prev aur current.

// ? Matrix is used to just get a good idea of about what is happening. Hume bas
// ? do array use karne hai prev aur current. Aur har iteration ki starting mai
```

```
// ? prev ko current mai copy kar dena hai.
```

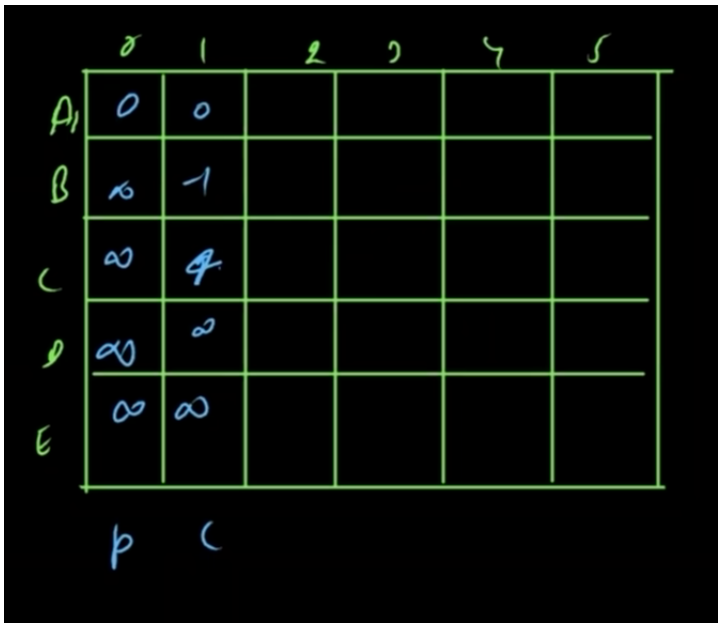
```
// # How to dry run
// Jitni vertex hain, utni kar sari edges ko traverse karna hai
// Ab manle AB edge aayi with wt = -1. To hum prev mai check karenge ki A kis
// weight ke sath aaya hai. Aur usme AB edge ki wt add kardenge to B mai us
// value ke sath pahunchenge. To Agar kam weight ke sath B pahunchenge to B ko
// current wale array mai update kar denge.

// # If current array does not update
// Agar Kahin pe bhi current wala array update nhi hota hai, to matlab aage aur
// traverse karne ki jaroorat nhi hai. Just whin pe break kar do. For ex :
// Agar ek graph hai . Ek single vertex ko sare vertex point kar rahe hon.
// To isi case mai hume aage traverse nhi karna hai.

// # For Negative Cycle
// Agar last edge jo hum extra traverse kar rahe hain. (extra isiliye kyunki
// hume sirf V - 1 edges he chahiye hote hain), agar usme current wala array ek
// ` bar bhi update hota hai, to iska matlab ye hai ki negativeCycle Exist karti
// hai. To bas tab hume sahi answer nhi mil sakta kyunki jaise ki uper bataya ki
// ∞ loop chalta hai to wo har bar he array ko update karta rahega. To answer
// galat milega.

// # Kyunki Number of edges = V -1 for spanning tree that is created by dikstra.
// Therefore yahan pe edgeCount ye signify kar raha hai ki atmost kitni edge ke
// sath mai kisi vertex pe sabse kam wt ke sath pahunch sakta hun
```



Case of no update

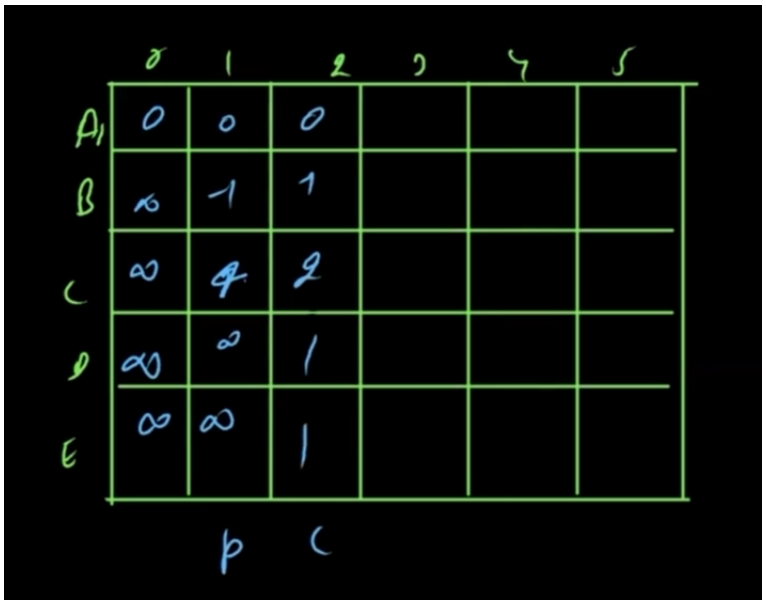


D C
G D -7
D E 2



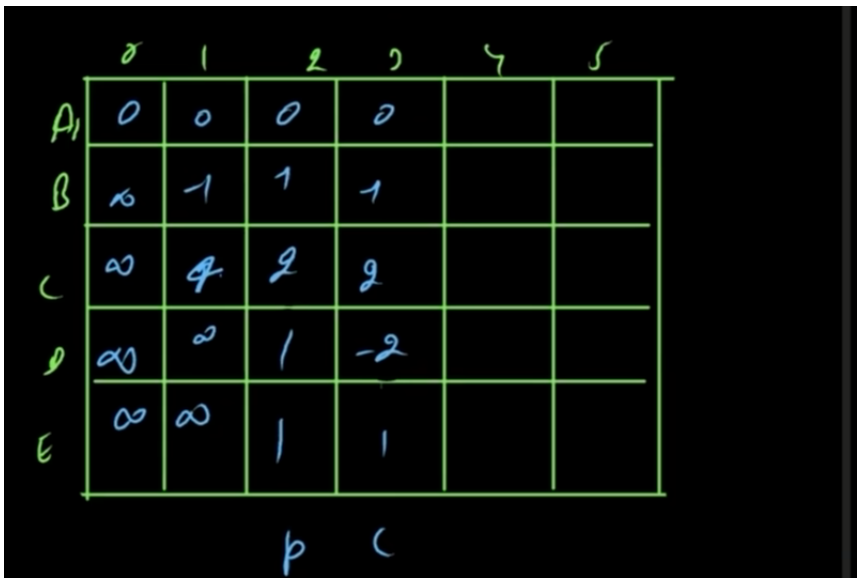
	0	1	2	3	4	5
A	0	0				
B	∞	-1				
C	∞	4				
D	∞	∞				
E	∞	∞				

P C



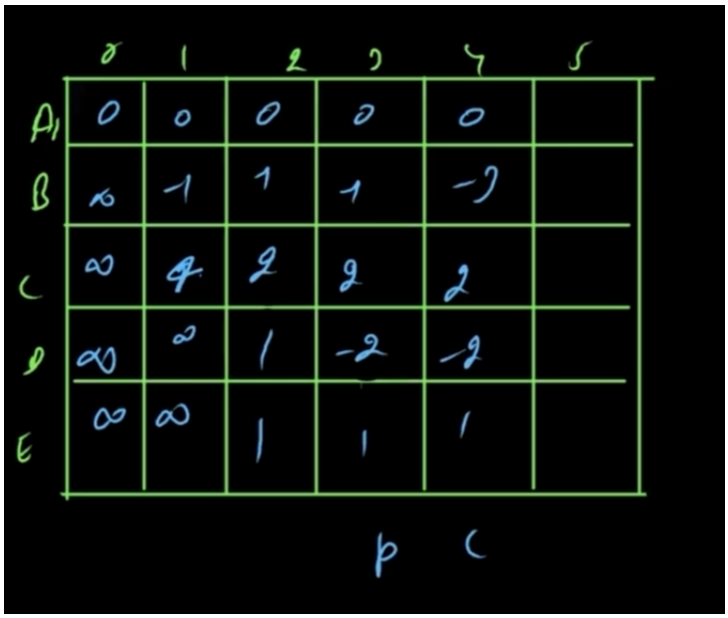
	0	1	2	3	4	5
A	0	0	0			
B	∞	-1	1			
C	∞	4	2			
D	∞	∞	1			
E	∞	∞	1			

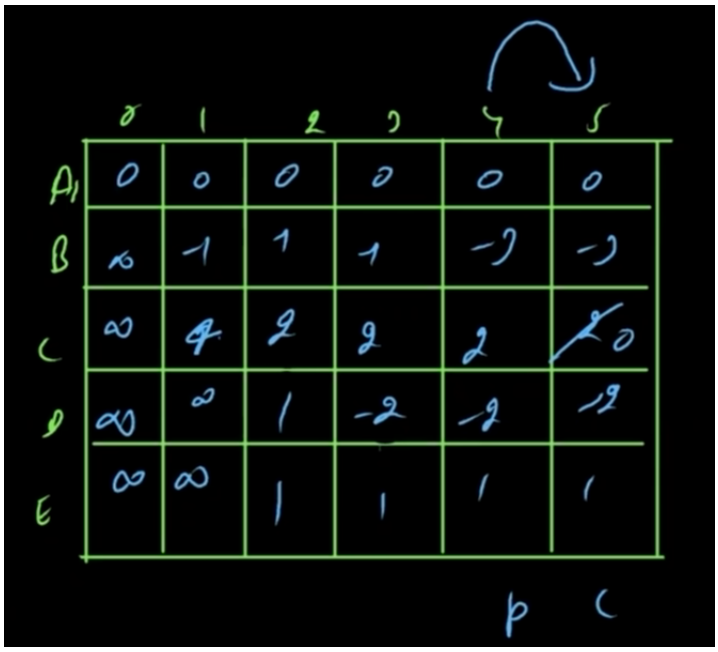
P C



	0	1	2	3	4	5
A	0	0	0	0		
B	∞	-1	1	1		
C	∞	4	2	2		
D	∞	∞	1	-2		
E	∞	∞	1	1		

P C





```java
// # edge : {src,dest,weight}
public static void bellmanFord(int[][] edges, int N, int src) {
    int[] prev = new int[N];
    Arrays.fill(prev, (int) 1e9);
    prev[src] = 0;

    boolean isNegativeCycle = false;
    for (int edgeCount = 1; edgeCount <= N; edgeCount++) { // edgeCount ki value ye denote karti hai ki mai atmost
                                                           // kitni edges ko use karke mai minimum weight ke sath us
                                                           // vertex tak pahuchunga
        int[] curr = new int[N];
        for (int i = 0; i < N; i++)
            curr[i] = prev[i];
```

```java
        boolean isAnyUpdate = false;
        for (int[] e : edges) {
            int u = e[0], v = e[1], w = e[2];
            if (prev[u] + w < curr[v]) { // mai prev mai jis weight ke sath aaya tha, agar usme mai current edge ka
                                         // weight add kardun aur wo curr[v] se kam hai to use update kardo.
                curr[v] = prev[u] + w;
                isAnyUpdate = true;
            }
        }
```

```
            prev = curr; // Prev ko ab current mai point kara diya aur ab next iteration mai prev ko
                         // current mai dubara copy kar denge

            if (edgeCount == N && isAnyUpdate) // Jo last edgeCount hai wo excess hai kyunki hume at most V - 1 edges
                                               // required hoti hain sari vertex tak pahunchne ke liye. To agar edge
                                               // count == N hua aur koi bhi update hua us time current array mai, to
                                               // -ve cycle hai. Aisa isiliye kyunki update tabhi hoga jab aur kam
                                               // weight ke sath aayenge, to wo ek loop chalta rahega.
                isNegativeCycle = true;

            if (!isAnyUpdate) // Agar kabhi bhi aisa hua ki curr ek bar kabhi bhi update nhi hua to mujhe aage
                              // traverse karne ki jaroorat he nhi hai. Whin pe break kardo. Ye kab hoga jab
                              // Agar ek graph hai . Ek single vertex ko sare vertex point kar rahe hon.To isi
                              // case mai hume aage traverse nhi karna hai.
                break;
        }
    }
```

52. Cheapest Flights Within K Stops

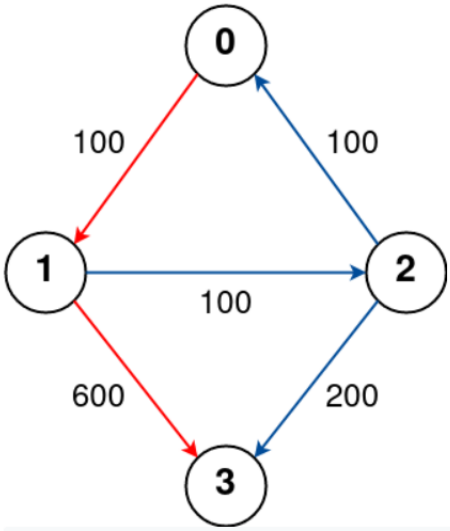
787. Cheapest Flights Within K Stops

Medium 5888 269 Add to List Share

There are `n` cities connected by some number of flights. You are given an array `flights` where `flights[i] = [from_i, to_i, price_i]` indicates that there is a flight from city `from_i` to city `to_i` with cost `price_i`.

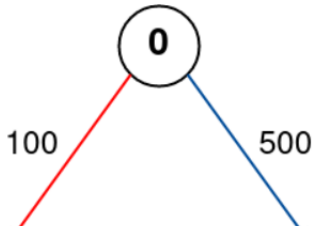
You are also given three integers `src`, `dst`, and `k`, return ***the cheapest price*** *from* `src` *to* `dst` *with at most* `k` *stops.* If there is no such route, return `-1`.

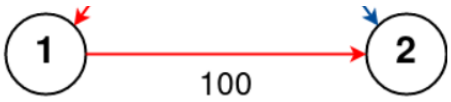
Example 1:



Input: n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1
Output: 700
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700.

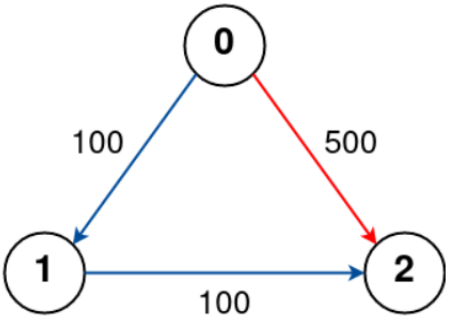
Example 2:





Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1
Output: 200
Explanation:
The graph is shown above.
The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 = 200.

Example 3:



Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 0
Output: 500
Explanation:
The graph is shown above.
The optimal path with no stops from city 0 to 2 is marked in red and has cost 500.

Constraints:

- `1 <= n <= 100`
- `0 <= flights.length <= (n * (n - 1) / 2)`
- `flights[i].length == 3`
- $0 <= from_i, to_i < n$
- $from_i != to_i$
- $1 <= price_i <= 10^4$
- There will not be any multiple flights between two cities.
- `0 <= src, dst, k < n`
- `src != dst`

```java
// b <============== 787. Cheapest Flights Within K Stops ====>
// https://leetcode.com/problems/cheapest-flights-within-k-stops

