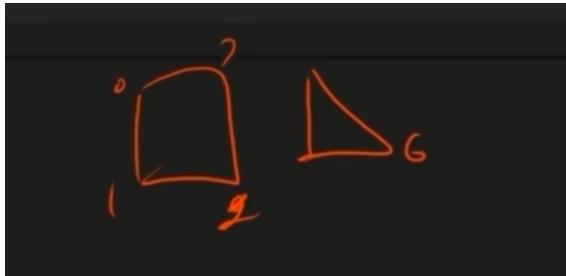


## Graphs Level Up

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; // 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(0, "" + src);
    }

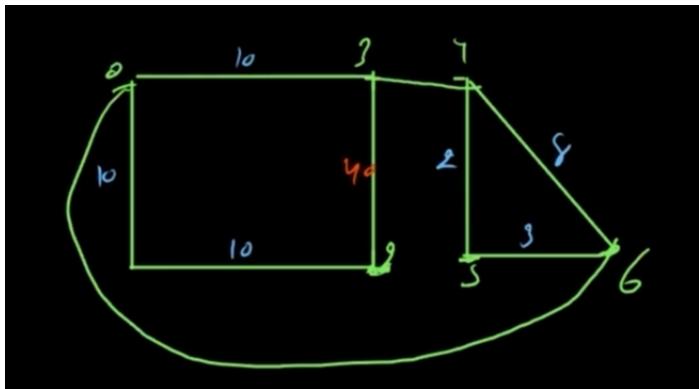
    vis[src] = true;
    Pair ans = new Pair();
    for (Edge e : graph[src]) {
        if (!vis[e.v]) {
            Pair recAns = heaviestPath(graph, e.v, dest, vis);

            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;
            }
        }
    }

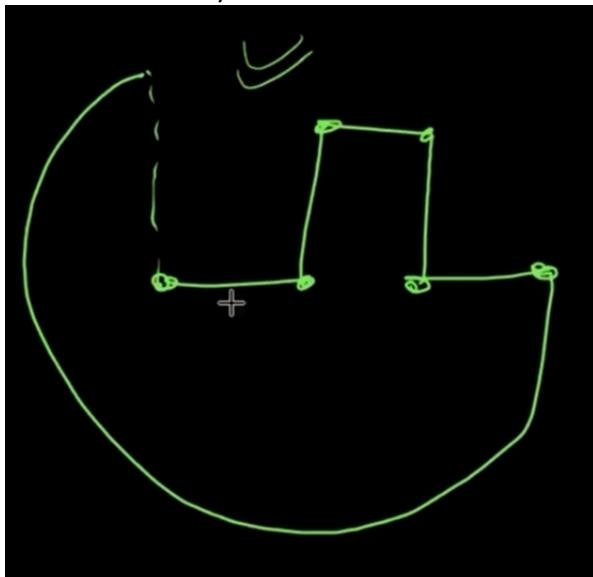
    vis[src] = false;
    return ans;
}
```

### 2. Hamiltonian Path and Cycle (Single Source Algorithms)

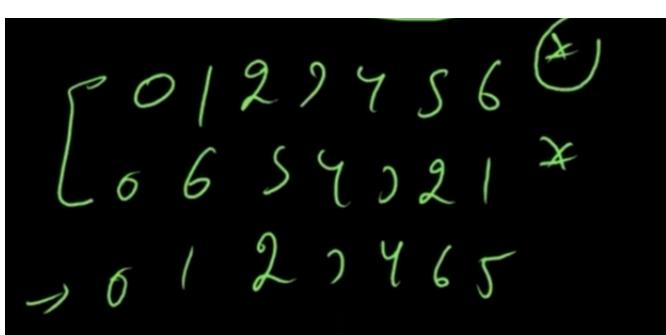
Note -> A hamiltonian path is such which visits all vertices without visiting any twice. A hamiltonian path becomes a cycle if there is an edge between first and last vertex.



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.