// Here used belman ford algo
// edgeCount denote atmost kitne edges ko use karke hum vertex tak kitne minimum
// weight ke sath pahunche

public int findCheapestPrice_(int n, int[][] flights, int src, int dest, int k) {

    int[] prev = new int[n];
    Arrays.fill(prev, (int) 1e9);
    prev[src] = 0;

    for (int edgeCount = 1; edgeCount <= n; edgeCount++) {
        int[] curr = new int[n];
        for (int i = 0; i < n; i++)
            curr[i] = prev[i];

        for (int[] edge : flights) {
            int u = edge[0], v = edge[1], w = edge[2];

            if (prev[u] + w < curr[v]) {
                curr[v] = prev[u] + w;
            }
        }
        prev = curr;

        if (edgeCount == k + 1)
            break;
    }
```

```
        return prev[dest] == (int) 1e9 ? -1 : prev[dest];
    }
```

53. The Maze

490 The Maze

Problem:

There is a ball in a maze with empty spaces and walls. The ball can go through empty spaces by rolling up, down, left or right, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

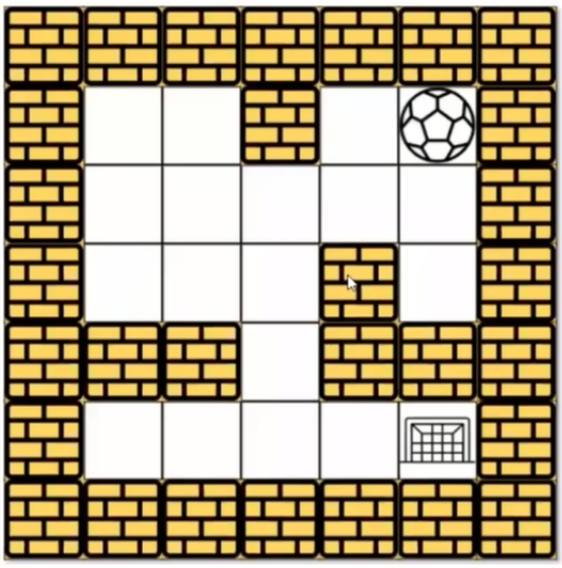
Given the ball's start position, the destination and the maze, determine whether the ball could stop at the destination.

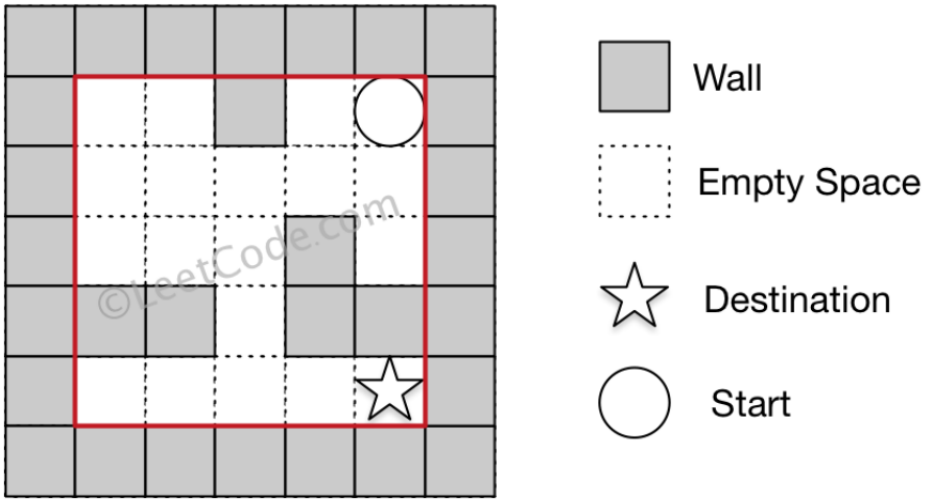
The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The start and destination coordinates are represented by row and column indexes.

Example 1

```
Input 1: a maze represented by a 2D array

0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (4, 4)

Output: true
Explanation: One possible way is : left -> down -> left -> down -> right -> down -> right.
```

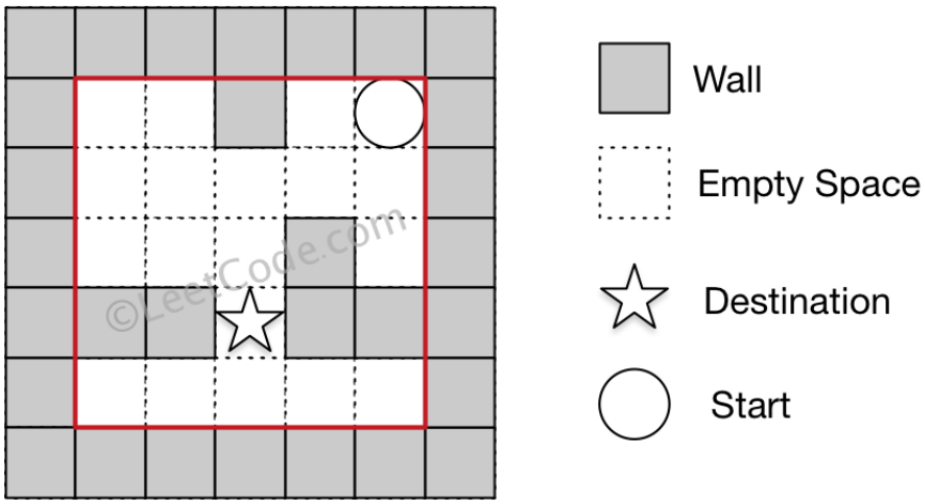
Example 1:





Example 2

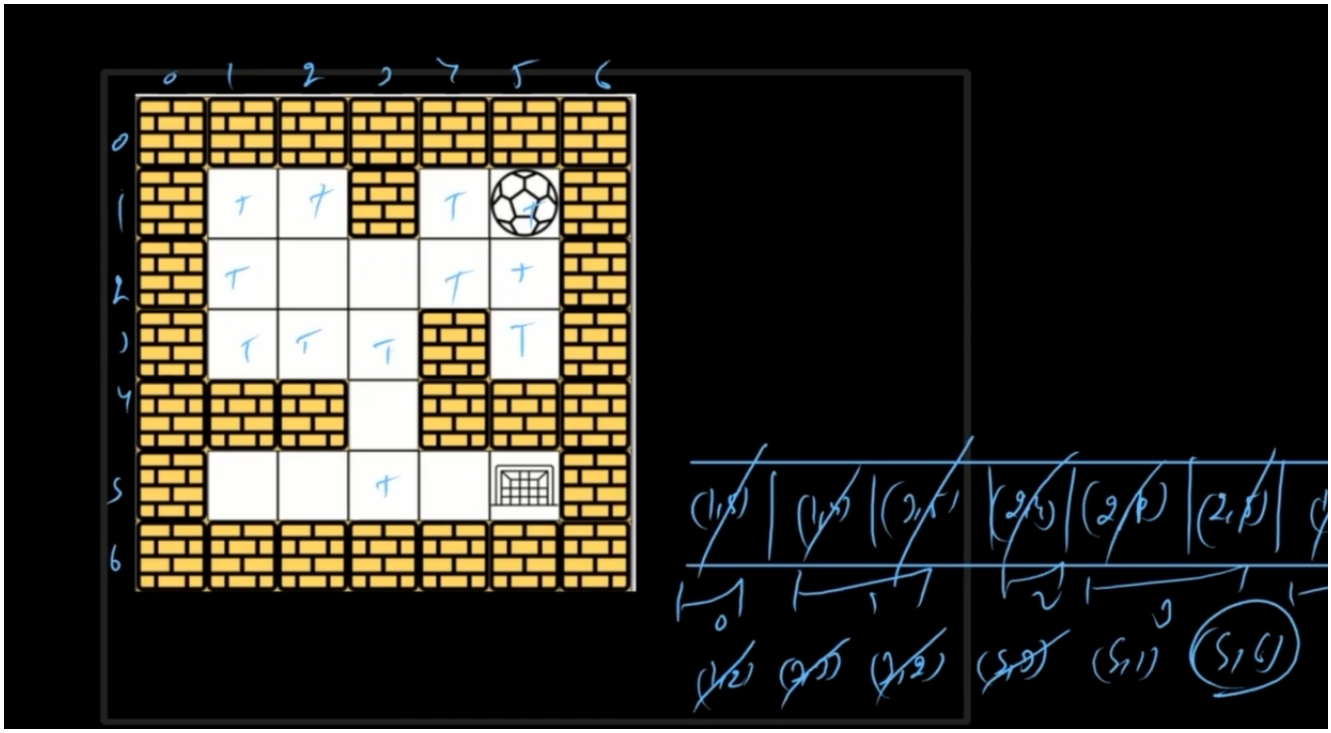
```
Input 1: a maze represented by a 2D array

0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (3, 2)

Output: false
Explanation: There is no way for the ball to stop at the destination.
```



Note:

1. There is only one ball and one destination in the maze.
2. Both the ball and the destination exist on an empty space, and they will not be at the same position initially.
3. The given maze does not contain border (like the red rectangle in the example pictures), but you could assume the border of the maze are all walls.
4. The maze contains at least 2 empty spaces, and both the width and height of the maze won't

exceed 100.



```
// Bhaiya Method
// 490

// Maine kiya kya ki kyunki yahan pe friction less surface de rakhi hai to jahan
// pe bhi ek push mai ball jaki rukeggi usko mai mark kar dunga

// yahi karna hai simple kaam
// ? Yahan pe humne visited ka array liya hai to mark. Why didn't we make
// ? changes in the maze Array?

// Aisa isiliye kiya kyunki agar mai aisa karta to ball jahan pe bhi rukti to
// mai use mark kar deta, aur next time wo mere liye wall ka kaam karta jo ki
// galat ho kyunki wall to wahan pe hain he nhi. Isse hume galat answer milta
```

```
public boolean hasPath_(int[][] maze, int[] start, int[] destination) {
    int n = maze.length, m = maze[0].length, sr = start[0], sc = start[1], er = destination[0], ec = destination[1];
    LinkedList<Integer> que = new LinkedList<>();
    boolean[][] vis = new boolean[n][m];
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

    que.add(sr * m + sc); // converted it into 1-d
    vis[sr][sc] = true;

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int idx = que.removeFirst(), i = idx / m, j = idx % m;
            for (int[] d : dir) {

                int r = i, c = j;
                while (r >= 0 && c >= 0 && r < n && c < m && maze[r][c] == 0) { // The radius loop in a different
                                                                                // way
                    r += d[0];
                    c += d[1];
                }
```

```
                // Direction minus isiliye kiya Taki mai 1 se pehle ka just 0 tha wahan pe aaun
                // aur mujhe wo r and c mil jaye
                r -= d[0];
                c -= d[1];

                if (vis[r][c])
                    continue;
```

```
                vis[r][c] = true;
                que.addLast(r * m + c);
                if (r == er && c == ec)
                    return true;
            }

        }
    }

    return false;
}
```

My Method Below :

```
public boolean hasPath(int[][] maze, int[] start, int[] destination) {
    // write your code here
    int n = maze.length, m = maze[0].length;
    int er = destination[0], ec = destination[1];
    LinkedList<int[]> que = new LinkedList<>();
    que.add(start);

    boolean[][] vis = new boolean[n][m];
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    vis[start[0]][start[1]] = true;
    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {

            int[] rn = que.removeFirst();
            int sr = rn[0], sc = rn[1];

            for (int d = 0; d < dir.length; d++) {
                int r = sr, c = sc;
                for (int rad = 1; rad < Math.max(n, m); rad++) {
                    r = sr + rad * dir[d][0];
                    c = sc + rad * dir[d][1];

                    if (r >= 0 && c >= 0 && r < n && c < m) {
                        if (maze[r][c] == 0)
                            continue;
                        else
                            break;
                    } else
                        break;
                }

                r -= dir[d][0];
                c -= dir[d][1];

                if (vis[r][c])
                    continue;
                vis[r][c] = true;
                que.addLast(new int[] { r, c });
                if (r == er && c == ec)
                    return true;
            }

        }
    }
    return false;
```

54. Maze II

505. The Maze II

Problem:

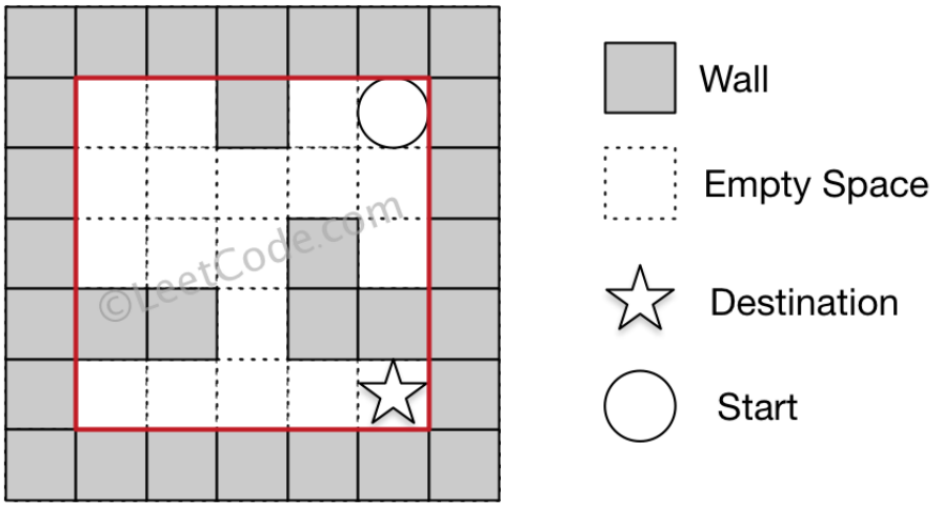
There is a ball in a maze with empty spaces and walls. The ball can go through empty spaces by rolling up, down, left or right, but it won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction.

Given the ball's start position, the destination and the maze, find the shortest distance for the ball to stop at the destination. The distance is defined by the number of empty spaces traveled by the ball from the start position (excluded) to the destination (included). If the ball cannot stop at the destination, return -1.

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the maze are all walls. The start and destination coordinates are represented by row and column indexes.

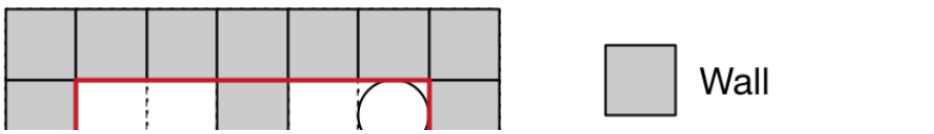
Example 1

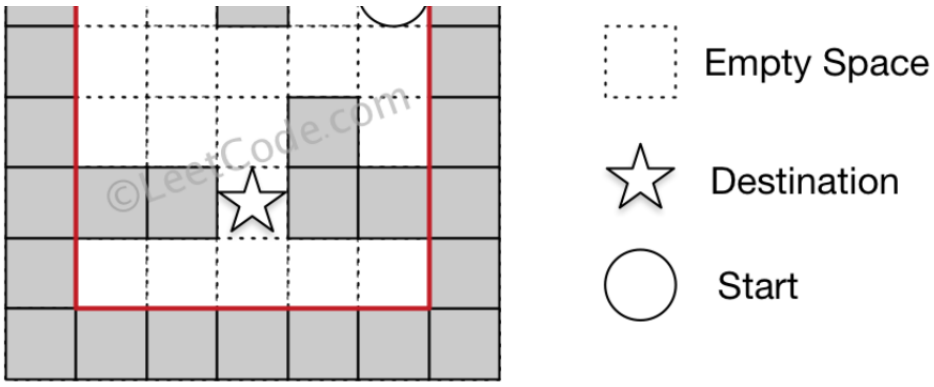
```
Input 1: a maze represented by a 2D array

0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (4, 4)

Output: 12
Explanation: One shortest way is : left -> down -> left -> down -> right -> down -> right.
             The total distance is 1 + 1 + 3 + 1 + 2 + 2 + 2 = 12.
```



Example 2

```
Input 1: a maze represented by a 2D array

0 0 1 0 0
0 0 0 0 0
0 0 0 1 0
1 1 0 1 1
0 0 0 0 0

Input 2: start coordinate (rowStart, colStart) = (0, 4)
Input 3: destination coordinate (rowDest, colDest) = (3, 2)

Output: -1
Explanation: There is no way for the ball to stop at the destination.
```





Note:

1. There is only one ball and one destination in the maze.
2. Both the ball and the destination exist on an empty space, and they will not be at the same position initially.
3. The given maze does not contain border (like the red rectangle in the example pictures), but you could assume the border of the maze are all walls.
4. The maze contains at least 2 empty spaces, and both the width and height of the maze won't exceed 100.

```
// b <================== Maze II ==================>
// https://www.lintcode.com/problem/788/description

// ! Important Point :

// # In dijikstra, you cannot use bfs without cycle. It will always give you
// # wrong result.

// ? Why ? Kyunki agar mai pehle he ek vertex ko mark kar diya to wo vertex
// ? dubara dalega he nhi chahe wo baad mai kam weight ke sath he kyun na aaye.
// ? To isiliye bfs with cycle use karte hain.

// Dry run kar samaj mai aa jayega. Take a basic rectangle graph.

// Baki question same uper wale ki tarah hai.

// ! To Dijikstra mai hum bfs without cycle use karte hain ya to distance array
// ! ka use karte hain.
```

```
public int shortestDistance(int[][] maze, int[] start, int[] destination) {
    // write your code here
    int n = maze.length, m = maze[0].length, sr = start[0], sc = start[1], er = destination[0], ec = des
    PriorityQueue<int[]> que = new PriorityQueue<>((a, b) -> {
        return a[2] - b[2];
    });
    boolean[][] vis = new boolean[n][m];
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

    // {row, columnm totalSteps}
    que.add(new int[] { sr, sc, 0 });
    vis[sr][sc] = true;

    while (que.size() != 0) {
        int size = que.size();
        while (size-- > 0) {
            int[] rn = que.remove();
            int i = rn[0], j = rn[1], totalSteps = rn[2];

            if (i == er && j == ec)
```

```
                return totalSteps;

            vis[i][j] = true;
            for (int d = 0; d < dir.length; d++) {
                int r = i, c = j;
                int steps = 0;
                while (r >= 0 && c >= 0 && r < n && c < m && maze[r][c] == 0) {
                    r += dir[d][0];
                    c += dir[d][1];
                    steps++;
                }

                r -= dir[d][0];
                c -= dir[d][1];
                steps--;
                if (!vis[r][c])
                    que.add(new int[] { r, c, totalSteps + steps });

            }

        }
    }
    return -1;
}
```

```
// b With Distance Array : Bhaiya's Code

public static class pair_ implements Comparable<pair_> {

    int r = 0, c = 0, steps = 0;

    pair_(int r, int c, int steps) {
        this.r = r;
        this.c = c;
        this.steps = steps;
    }

    @Override // Override to check if the override function name is same as written in java
    public int compareTo(pair_ o) {
        return this.steps - o.steps; // same this - other concept
    }

}
```

```
public int shortestDistance_(int[][] maze, int[] start, int[] destination) {
    int n = maze.length, m = maze[0].length, sr = start[0], sc = start[1], er = destination[0], ec = de
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

    int[][] dis = new int[n][m];
    for (int[] d : dis)
        Arrays.fill(d, (int) 1e8);

    PriorityQueue<pair_> que = new PriorityQueue<>();
    que.add(new pair_(sr, sc, steps: 0));
    dis[sr][sc] = 0;

    while (que.size() != 0) {
        pair_ p = que.remove();
        if (p.r == er && p.c == ec)
            return p.steps;

        for (int[] d : dir) {
            int r = p.r, c = p.c, steps = p.steps;
            while (r >= 0 && c >= 0 && r < n && c < m && maze[r][c] == 0) {
                r += d[0];
```

```
                c += d[1];
                steps++;
            }

            r -= d[0];
            c -= d[1];
            steps--;

            if (steps >= dis[r][c])
                continue;

            que.add(new pair_(r, c, steps));
            dis[r][c] = steps;
        }
    }

    return -1;
}
```

55. Maze III

499. The Maze III

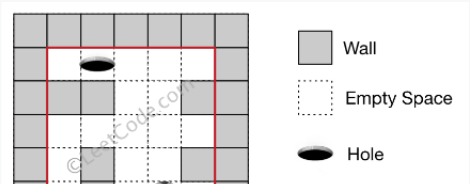
There is a **ball** in a maze with empty spaces and walls. The ball can go through empty spaces by rolling **up** (u), **down** (d), **left** (l) or **right** won't stop rolling until hitting a wall. When the ball stops, it could choose the next direction. There is also a **hole** in this maze. The ball v the hole if it rolls on to the hole.

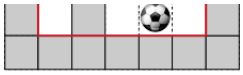
Given the **ball position**, the **hole position** and the **maze**, find out how the ball could drop into the hole by moving the **shortest distan** distance is defined by the number of **empty spaces** traveled by the ball from the start position (excluded) to the hole (included). Outpu **directions** by using 'u', 'd', 'l' and 'r'. Since there could be several different shortest ways, you should output the **lexicographically sma** ball cannot reach the hole, output "impossible".

The maze is represented by a binary 2D array. 1 means the wall and 0 means the empty space. You may assume that the borders of the walls. The ball and the hole coordinates are represented by row and column indexes.

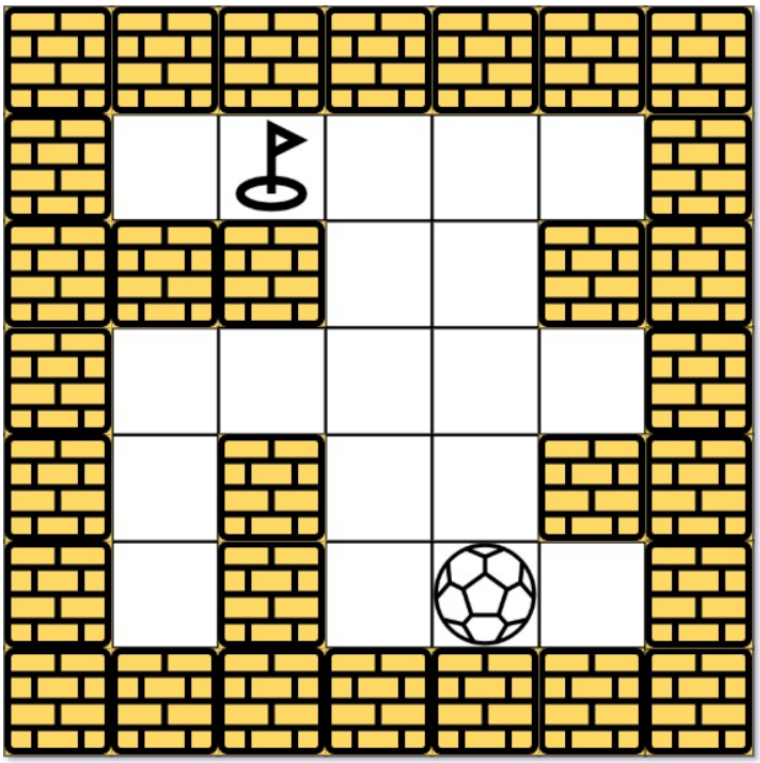
Example 1:

```
Input 1: a maze represented by a 2D array

0 0 0 0 0
1 1 0 0 1
0 0 0 0 0
0 1 0 0 1
0 1 0 0 0

Input 2: ball coordinate (rowBall, colBall) = (4, 3)
Input 3: hole coordinate (rowHole, colHole) = (0, 1)

Output: "lul"

Explanation: There are two shortest ways for the ball to drop into the hole.
The first way is left -> up -> left, represented by "lul".
The second way is up -> left, represented by 'ul'.
Both ways have shortest distance 6, but the first way is lexicographically smaller because 'l' < 'u'. So the output is "lul"
```





Ball

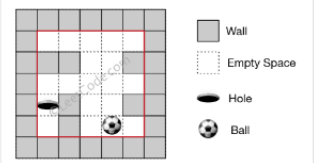


Example 2

Input 1: a maze represented by a 2D array

```
0 0 0 0 0
1 1 0 0 1
0 0 0 0 0
0 1 0 0 1
0 1 0 0 0
```

Input 2: ball coordinate (rowBall, colBall) = (4, 3)
Input 3: hole coordinate (rowHole, colHole) = (3, 0)
Output: "impossible"
Explanation: The ball cannot reach the hole.




```
// b <=================== Maze III ========================>
// Leetcode 499

// Yahan pe basically lexographically sort wali first string return karni hai
// agar kahin pe jane ke liye steps equal hain. To isiliye comparator mai string
// ka comparison add kiya taki agar steps equal ho to wo pehle wali string
// nikale jo lexographically pehle aaye

// ? Baki puar same hai. Thode se hole ke check add kiye hain. Bas. Same uper
// ? wale question ki tarah hai.
```

```
public static class stringPair implements Comparable<stringPair> {
    int r = 0, c = 0, steps = 0;
```

```
    String str = "";

    stringPair(int r, int c, int steps, String str) {
        this.r = r;
        this.c = c;
        this.steps = steps;
        this.str = str;
    }

    @Override
    public int compareTo(stringPair p) {
        if (this.steps != p.steps)
            return this.steps - p.steps;
        return this.str.compareTo(p.str); // To compare two strings lexographically. Same this - other
    }
}
```

```
public static String findShortestWay(int[][] maze, int[] ball, int[] hole) {
    int sr = ball[0], sc = ball[1], er = hole[0], ec = hole[1], n = maze.length, m = maze[0].length;
    PriorityQueue<stringPair> que = new PriorityQueue<>();
    que.add(new stringPair(sr, sc, steps: 0, str: ""));
    int[][] dis = new int[n][m];
    for (int[] d : dis) {
        Arrays.fill(d, (int) 1e9);
    }

    dis[sr][sc] = 0;

    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    String[] dirS = { "d", "u", "r", "l" };

    while (que.size() != 0) {
        int size = que.size();

        while (size-- > 0) {
            stringPair rn = que.remove();
            int i = rn.r, j = rn.c, totalSteps = rn.steps;
            String str = rn.str;

            if (i == er && j == ec)
                return str;

            for (int d = 0; d < dir.length; d++) {
                int r = i, c = j;
```

```
                int steps = 0;
                // Jaise he hole milta hai to break karna hai to isisliye !(r == er && c == ec)
                // check add kiya.

                // !(r == er && c == ec) check can also be written as (r != er) || (c != ec)
                while (r >= 0 && c >= 0 && r < n && c < m && maze[r][c] == 0 && !(r == er && c == ec))
                    r += dir[d][0];
                    c += dir[d][1];
                    steps++;
                }

                // Agar hole milta hai to use sidhe queue mai add karna hai
                if (!(r == er && c == ec)) { // Agar hole nhi hai to he mai minus karunga
                    r -= dir[d][0];
                    c -= dir[d][1];
                    steps--;
                }

                if (totalSteps + steps > dis[r][c]) // equal to check not added since we want the lexog
                                                    // string first for same steps to the hole
                    continue;
```

```
                que.add(new stringPair(r, c, totalSteps + steps, str + dirS[d]));
                dis[r][c] = totalSteps + steps;
            }
        }
    }
    return "impossible";
}
```

Normal Visited Method :

```
class Solution {
    class Point implements Comparable<Point> {
        int x;
        int y;
        int dis;  // distance from ball
        String path;  // directions from ball
        Point(int x, int y, int dis, String path) {
            this.x = x;
            this.y = y;
            this.dis = dis;
            this.path = path;
        }
        // if both ways have shortest distance, they should be ordered lexicographically
        public int compareTo(Point point) {
            return this.dis == point.dis ? this.path.compareTo(point.path) : this.dis - point.dis;
        }
    }


    public String findShortestWay(int[][] maze, int[] ball, int[] hole) {
        int rows = maze.length, cols = maze[0].length;
        boolean[][] visited = new boolean[rows][cols];

        PriorityQueue<Point> pq = new PriorityQueue<>();
        pq.offer(new Point(ball[0], ball[1], 0, ""));

        // arrays used for exploring 4 directions from a point
        char[] dstr = {'u', 'd', 'l', 'r'};
        int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

        while (!pq.isEmpty()) {
            Point position = pq.poll();
            if (position.x == hole[0] && position.y == hole[1]) {
                return position.path;
            }

            for (int i = 0; i < dirs.length; i++) {
                int x = position.x;
                int y = position.y;
                int dis = position.dis;
                String path = position.path;
```

visited[position.x][position.y] = true; ==> isko for loop se pehle bhi mark kar sakte hain. Same he baat hai

```
                // Explore current direction until hitting a wall or the hole
                while (x >= 0 && x < rows && y >= 0 && y < cols && maze[x][y] == 0 && (x != hole[0] || y != hole[1])) {
                    x += dirs[i][0];
                    y += dirs[i][1];
                    dis += 1;
                }
                // if the ball didn't encounter the hole, we need to roll back one step
                // to get the right position that the ball can reach (in range)
                if (x != hole[0] || y != hole[1]) {
                    x -= dirs[i][0];
                    y -= dirs[i][1];
                    dis -= 1;
                }
                if (!visited[x][y]) {
```

```
                visited[position.x][position.y] = true;
                pq.offer(new Point(x, y, dis, path + dstr[i]));
            }
        }
    }

    return "impossible";
}
```

```
// Bhaiya's Method :

// Yahan pe jo update kar rehe hain dis[r][c] with pointPair wo karne ki
// jaroorat nhi hai waise kyunki jo humne class mai comparable likha hai wo sab
// handle kar lega agar do string aise aayi jinke steps equal hue to

// Aur hum yahan pe pura bfs run hone de rahe hain jab answer mil gaya tab bhi
// joki bilkul bhi requierd nhi hai.

// ? Isko Simple visited se bhi kar sakte hain
```

```
public static class pointPair implements Comparable<pointPair> {
    int r = 0, c = 0, steps = 0;
    String psf = "";

    pointPair(int r, int c, int steps, String psf) {
        this.r = r;
        this.c = c;
        this.steps = steps;
        this.psf = psf;
    }

    @Override
    public int compareTo(pointPair o) {
        if (this.steps != o.steps)
            return this.steps - o.steps;
        else
            return this.psf.compareTo(o.psf);
    }
}
```

```
public String findShortestWay_(int[][] maze, int[] start, int[] destination) {
    int n = maze.length, m = maze[0].length, sr = start[0], sc = start[1], er = destination[0], ec = des
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    String[] dirS = { "d", "u", "r", "l" };
    pointPair[][] dis = new pointPair[n][m];
    for (int i = 0; i < n * m; i++) // Good way to add some data in 2-D array
        dis[i / m][i % m] = new pointPair(i / m, i % m, (int) 1e8, psf: "");

    PriorityQueue<pointPair> que = new PriorityQueue<>();
    pointPair src = new pointPair(sr, sc, steps: 0, psf: "");

    que.add(src);
    dis[sr][sc] = src;

    while (que.size() != 0) {
        pointPair p = que.remove();
        for (int i = 0; i < 4; i++) {
            int[] d = dir[i];

            int r = p.r, c = p.c, steps = p.steps;
            while (r >= 0 && c >= 0 && r < n && c < m && maze[r][c] == 0 && !(r == er && c == ec)) { //
                                                                                                    //
                                                                                                    //
                                                                                                    //
                r += d[0];
```

```
                c += d[1];
                steps++;
            }
```

```
            if (!(r == er && c == ec)) {
                r -= d[0];
                c -= d[1];
                steps--;
            }

            pointPair np = new pointPair(r, c, steps, p.psf + dirS[i]);
            if (steps > dis[r][c].steps || dis[r][c].compareTo(np) <= 0)
                continue;
            // This dis[r][c].compareTo(np) <= 0 check added since hume dis[r][c] ko tabhi
            // ` update karna hai ya to steps kam ho ya agar steps equal ho aur jo string aa
            // rahi hai wo pehle wali string se lexographically achi ho.

            // For example agar pehle dis[r][c] ki string thi "ul" aur ab aayi "lul" to
            // {"ul" - "lul" > 0} by this - other concept. To ab update karna hai kyunki
            // "lul" achi hai lexographically "ul" se.

            que.add(np);
            dis[r][c] = np;
        }
    }

    pointPair ans = dis[er][ec]; // end mai answer return kara
    return ans.steps != (int) 1e8 ? ans.psf : "impossible";
}
```

56. Chocolate Journey

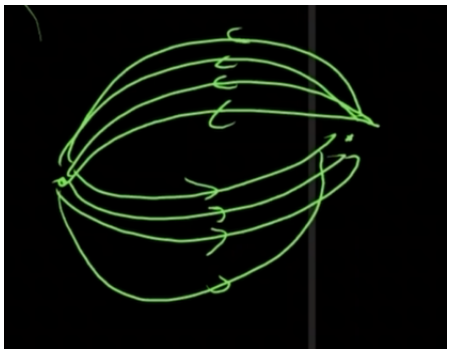
Problem

You live in the city **B**. Your friend is living in the city **A**. You need a special chocolate xyz. The chocolate is not available in your city and is available only at **k** cities. There are **N** cities in the country and **M** bi-directional roads between the cities and length of each of these bi-directional roads is given. The chocolate is preserved in cold containers and can stay for infinite time if it is preserved in those containers. If it is once taken out of the cold container, It expires in **x** units of time and you cannot put it back into the cold container to make it available for the infinite time.

It takes *1* unit of time to cover *1* unit of distance.

What is the **minimum** amount of time your friend needs to reach you with the chocolate?
If it is not possible to reach you with the chocolate, print "-1" (without quotes).

Note : There are no self loops and no multiple edges.



Above is a Multiple edges graph

Input Format:

Input Format:
The first line contains four integers **N** (number of cities), **M** (number of bidirectional roads), **k** (number of cities where chocolate is available), **x** (the expiry time).

The next line contains **k** space separated integers denoting cities where chocolate is available(Assume cities are indexed from **1** to **N**).

The next **M** lines contains *3* integers **u**, **v**, **d** in each line indicating that there is a path between **u** and **v** and having a length of **d**.

The last line contains *2* space separated integers **A** and **B** denoting your friend's and your city respectively.

Output Format:
Print the minimum amount of time taken to reach the city **B** from **A** with the chocolate. If it is not possible, print "-1" (without quotes).

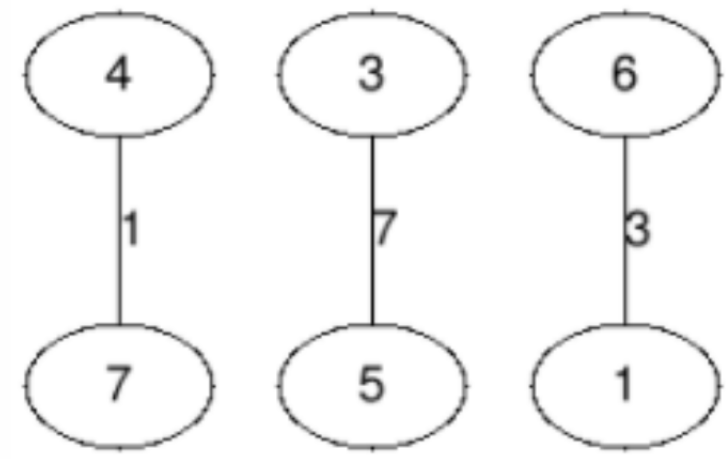
Input Constraints:
$1 \leq N \leq 10^5$
$1 \leq M \leq minimum(10^6, N * (N-1)/2)$
$1 \leq k \leq N-1$
$1 \leq x \leq N$
$1 \leq d \leq 500$
$1 \leq u, v, A, B \leq N$

Note: Use fast I/O techniques.

Sample Input	Sample Output
7 3 1 6 1 4 7 1 3 5 7 6 1 3 6 2	-1

Time Limit: 2
Memory Limit: 512
Source Limit:

Explanation



So, here A is 6 and B is 2. As we don't have any path from 6 -> 2, our answer is -1.

```
// B ==> me , A == > Friend
// Chocolate available only at k cities
// N cities in the country
// M bi-directional roads
// Chocolate expires in x units of time
// 1 Unit time covers 1 distance
// Find minimum time friend needs to reach you with chocolate.
// So A has to reach me.

// input == > N, M, k, x
```

```
// input ==> cities having chocolate
// input ==> M bidirectioal roads with (u,v,w)
// input == > at last A and B.
```

```
public static void main(String[] args) throws IOException {
    chocolateJourney();
}

// Simple tha, do bar dijikstra call karna tha

// To humne kiya ki A se sare vertex tak pahunchne ka min time nikala
// Same humne B ke liye nikala har vertex tak pahunchne ka min time

// Ab agar kisi vertex pe chocolate hai to humne ye check kiya ki A ka us
// chocolate tak ka pahunchne ka time kya tha aur B ko kitna time lga chocolate
// tak pahunchne mai. Dono ko add kiya condition check karke melting wali. Bas
// aise hume answer mila.

// ! Important point to note :

// # A chocolate wale vertex se B tak jaye ye B chocolate wale vertex tak aaye,
// # time dono ko same he lagega.
```

```
public static void dijikstra(int src, int[] dis, ArrayList<int[]>[] graph) { // Simple dijikstra lagaya

    // {v,wsf}
    PriorityQueue<int[]> que = new PriorityQueue<>((a, b) -> {
        return a[1] - b[1];
    });

    que.add(new int[] { src, 0 });
    dis[src] = 0;

    while (que.size() != 0) {
        int size = que.size();

        while (size-- > 0) {
            int[] rn = que.remove();
            int u = rn[0], wsf = rn[1];

            if (wsf > dis[u]) // Equal to wali condition nhi lagayi kyunki aisa ho sakta hai ki A ko cho
                              // wali jahah tak aane mai same time laga per B ko wahan aane mai alag alg
                              // lag sakta hai to humko sabko compare karna hai.
                continue;

            for (int[] e : graph[u]) {
                int v = e[0], wt = e[1];
                if (wsf + wt < dis[v]) {
                    dis[v] = wsf + wt;
                    que.add(new int[] { v, wsf + wt });
                }
            }
        }

    }
}
```

```
public static void chocolateJourney() throws IOException {
    // Agar koi mera function exception throw karta hai to mai bhi exception throw
    // karunga
    Reader scn = new Reader();
    int n = scn.nextInt();
    int m = scn.nextInt();
    int k = scn.nextInt();
    int x = scn.nextInt();

    boolean[] chocolate = new boolean[n + 1];
```

```
    for (int i = 0; i < k; i++) // Marked the chocolate vertex true.
        chocolate[scn.nextInt()] = true;

    ArrayList<int[]>[] graph = new ArrayList[n + 1];
    for (int i = 0; i <= n; i++) {
        graph[i] = new ArrayList<>();
    }

    while (m-- > 0) { // ? Created graph
        int u = scn.nextInt(), v = scn.nextInt(), w = scn.nextInt();
        graph[u].add(new int[] { v, w });
        graph[v].add(new int[] { u, w });
    }

    int A = scn.nextInt(), B = scn.nextInt();
```

```
// Dono ke distance array ko fill karaya taki min distance mil jaye har vertex
// tak.
int[] disA = new int[n + 1];
Arrays.fill(disA, (int) 1e9);
dijikstra(A, disA, graph);

int[] disB = new int[n + 1];
Arrays.fill(disB, (int) 1e9);
dijikstra(B, disB, graph);

int ans = (int) 1e9;
for (int i = 0; i <= n; i++) {
    if (chocolate[i]) {
        // disA[i] != (int) 1e9 ==> Agar manle A ko chocolate mili he nhi, to wo kabhi
        // ` bhi B tak chocolate leke jaa he nhi payega

        // disB[i] < x ==> B ka chocolate wali vertex tak chocolate ko melt hone se
        // pehle pahunchna hoga. Same concept jaba A ko chocolate mili to A B tak jaye
        // ya B A tak aaye same dono ko same he time lagega

        if (disA[i] != (int) 1e9 && disB[i] < x) {
            ans = Math.min(ans, disA[i] + disB[i]);
        }
    }
}

if (ans == (int) 1e9)
    System.out.println(x: "-1");
else
    System.out.print(ans);
```

```
static class Reader { // # class used for fast input output in java
    final private int BUFFER_SIZE = 1 << 16;
    private DataInputStream din;
    private byte[] buffer;
    private int bufferPointer, bytesRead;

    public Reader() {
        din = new DataInputStream(System.in);
        buffer = new byte[BUFFER_SIZE];
        bufferPointer = bytesRead = 0;
    }

    public Reader(String file_name) throws IOException {
        din = new DataInputStream(
                new FileInputStream(file_name));
        buffer = new byte[BUFFER_SIZE];
        bufferPointer = bytesRead = 0;
    }

    public String readLine() throws IOException {
        byte[] buf = new byte[64]; // line length
        int cnt = 0, c;
        while ((c = read()) != -1) {
            if (c == '\n') {
```

```
            if (cnt != 0) {
                break;
            } else {
                continue;
            }
        }
        buf[cnt++] = (byte) c;
    }
    return new String(buf, offset: 0, cnt);
}
```

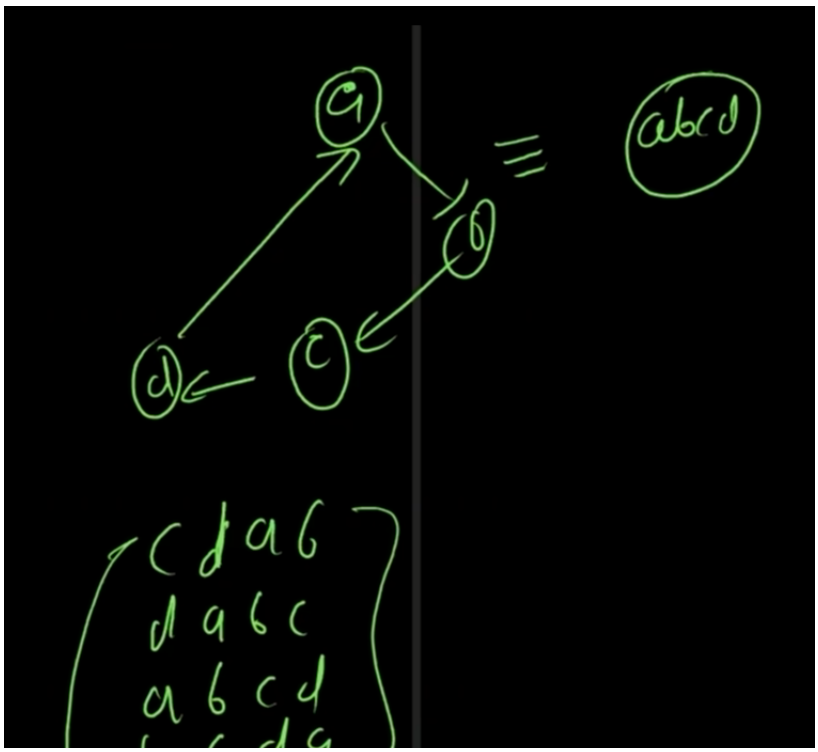
Used the Reader class for fast input/output. Copied from Gfg .

Important Point :

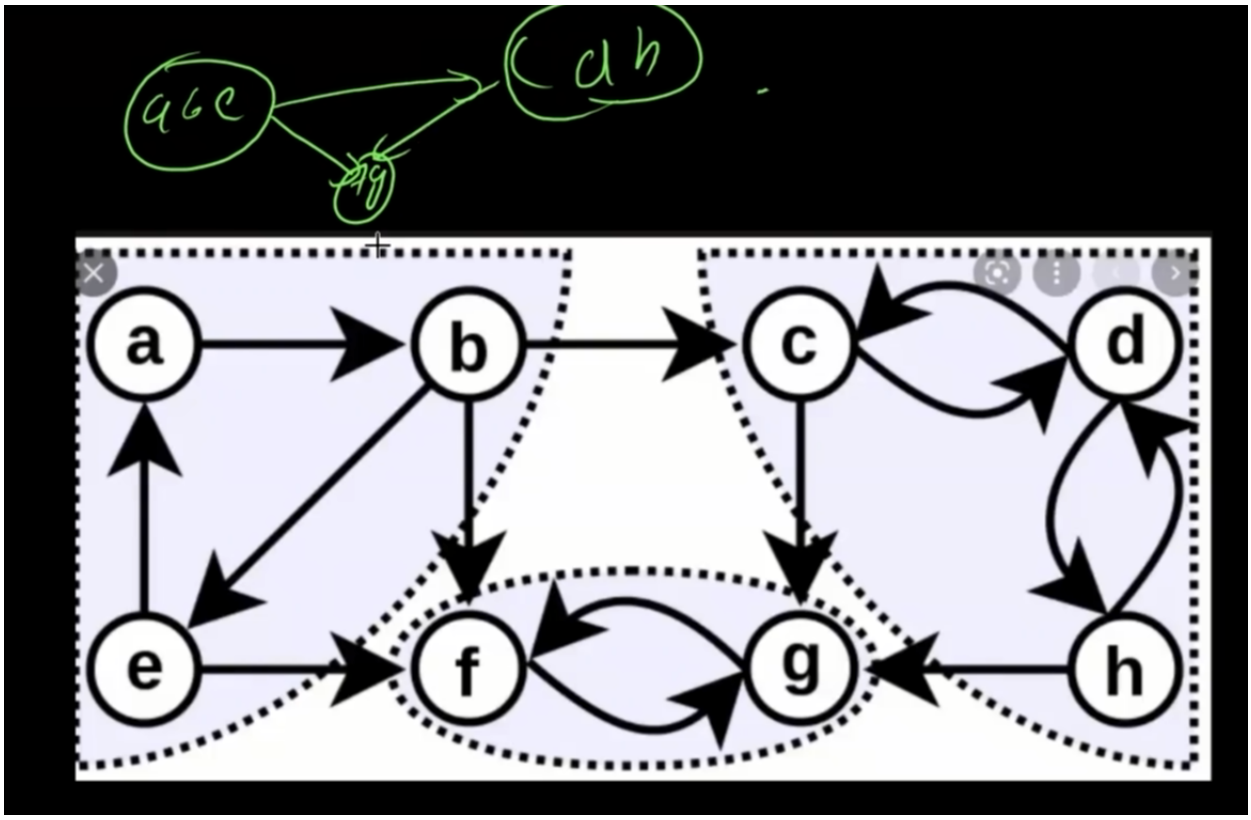
```
while (pq.size() != 0) {

    pair rn = pq.remove();
    int vtx = rn.src, wsf = rn.wsf;


    if (vis[vtx]) // wsf>dis[vtx] == > basically agar mera wsf dis[vtx] se bada hai to
                  // mujhe distance array ko update karne ki jaroorat he nhi hai to continue kar
                  // do. To matlab cycle hai
        continue;

    vis[vtx] = true;
    for (Edge e : graph[vtx]) {
        int v = e.v, w = e.w;
        if (!vis[v] && wsf + w < dis[v]) { // Agar mera distance wsf + w se jyada hai, iska matlab mujhe koi
                                           // aisa mila hai jo meretak current dis[v] se bhi lightest path ke
                                           // sath pahunch jayego, to maine tab apne distance ko mara update;
            dis[v] = wsf + w;
            par[v] = vtx; // Aur parent ko bhi sath mai update mar diya.
            pq.add(new pair(v, wsf + w));
        }
    }
}
```

Agar wsf > dis[vtx] :=> Isme equal to check nhi lagana hai. Uper continue wali statement mai. Kyunki wo basically uske aage wale jo kam hai unko rok dega joki shi nhi hai. Same like in chocolate journey.

57. Kosa Raju Algorithm



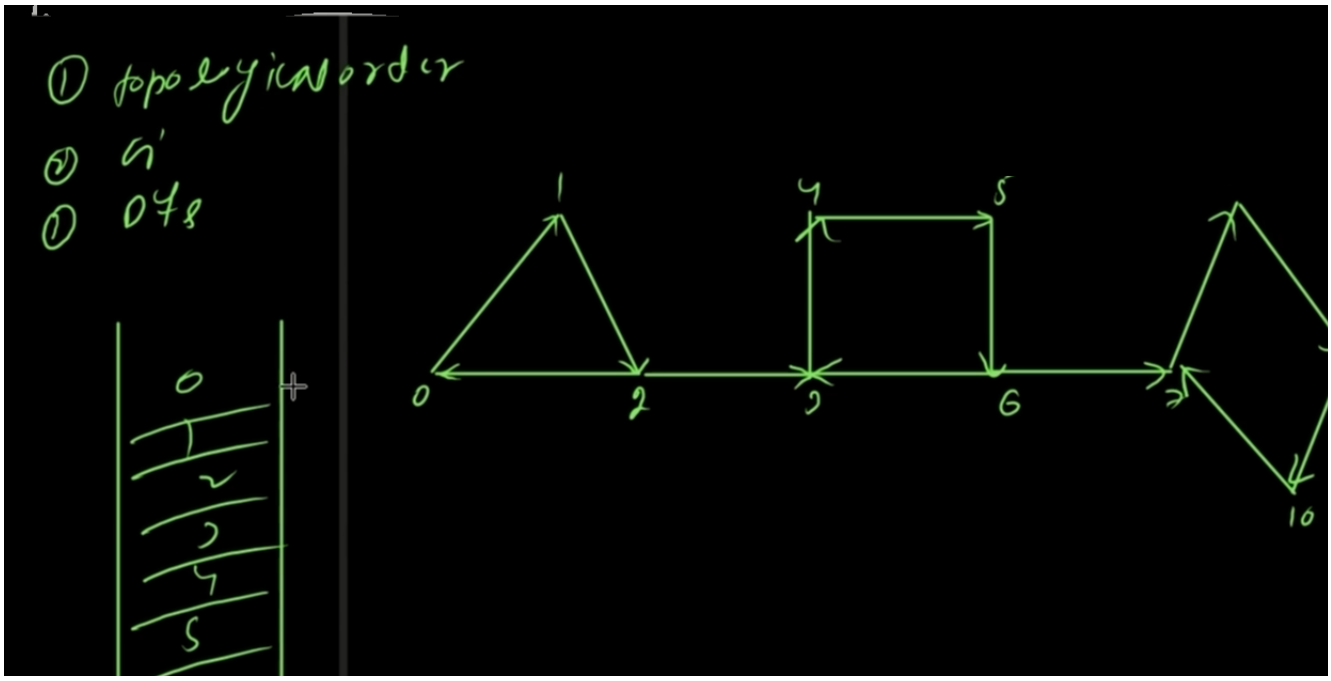
So here abcd is strongly connected since using any vertex I can reach all other vertices in the component.



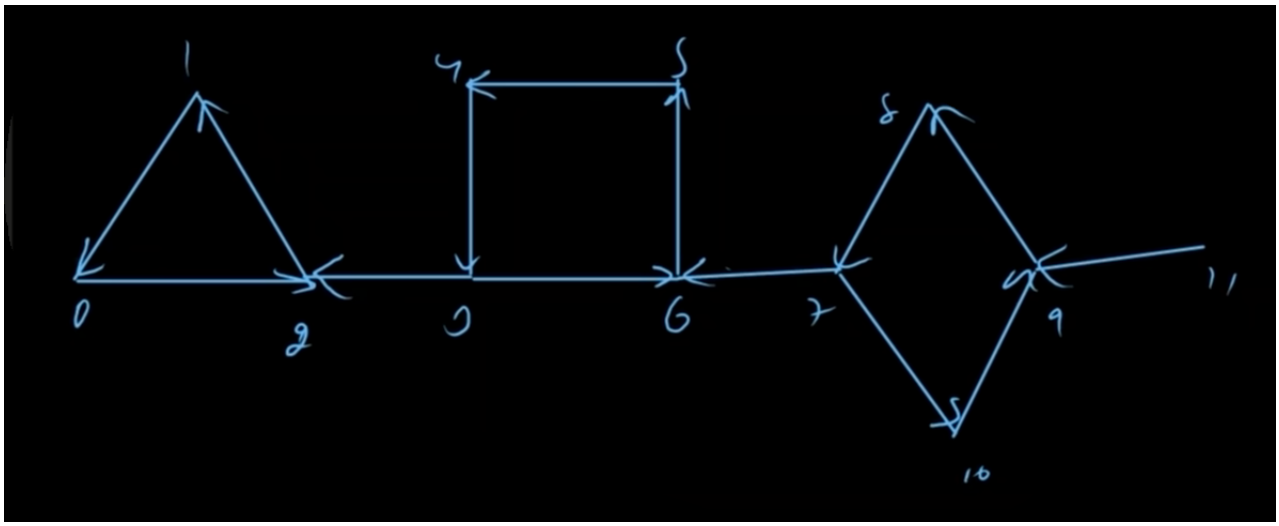
Converts the cyclic graph to acyclic graph.

So it basically has three steps :

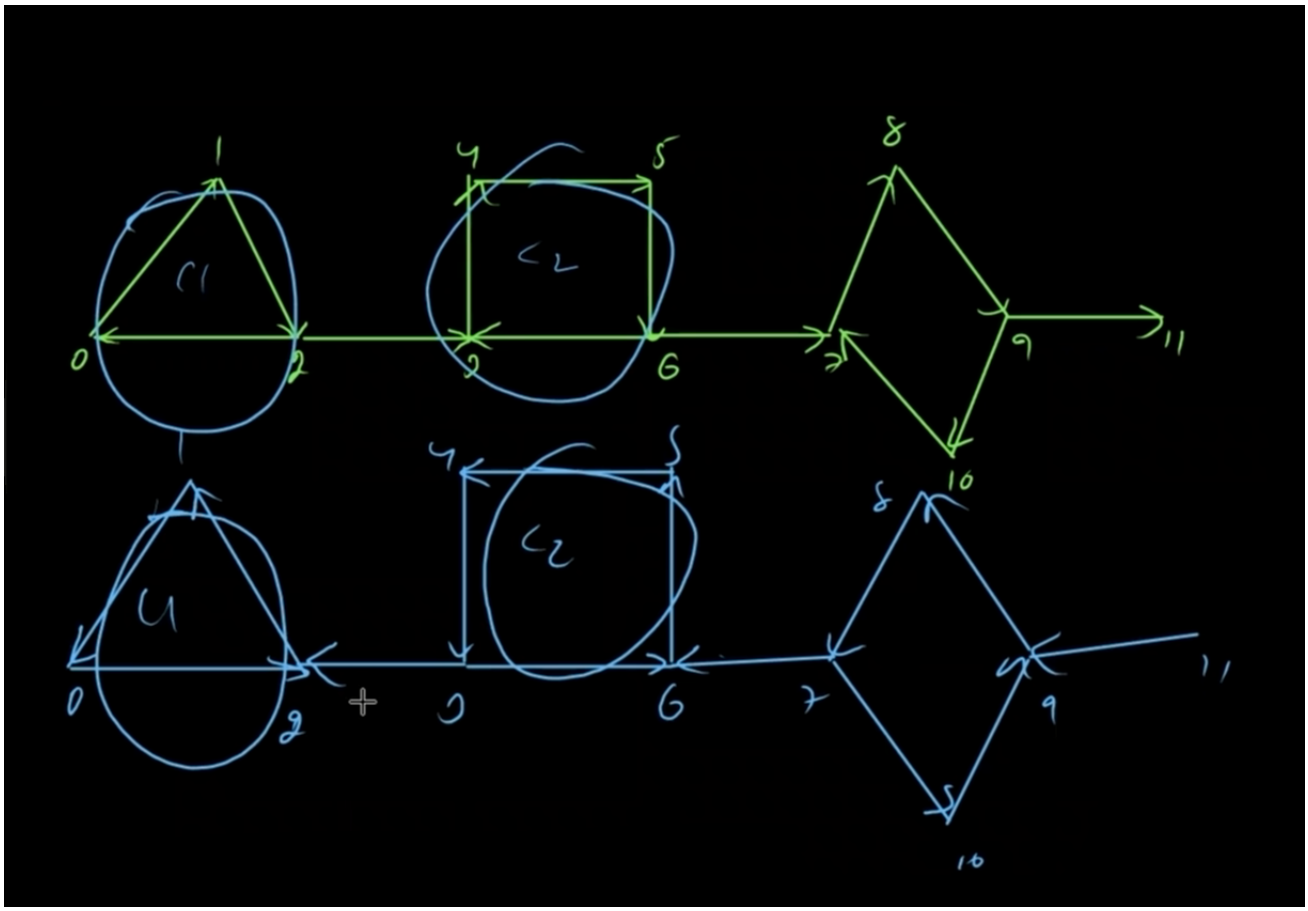
a. Find the topological order
b. Find the inverse of the graph
c. Run the Dfs



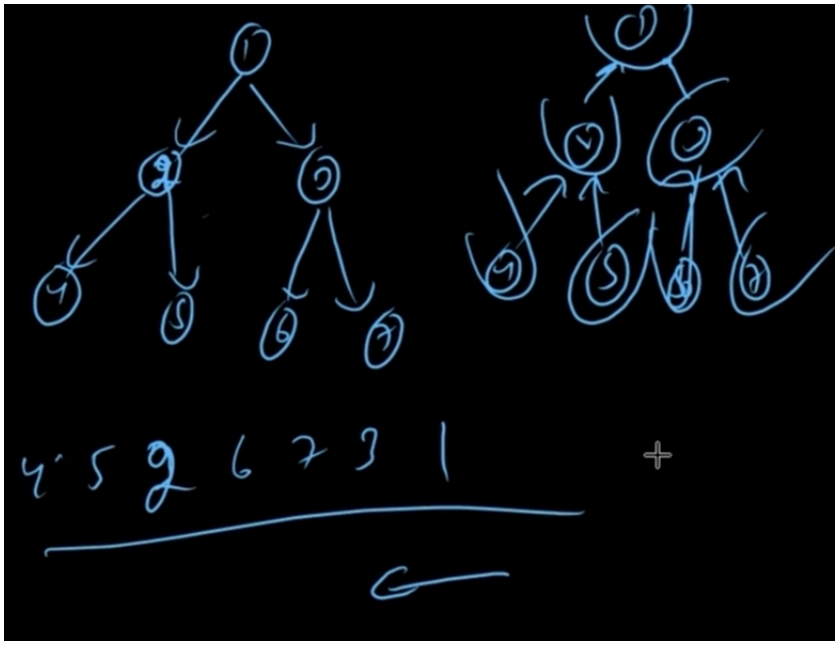
After the graph is reversed.



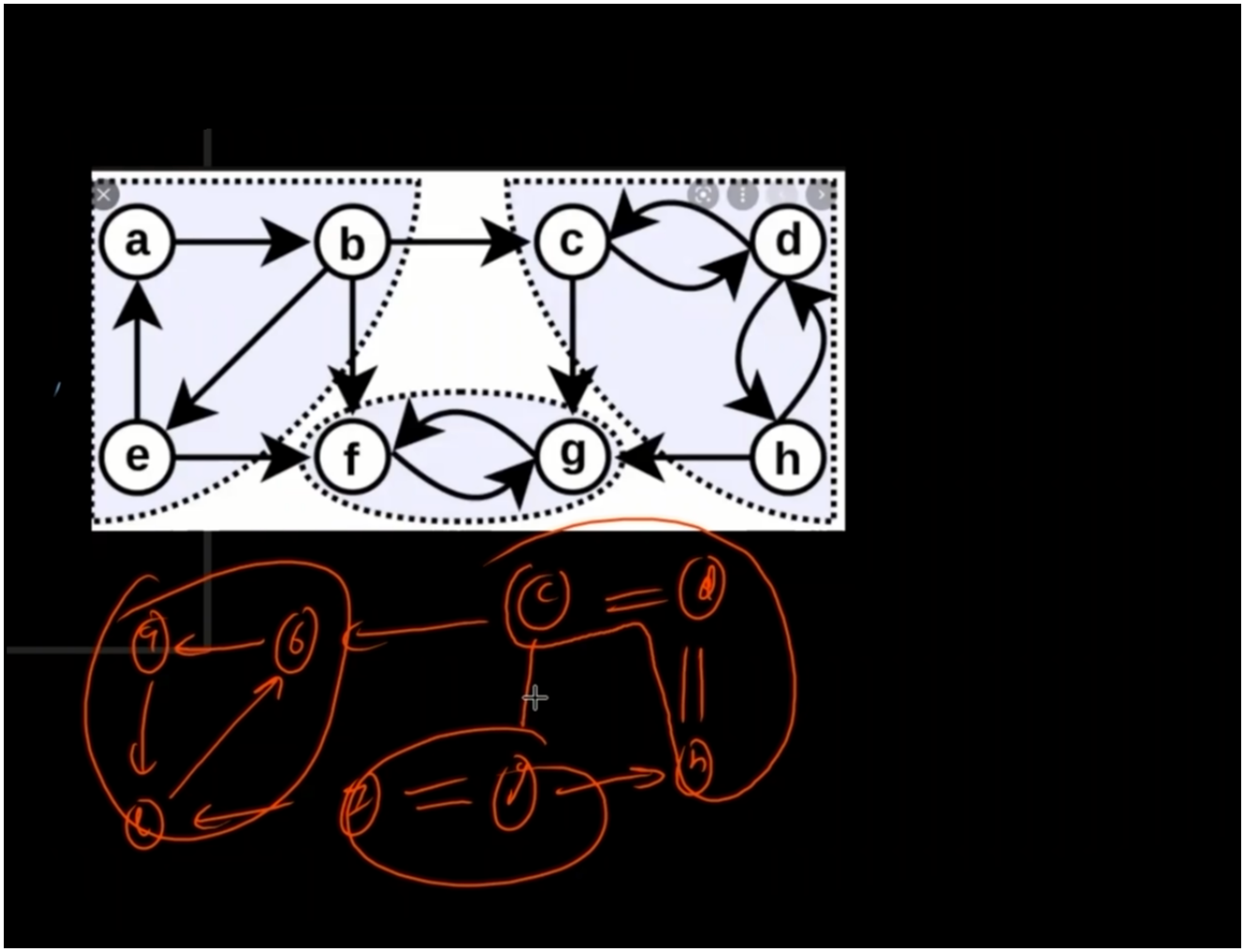
So jo cycle haiu use farak he nhi pada.



Pehle c1 se c2 reach ho sakta tha, per after inverse c2 se c1 reach nhi ho sakta (hum components ko mark karte hue chal rahe hain isiliye 3 2 ko reach nhi kar payega).



Like in case of tree shown above. If there are n nodes, then there will be n strongly connected components since each node will act as scc.




```
// b <================ KosaRaju Algo ======================>

// 1 It is used to find number of strongly connected components in a graph
```

```
// ? It is used to find number of strongly connected components in a graph.
// ? It converts cyclic graph into acyclic graph.

// Wo component jisme aap har vertex ka use karke har vertex tak reach kar paa
// rahe ho.

// # To ye Strongly connected components banate hain.

// A single vertex is a strongly connected component.

// 1. Find the topological order (With post Order)
// 2. Find the inverse of the graph
// 3. Run the Dfs on the topological order of the original graph on the inverse
// graph.
```

```
// ! Important Point :

// # Jab mai graph ko inverse karta hun na, bas cycle ko farak nhi padta, baki
// # sabko farak padta hai.

// To ab jab mai original graph ke topological order ko mante hue mai dfs run
// karunga inverse wale graph mai to jo single roads hongi wo rasta rok denge
// ` baki aur cycles ko access karne ka.

// ! Important point :

// # When we studied topological order,we found out that when there is cycle
// # present, dfs does not give the right answer. So why used here ?

// Humko dfs traversal wali cheez to shi bata raha hota hai. Matlab isse besahq
// topological order galat aayega, per jab humne graph ko reverse kiya, to cycle
// ko koi farak nhi pada.
```

```
// ! Steps for KosaRaju algo

// 1. Find the topological order from postorder.
// 2. Inverse the graph
// 3. Using that topological order, again call dfs and get the scc count.
```

```
// # scc : - Strongly Connected Components
public static void dfs_01(int src, ArrayList<Edge>[] graph, boolean[] vis, LinkedList<Integer> st) { //
                                                                                                    //
                                                                                                    //
                                                                                                    //
    vis[src] = true;
    for (Edge e : graph[src]) {
        if (!vis[e.v])
            dfs_01(e.v, graph, vis, st);
    }

    st.addFirst(src);
}

public static void dfs_02(int src, ArrayList<Edge>[] graph, boolean[] vis) { // Just to traverse to each
                                                                             // component
    vis[src] = true;
    for (Edge e : graph[src]) {
        if (!vis[e.v])
            dfs_02(e.v, graph, vis);
    }
}
```

```
public static void KosaRajuAlgo(int n, ArrayList<Edge>[] graph) {
    boolean[] vis = new boolean[n];
    LinkedList<Integer> st = new LinkedList<>(); // Created a stack to store the topological order.
```

```
    for (int i = 0; i < n; i++) {
        if (!vis[i])
            dfs_01(i, graph, vis, st);
    }

    // Inverse Of a graph
    ArrayList<Edge>[] nGraph = new ArrayList[n]; // Inverse the graph so that one cycle may not reach ot
    for (int i = 0; i < n; i++) {
        for (Edge e : graph[i]) {
            int v = e.v, wt = e.w;
            nGraph[v].add(new Edge(i, wt));
        }
    }

    int scc = 0; // To maintain scc count
    vis = new boolean[n]; // Created a new visited
    while (st.size() != 0) { // So now we will start traversing vertex as stored in stack. (The topologi
                             // order)
        int vtx = st.removeFirst();
        if (!vis[vtx]) {
            dfs_02(vtx, nGraph, vis);
            scc++;
        }
    }
}
```

58. Mother Vertex

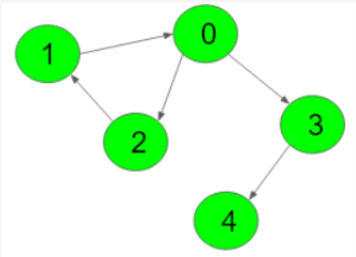
Mother Vertex

Easy Accuracy: **49.76%** Submissions: **24736** Points: **2**

Given a Directed Graph, find a Mother Vertex in the Graph (if present).
A Mother Vertex is a vertex through which we can reach all the other vertices of the Graph.

Example 1:

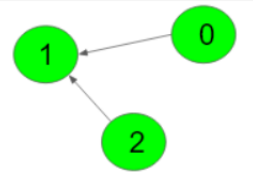
Input:



Output: 0
Explanation: According to the given edges, all nodes can be reaced from nodes from 0, 1 and 2. But, since 0 is minimum among 0,1 and 3, so 0 is the output.

Example 2:

Input:



Output: -1

Explanation: According to the given edges, no vertices are there from where we can reach all vertices. So, output is -1.

Your Task:

You don't need to read or print anything. Your task is to complete the function **findMotherVertex()** which takes V denoting the number of vertices and adjacency list as imput parameter and returns the verticex from through which we can traverse all other vertices of the graph. If there is more than one possible nodes then returns the node with minimum value.If not possible returns -1.

Expected Time Complexity: O(V + E)
Expected Space Compelxity: O(V)

Constraints:

$1 \leq V \leq 500$

```
// b <================== Mother Vertex ==================>
// https://practice.geeksforgeeks.org/problems/mother-vertex/1

// To maine wo last vertex nikala jisse graph traverse ho raha hai pura.

// Kyunki mai initially 0 se start kiya hai aur hum sequentially chal rahe hain
// to ye ensure karta hai ki mujhe vertex min he milega, agar multiple mother
// vertex hain to.

// To bas jaise he last vertex mili ab ye confirm karne ke liye ki wo mother
// vertex hai, pure graph ko dubara se traverse kareke size nikala. agar ab size
// equal aata hain total number of vertex ke, to matlab vertex mother vertex hai
```

```
public static void dfs_motherVertex_01(int src, int V, ArrayList<ArrayList<Integer>> graph, boolean[] v

    vis[src] = true;
    for (int vtx : graph.get(src)) {
        if (!vis[vtx])
            dfs_motherVertex_01(vtx, V, graph, vis);
    }
}

public static int dfs_motherVertex_02(int src, int V, ArrayList<ArrayList<Integer>> graph, boolean[] vi

    vis[src] = true;
    int size = 0;
    for (int vtx : graph.get(src)) {
        if (!vis[vtx])
            size += dfs_motherVertex_02(vtx, V, graph, vis);
    }

    return size + 1;
```

```
    }
```

```
public static int findMotherVertex(int V, ArrayList<ArrayList<Integer>> graph) {
    // Code here

    boolean[] vis = new boolean[V];
    int lastVertex = -1;
    for (int i = 0; i < V; i++) {
        if (!vis[i]) {
            dfs_motherVertex_01(i, V, graph, vis);
            lastVertex = i; // Since we are starting from 0, this ensures that we have smallest element
                            // the last that may traverse the whole graph
        }
    }

    vis = new boolean[V];
    int size = dfs_motherVertex_02(lastVertex, V, graph, vis);

    return size == V ? lastVertex : -1;
}
```

59. Path With Minimum Effort

1631. Path With Minimum Effort

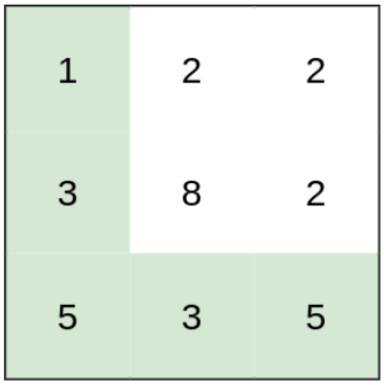
Medium 3622 147 Add to List Share

You are a hiker preparing for an upcoming hike. You are given `heights`, a 2D array of size `rows x columns`, where `heights[row][col]` represents the height of cell `(row, col)`. You are situated in the top-left cell, `(0, 0)`, and you hope to travel to the bottom-right cell, `(rows-1, columns-1)` (i.e., **0-indexed**). You can move **up**, **down**, **left**, or **right**, and you wish to find a route that requires the minimum **effort**.

A route's **effort** is the **maximum absolute difference** in heights between two consecutive cells of the route.

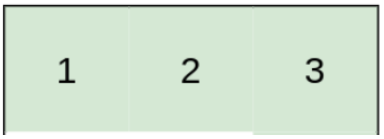
Return *the minimum **effort** required to travel from the top-left cell to the bottom-right cell.*

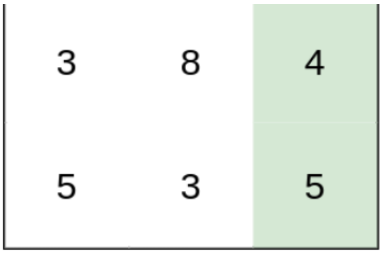
Example 1:




```
Input: heights = [[1,2,2],[3,8,2],[5,3,5]]
Output: 2
Explanation: The route of [1,3,5,3,5] has a maximum absolute difference of 2 in consecutive
cells.
This is better than the route of [1,2,2,2,5], where the maximum absolute difference is 3.
```

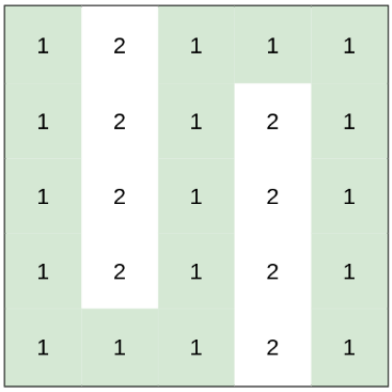
Example 2:






```
Input: heights = [[1,2,3],[3,8,4],[5,3,5]]
Output: 1
Explanation: The route of [1,2,3,4,5] has a maximum absolute difference of 1 in consecutive
cells, which is better than route [1,3,5,3,5].
```

Example 3:




```
Input: heights = [[1,2,1,1,1],[1,2,1,2,1],[1,2,1,2,1],[1,2,1,2,1],[1,1,1,2,1]]
Output: 0
Explanation: This route does not require any effort.
```

Constraints:

- `rows == heights.length`
- `columns == heights[i].length`
- `1 <= rows, columns <= 100`
- `1 <= heights[i][j] <= 10^6`

```java
// b <====================== Path With Minimum Effort ======================>
// https://leetcode.com/problems/path-with-minimum-effort/

// Simple dijikstara lagaya.
public static class pathPair implements Comparable<pathPair> {
    int r = 0, c = 0, absValue = 0;

    pathPair(int r, int c, int absValue) {
        this.r = r;
        this.c = c;
        this.absValue = absValue;
    }

    @Override
    public int compareTo(pathPair p) {
        return this.absValue - p.absValue;
    }
}
```

```java
public int minimumEffortPath(int[][] heights) { // b (90% effcient)

    int n = heights.length, m = heights[0].length;
    int er = n - 1, ec = m - 1;
```

```
        PriorityQueue<pathPair> que = new PriorityQueue<>();
        int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

        que.add(new pathPair(r: 0, c: 0, absValue: 0)); // Initial jo absolute value hogi wo to 0 he hogi na
        boolean[][] vis = new boolean[n][m];

        while (que.size() != 0) {
            int size = que.size();

            while (size-- > 0) {
                pathPair rn = que.remove();
                int sr = rn.r, sc = rn.c, absValue = rn.absValue;

                if (sr == er && sc == ec)
                    return absValue;

                if (vis[sr][sc])
                    continue;

                vis[sr][sc] = true;
```

```
                for (int d = 0; d < dir.length; d++) {
                    int r = sr + dir[d][0];
                    int c = sc + dir[d][1];

                    if (r >= 0 && c >= 0 && r < n && c < m) {
                        if (!vis[r][c]) {
                            que.add(new pathPair(r, c, Math.max(absValue, Math.abs(heights[sr][sc] - height
                        }
                    }
                }

            }
        }
        return -1;
    }
```

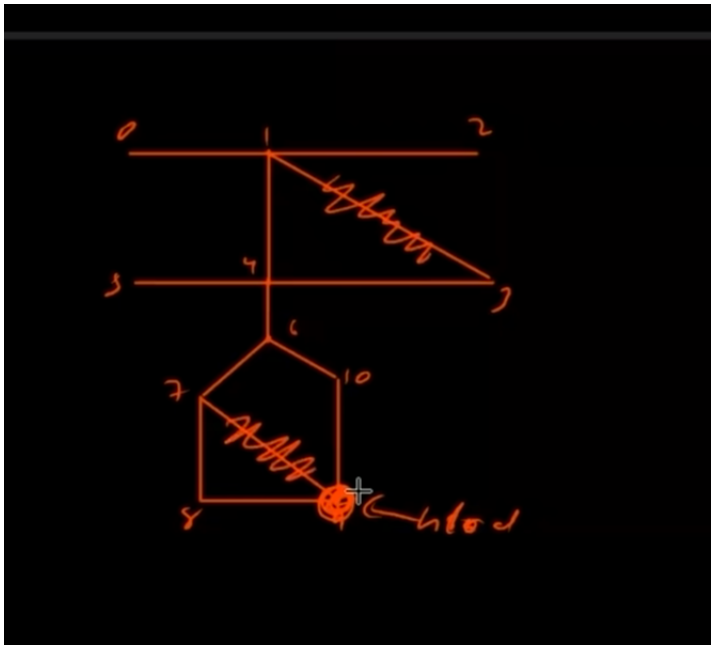
```
    // b Method 2 :

    // ! Using the distance array concept (More efficient, 99.99 %) since it escapes
    // ! few visits to vertexs

    public int minimumEffortPath_(int[][] heights) {
        int n = heights.length, m = heights[0].length;
        int er = n - 1, ec = m - 1;

        PriorityQueue<pathPair> que = new PriorityQueue<>();
        int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

        que.add(new pathPair(r: 0, c: 0, absValue: 0)); // Initial jo absolute value hogi wo to 0 he hogi na
        int[][] dis = new int[n][m];
        for (int i = 0; i < n * m; i++)
            dis[i / m][i % m] = (int) 1e9;
        dis[0][0] = 0;

        while (que.size() != 0) {
            int size = que.size();

            while (size-- > 0) {
                pathPair rn = que.remove();
                int sr = rn.r, sc = rn.c, absValue = rn.absValue;

                if (sr == er && sc == ec)
                    return absValue;
```

```
                if (absValue > dis[sr][sc])
                    continue;
```

```
            for (int d = 0; d < dir.length; d++) {
                int r = sr + dir[d][0];
                int c = sc + dir[d][1];
                if (r >= 0 && c >= 0 && r < n && c < m) {
                    int newAbsValue = Math.max(absValue, Math.abs(heights[sr][sc] - heights[r][c]));
                    if (newAbsValue < dis[r][c]) {
                        que.add(new pathPair(r, c, newAbsValue));
                        dis[r][c] = newAbsValue;
                    }
                }
            }
        }
    }
    return -1;
}
```

60. Articulation Point

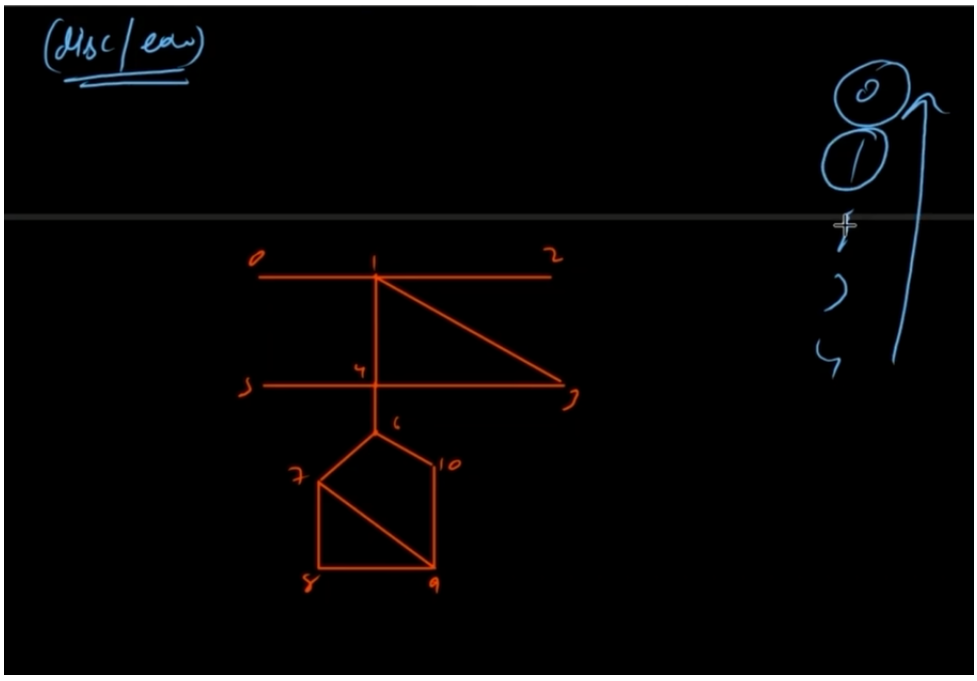
```
// b <============= Articulation Point ======================>

// # Koi aisa point jisko nikalne se ya hatane se mere graph ke number of
// # components increase karte hain, use articulation point bolte hain.

// Ek graph mai multiple articulation point ho sakta hain.

// Ek aisa edge jisko nikalke number of components increase ho jate hain to use
// articulation edge bolte hain
```

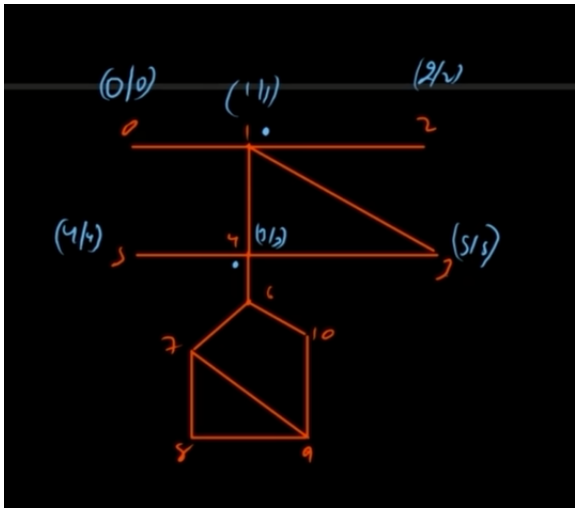


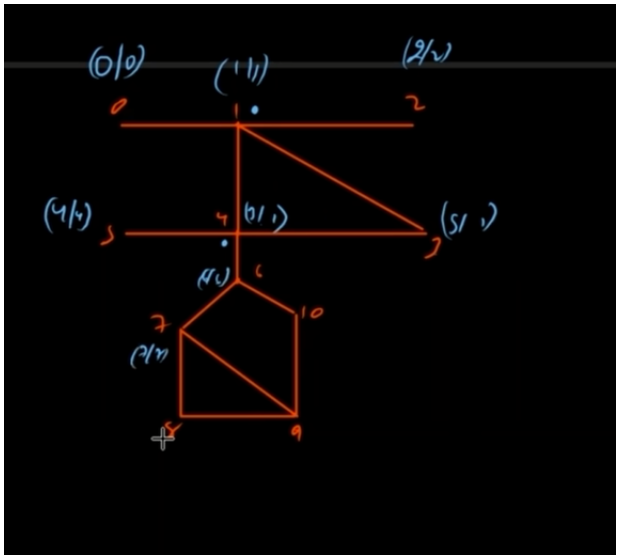

```
// ! How the question can be formed ?

// 1. Jaise ki manle bahut sari cities aapsa mai connected hain in a country aur
// war ho gayi. To war mai kisi city ke uper bomb gira diya. Ab meri country mai
// ek headquater city hai jahan se food supply hota hai. To kya hai bomb wali
// city ko chodke us country ke har ek city tak food pahuncha paunga?

// 2. Uper wale question aise bhi frame ho sakta hai ki two connecting cities
// ke pull pe bomb gira. Ab batao ki food har ek city tak pahunch payega.
```

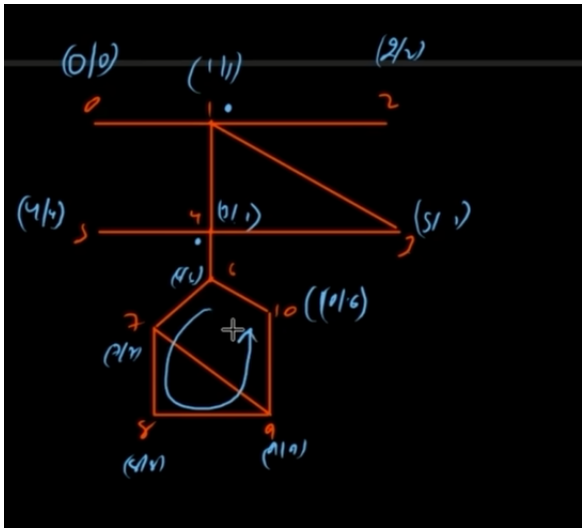
```
// 3. Manlo bahut sare planets hain aur unka aapas mai communication ho raha
// hai. Ab ek metriod aaya aur kisi planet se aake takraya aur destroy hua. To
// kya abhi bhi sare planets ek dusre se communicate kar payenge???
```

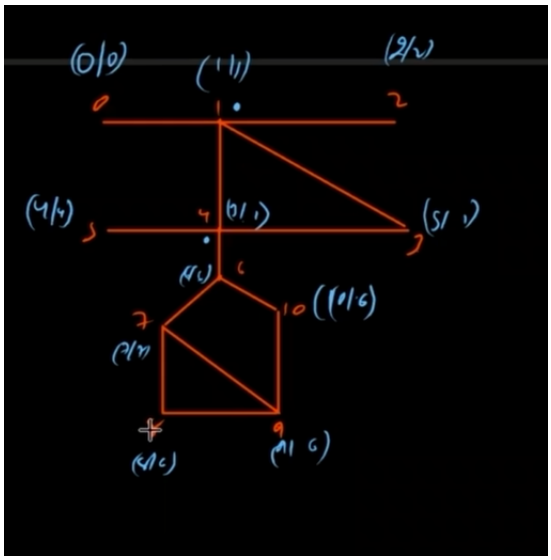
```
// ! Important Points :

// # Discovery Time : Jab maine chalna start kiya, to kaunse vertex mai kis time
// # mai discover kiya

// ===> Jiski Discovery Pehle, wo jyada powerful
// ===> Jiski Discovery Badme, wo jyada weak

// # Low Time : Mai kis powerful person ko janta hun. (Tum apne se pehle kisi
// # powerfull person ko jante ho)
// ===> Iske bare mai hum tabhi sochenge jab hum backtrack karenge.

// ? Agar koi vertex mera parent nhi hai aur wo already visited hai aur mera
// ? neighbour hai to wo heiarchy mai mujse pehle/uper raha hoga.
```

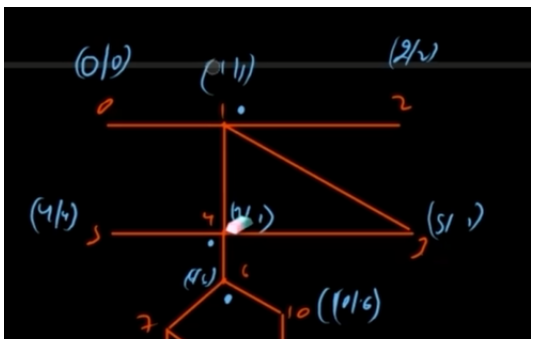


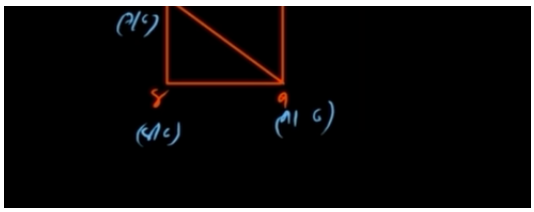


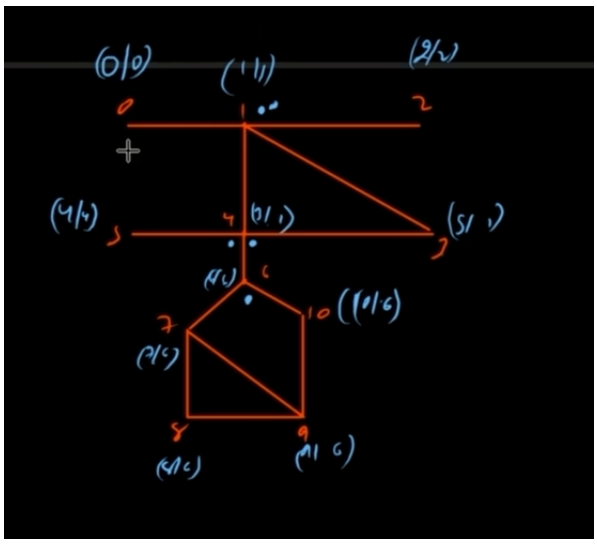

(0/0)
(1/1)
(2/2)
0
2
(4/4)
4
(3/3)
(5/5)
5
3
(4/4)
6
7
10
(4/4)
8
9


(0/0)
(1/1)
(2/2)
0
2
(4/4)
4
(3/3)
(5/5)
5
3
(4/4)
6
7
10
(10/6)
(4/4)
8
9
(10/10)

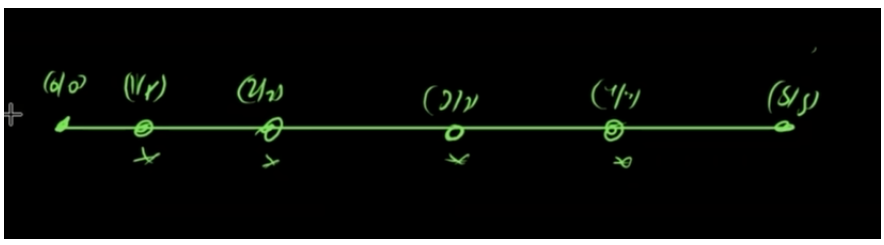

(0/0)
(1/1)
(2/2)
0
2
(4/4)
4
(3/3)
(5/5)
5
3
(4/4)
6
7
10
(10/6)
(4/4)
8
9
(10/10)


(0/0)
(1/1)
(2/2)
0
2
(4/4)
4
(3/3)
(5/5)
5
3
(4/4)
6
7
10
(10/6)

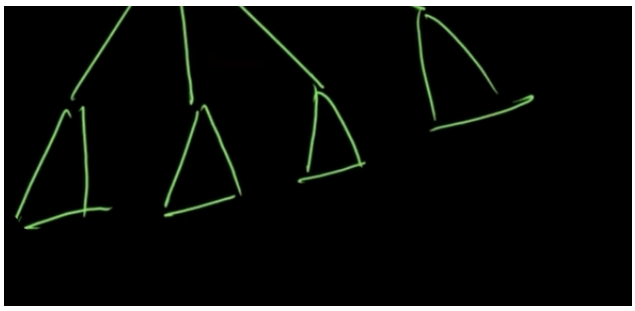


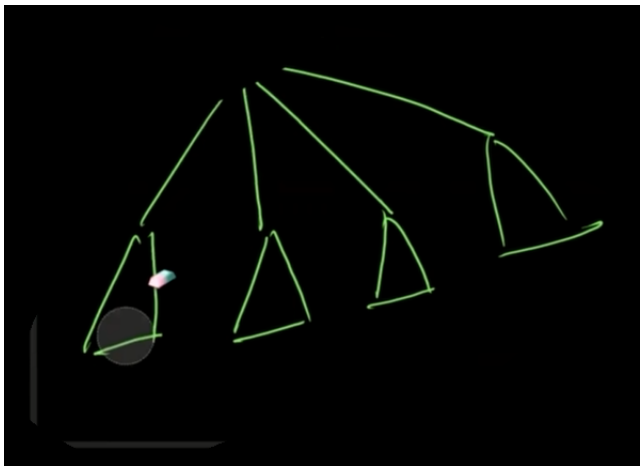


```
// # Jitne dots ek vertex pe honge(denoting that it is a articulation point
// # w.r.t other vertex), the no. of components formed by removing that vertex
// # will be (dots + 1). In Code we have maintained AP array to calculate the
// # dots.

// ! Important Point :
// This will be only true for all vertex except the root vertex.

// ? The normal dfs call make the root a articulation point even if the number
// ? of call from it is just one, which is wrong so therefore we have to handle
// ? root differently

// Agar root se number of calls is more than one, then it is a articulation
// point, otherwise not. For example take a example of nodes in a single line.
// If the root node are the endpoints node, then those points cannot be the
// articulation points since removing them, the number of components does not
// increase.

// If the number of calls is greater than one, than the total number of
// components formed by removing the root is equal to the number of dots.
// Therefore due to this, AP[root]=-1 in code to gernalize for everyNode in the
// graph.
```



For single line graph, the endpoints node cannnot be the articulation point.





For root If the calls are greater than one, then it can be a articulation point.

To handle this added Ap[i] -- in code

```
            if (par == -1)
                rootCalls++;

            dfs(e.v, src, graph);
            if (disc[src] <= low[e.v])
                AP[src]++;
            low[src] = Math.min(low[src], low[e.v]);

        } else if (e.v != par)
            low[src] = Math.min(low[src], disc[e.v]);
    }
}

public static void APB(int N, ArrayList<Edge>[] graph) {
    low = new int[N];
    disc = new int[N];
    AP = new int[N];
    vis = new boolean[N];

    for (int i = 0; i < N; i++) {
        if (!vis[i]) {
            dfs(i, -1, graph);
            if (rootCalls == 1)
                AP[i] = 0;
            else
                AP[i]--;
        }
    }
```

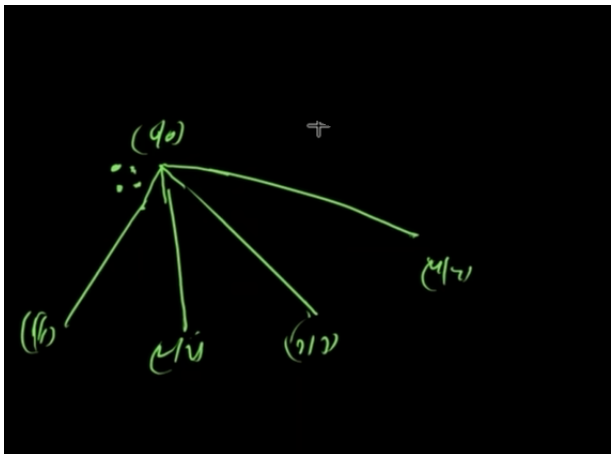
```
for (int i = 0; i < N; i++) {
    if (!vis[i]) {
        AP[i] = -1;
        dfs(i, -1, graph);
        // if (rootCalls == 1)
        // AP[i] = 0;
        // else
        // AP[i]--; // No of components is value + 1;
    }
```

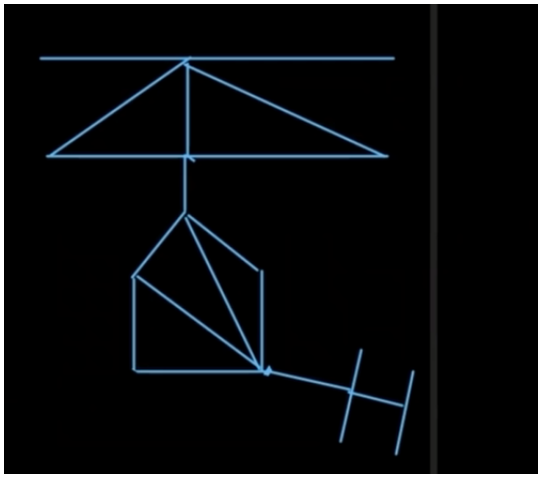
```
}
```

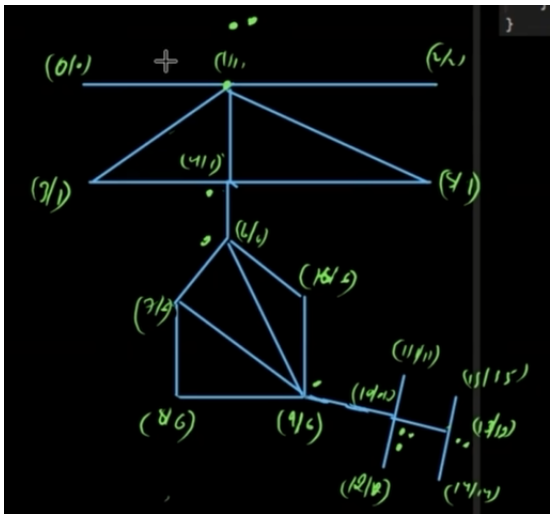
To more generalize, initially made AP[i]=-1 so that formula becomes dots +1 for all vertex.
And if there is only one call from root then it will be 0 since the normal dfs call make the root a articulation point, which is wrong.

And if the number of calls is more than 1, then the formula remain the same for number of component formed. (dots + 1) ==> in code (AP[i] + 1);

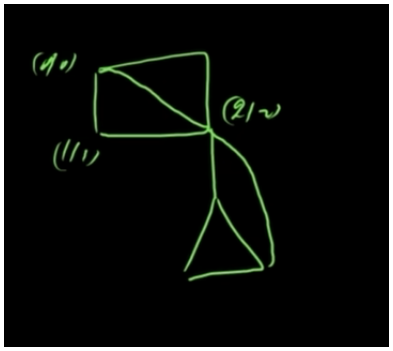
```
// ! Important Point :

// # Kyunki hume mostly sirf ye pucha jayega ki ye point articulation point hai
// # ki nhi, to hum boolean ka array use kar sakte hain aur wahan pe rootcalls
// # wali condition important ho jati hai root vale case ko handle karne mai
```

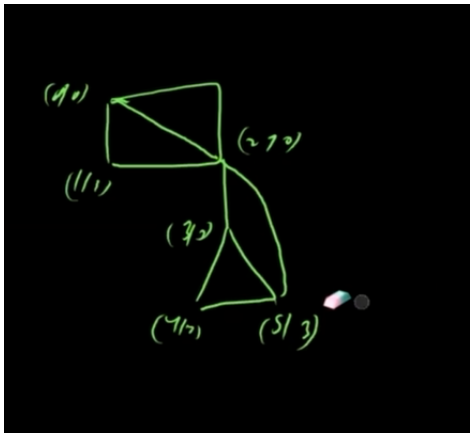


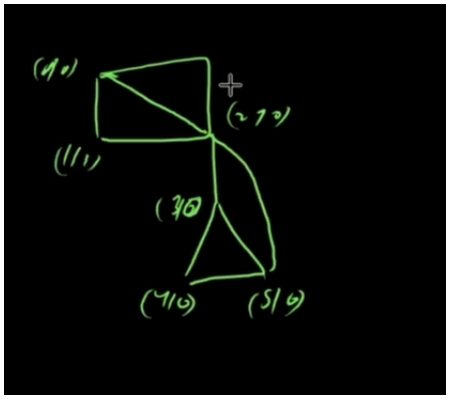




Below is the important test case. This will tell us why we have not compared low time.

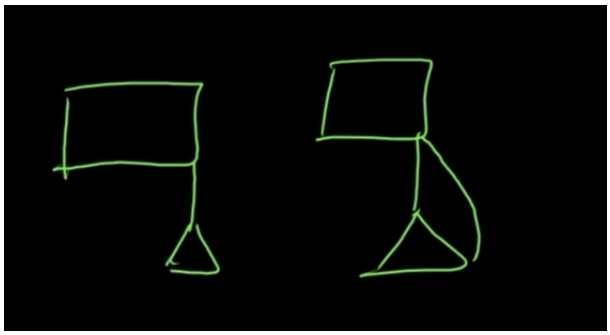


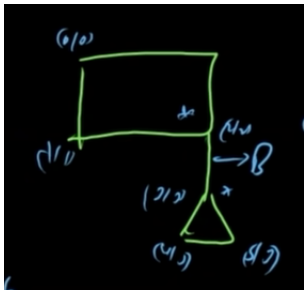


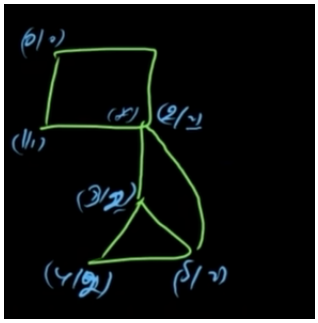


```
// ! Why have we compared with discovery time, not low time.

// In the notes section, consider the test case.
// Agar manle mai low time se compare karta hunto (5,2) ki jagah pe (5,0) hota.
// Similarly, (4,4) update hota (4,0) mai
// Similarly, (3,3) update hota (3,0) mai

// Ab agar tu (2,0) mai hota to tu (3,0) ko dekh ke ye sochta ki 3 ke pass aur
// koi rasta hai 0 tak pahunchne ka jo ki mujhse bhi powerful hai. To 3 ko meri
// need he nhi hai

// # But actual mai aisa to hai he nhi. 2 wo jaria hai jiski wajah se 3 uper
// # wale components se connected hai. Therefore ye galat hai.

// ? Isiliye hum discovery ko compare karte hain. Naki low time ko.
```

For Articulation Edge, use the below test case to find the condition.

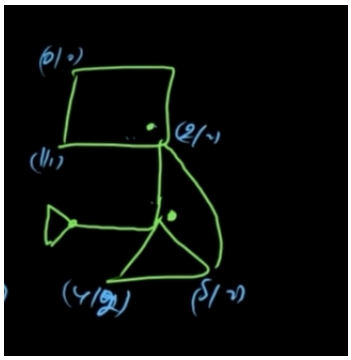






B denotes the articulatino Bridge

```
// ! Important Point :
// # Do articuation point ke beech mai ek articuation edge exist kare humesha ye
// # baat sach nhi hogi.Like in the below test case.
```



```
private static int[] low, disc, AP;
private static int time = 0, rootCalls;
private static boolean[] vis, APoints;

public static void dfs(int src, int par, ArrayList<Edge>[] graph) {
    disc[src] = low[src] = time;
    vis[src] = true;
    for (Edge e : graph[src]) {
        if (!vis[e.v]) {
            dfs(e.v, src, graph);

            if (par == -1) // Agar root he parent to parent to -1 he hoga na.
                rootCalls++;
```

```
            // Articualation Point
            if (disc[src] <= low[e.v]) { // Agar meri discovery pehle hui hai jis most powerful vertex ko tu janta
                                         // hai to mai to ek articuation point to honga he
                AP[src]++;
                APoints[src] = true;
            }

            // Articulation Edge
            if (disc[src] < low[e.v]) {
                System.out.println("Articulation Edge : (" + src + "," + e.v + ") ");
                // e.v is a articuation edge with src
            }

            low[src] = Math.min(low[src], low[e.v]);

        } else if (e.v != par) { // Agar e.v mera parent nhi hai aur wo already visited hai, to mai apne aap ko
                                 // discovery time se update kar lunga
            low[src] = Math.min(low[src], disc[e.v]);
        }
    }
}
```

```
public static void APB(ArrayList<Edge>[] graph, int N) {

    low = new int[N];
    disc = new int[N];
    AP = new int[N];
    vis = new boolean[N]; // Not needed. We can just fill the disc Array with -1 initially.
    APoints = new boolean[N];

    for (int i = 0; i < N; i++) {
        if (!vis[i]) {
            AP[i] = -1; // For specific root to generalize the formula of components formed to (dots +
                        // 1) where in code dots means AP[i] + 1;

            dfs(i, -1, graph);
            if (rootCalls == 1) // For the condition since this dfs, it tells that the root is also a
                                // articuation point evenif it is not. Consider a single line graph with nodes.
                                // So if the number of calls are greater than one, then the root is also a
                                // articulation point.
                APoints[i] = false;

        }

    }
}
```

61. Critical Connections in a Network

1192. Critical Connections in a Network

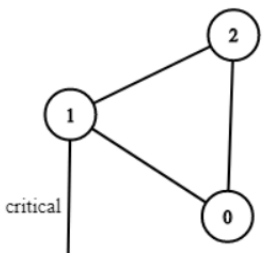
Hard 4939 167 Add to List Share

There are `n` servers numbered from `0` to `n - 1` connected by undirected server-to-server `connections` forming a network where `connections[i] = [a_i, b_i]` represents a connection between servers a_i and b_i. Any server can reach other servers directly or indirectly through the network.

A *critical connection* is a connection that, if removed, will make some servers unable to reach some other server.

Return all critical connections in the network in any order.

Example 1:



```
Input: n = 4, connections = [[0,1],[1,2],[2,0],[1,3]]
Output: [[1,3]]
Explanation: [[3,1]] is also accepted.
```

Example 2:

```
Input: n = 2, connections = [[0,1]]
Output: [[0,1]]
```

Constraints:

- $2 <= n <= 10^5$
- $n - 1 <= connections.length <= 10^5$
- $0 <= a_i, b_i <= n - 1$
- $a_i != b_i$
- There are no repeated connections.

```java
// b <=======Critical Connections in a Network ==================>
// https://leetcode.com/problems/critical-connections-in-a-network/

public static void criticalConnections_dfs(int src, int par, ArrayList<Integer>[] connections,
        List<List<Integer>> ans) {
    disc[src] = low[src] = time++;
    vis[src] = true;
    for (int v : connections[src]) {
        if (!vis[v]) {
            criticalConnections_dfs(v, src, connections, ans);
            if (disc[src] < low[v]) {
                List<Integer> myAns = new ArrayList<>();
                myAns.add(src);
                myAns.add(v);
                ans.add(myAns);
            }
            low[src] = Math.min(low[src], low[v]);
        } else if (v != par)
            low[src] = Math.min(low[src], disc[v]);
    }
}
```

```java
public List<List<Integer>> criticalConnections(int n, List<List<Integer>> connections) {

    List<List<Integer>> ans = new ArrayList<>();
    time = 0;
    vis = new boolean[n];
    disc = new int[n];
    low = new int[n];

    ArrayList<Integer>[] graph = new ArrayList[n]; // Created our own graph from edges
    for (int i = 0; i < n; i++) {
        graph[i] = new ArrayList<>();
    }

    for (List<Integer> e : connections) {
        int u = e.get(index: 0), v = e.get(index: 1);
        graph[u].add(v);
        graph[v].add(u);
    }

    for (int i = 0; i < n; i++) {
        if (!vis[i]) {
            criticalConnections_dfs(i, -1, graph, ans);
        }
    }
```

```
        }

        return ans;
    }
```

62. Minimum Number of Days to Disconnect Island

1568. Minimum Number of Days to Disconnect Island

Hard 497 134 Add to List Share

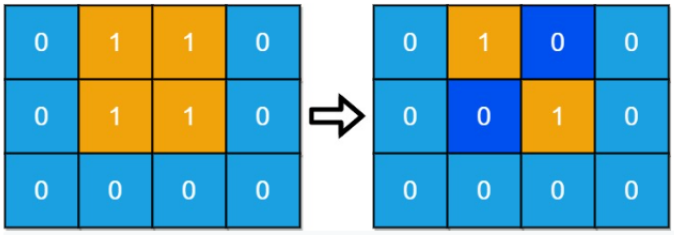
You are given an `m x n` binary grid `grid` where `1` represents land and `0` represents water. An **island** is a maximal **4-directionally** (horizontal or vertical) connected group of `1`'s.

The grid is said to be **connected** if we have **exactly one island**, otherwise is said **disconnected**.

In one day, we are allowed to change **any** single land cell `(1)` into a water cell `(0)`.

Return *the minimum number of days to disconnect the grid.*

Example 1:




```
Input: grid = [[0,1,1,0],[0,1,1,0],[0,0,0,0]]

Output: 2
Explanation: We need at least 2 days to get a disconnected grid.
Change land grid[1][1] and grid[0][2] to water and get 2 disconnected island.
```

Example 2:


```
Input: grid = [[1,1]]
Output: 2
Explanation: Grid of full water is also disconnected ([[1,1]] -> [[0,0]]), 0 islands.
```

Example 3:

```
Input: grid = [[1,0,1,0]]
Output: 0
```

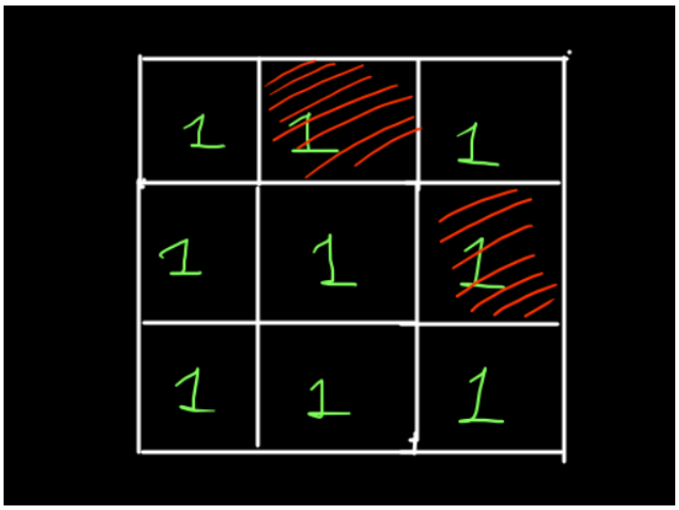
Example 4:

```
Input: grid = [[1,1,0,1,1],
               [1,1,1,1,1],
               [1,1,0,1,1],
               [1,1,0,1,1]]
Output: 1
```

Example 5:

```
Input: grid = [[1,1,0,1,1],
               [1,1,1,1,1],
               [1,1,0,1,1],
               [1,1,1,1,1]]
Output: 2
```

Answer to why atmost two can be the answer :




```java
// b <================== Minimum Number of Days to Disconnect Island ==========>
// https://leetcode.com/problems/minimum-number-of-days-to-disconnect-island/

// ! Brute Force, passed since the test case and constraints are small.

public static void dfs_numsIsland(int sr, int sc, int[][] grid, int[][] dir, boolean[][] vis) {
    vis[sr][sc] = true;

    for (int d = 0; d < dir.length; d++) {
        int r = sr + dir[d][0];
        int c = sc + dir[d][1];

        if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 1 && !vis[r][c]) {
            dfs_numsIsland(r, c, grid, dir, vis);
        }
    }
}
```

```java
public static int numOfIslands(int[][] grid) {
    int n = grid.length, m = grid[0].length;
    boolean[][] vis = new boolean[n][m];

    int noOfIslands = 0;
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, -1 }, { 0, 1 } };
    for (int i = 0; i < n * m; i++) {
        int r = i / m, c = i % m;
        if (!vis[r][c] && grid[r][c] == 1) {
            dfs_numsIsland(r, c, grid, dir, vis);
            noOfIslands++;
        }
    }

    return noOfIslands;
}
```

```java
public int minDays_(int[][] grid) {
    int n = grid.length, m = grid[0].length;
    int initialComponents = numOfIslands(grid);
    if (initialComponents > 1 || initialComponents == 0)
```

```
            return 0;

        for (int i = 0; i < n * m; i++) {
            int r = i / m, c = i % m;

            if (grid[r][c] == 1) {
                grid[r][c] = 0;
                int noOfComponents = numOfIslands(grid);
                if (noOfComponents > 1 || noOfComponents == 0)
                    return 1;
                grid[r][c] = 1;
            }
        }
        return 2;
    }
```

```
    // B Optimized using the articulation Point

    private static int[] low, disc;
    private static int time = 0;
    private static boolean[] vis;

    public static int dfs_size(int idx, int[][] grid, boolean[] vis) {
        int n = grid.length, m = grid[0].length;
        int sr = idx / m, sc = idx % m;

        vis[idx] = true;

        int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, -1 }, { 0, 1 } };
        int count = 0;

        for (int d = 0; d < dir.length; d++) {
            int r = sr + dir[d][0];
            int c = sc + dir[d][1];

            if (r >= 0 && c >= 0 && r < grid.length && c < grid[0].length && grid[r][c] == 1 && !vis[r * m +
                count += dfs_size(r * m + c, grid, vis);
            }
        }

        return count + 1;
    }
```

```
    public static boolean tarjans(int src, int par, int[][] grid) {
        int n = grid.length, m = grid[0].length;
        disc[src] = low[src] = time++;
        vis[src] = true;

        int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, -1 }, { 0, 1 } };

        boolean res = false;
        for (int d = 0; d < dir.length; d++) {
            int sr = src / m, sc = src % m;

            int r = sr + dir[d][0];
            int c = sc + dir[d][1];

            if (r >= 0 && c >= 0 && r < n && c < m && grid[r][c] == 1) {
                int nbr = r * m + c;
                if (!vis[nbr]) {
                    res = res || tarjans(nbr, src, grid);
                    if (disc[src] < low[nbr]) { // Yahan pe equal to sign nhi kiya use. Why? ==> Kyunki hume cycle wale
                                                // structure ke liye bhi true return karna tha kyunki wahan pe do vertex
                                                // ko nikal ke graph disconnected ban sakta hai. Example is of a square,
                                                // removing the diagonal vertex will make component disconnected.
                        return true;
                    }
                    low[src] = Math.min(low[nbr], low[src]);
                } else if (nbr != par) {
                    low[src] = Math.min(low[src], disc[nbr]);
                }
```

```
        }

    }
    return res;
}
```

```
public int minDays(int[][] grid) {
    int n = grid.length, m = grid[0].length;

    disc = new int[n * m];
    low = new int[n * m];
    vis = new boolean[n * m];
    int root = -1;
    int noOfComponents = 0, size = 0;
    for (int i = 0; i < n * m; i++) {
        int r = i / m, c = i % m;

        if (grid[r][c] == 1 && !vis[i]) {
            root = i;
            size += dfs_size(i, grid, vis);
            noOfComponents++;
        }
    }

    if (noOfComponents == 0 || noOfComponents > 1) // Agar mera component 0 hai ya 1 ha se bada hai to mai already
                                                   // disconnected hun, to 0 retun kardo
        return 0;
    else if (size <= 2) // Ab kyunki mai uper component ka check karke aaya hun to mai sure hun ki ab ek
                        // he single component hai graph mai. To agar component ka size 1 ya 2 hua, to
                        // ` utne he din lagte use disconnect karne mai jitna size hota.
        return size;

    vis = new boolean[n * m];
    boolean res = tarjans(root, -1, grid);
    return res ? 1 : 2; // Ab agar mujhe articulation point milta hai to mai to mai 1 return kardunga,
                        // aur agar nhi milta hai to mai 2 return kardunga kyunki at most mera answer 2
                        // ho sakta hai.

}
```