```
// b <===== Hamiltonian Path and cycle =====>
// # Also known 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;
}
```

```
// 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) {

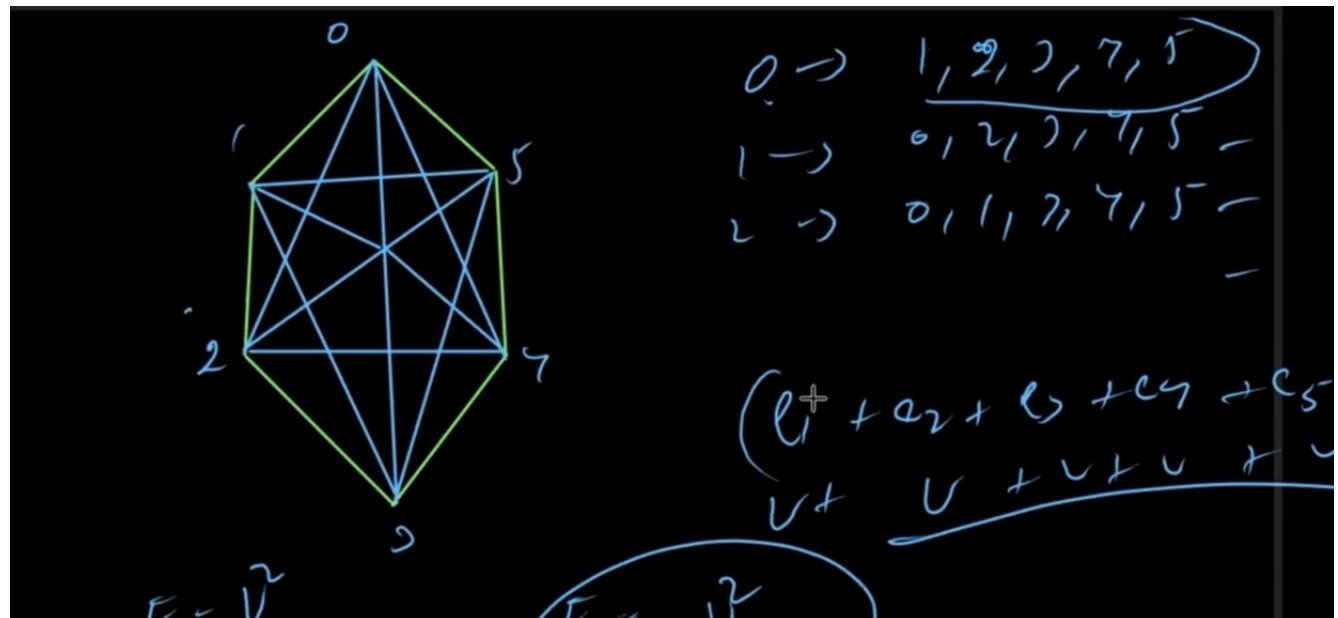
    if (edgeCount == graph.length - 1) {

        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  $e_1 + e_2 + e_3 + e_4 + \dots + e_{v-1}$ .

But if we see closely, every vertex is connected to every other vertex. So  $e_1 = e_2 = e_3 = e_4 = V$  (Total number of vertex)

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

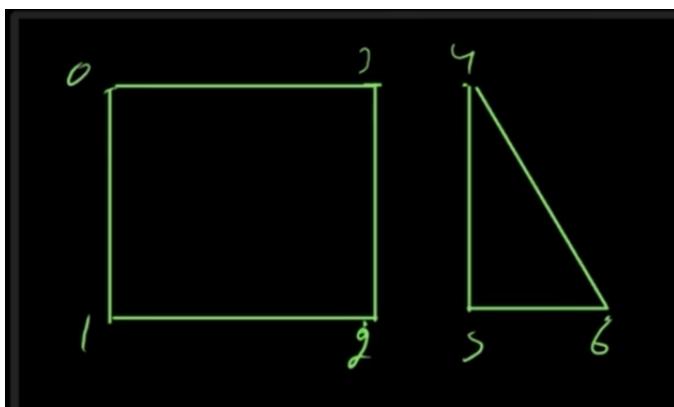
$O(E) ==> O(V^2)$

#### 4. Just Random Information

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

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

#### 5. Get Connected Components



```
// 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  $m \times n$  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:

- $m == \text{grid.length}$
- $n == \text{grid[i].length}$
- $1 \leq m, n \leq 300$
- $\text{grid}[i][j]$  is '0' or '1'.

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

// This is same as get connected components.
// 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  $m \times n$  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	0	1	1	1	0	0	0	0
0	1	1	0	1	0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	0	1	0	1	0	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	0	1	0	0	0
0	0	0	0	0	0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	1	1	0	0	0	0	0

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

**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 <= 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 the number of islands but here we also have to calculate 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 neighbours 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;
}
```

```
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

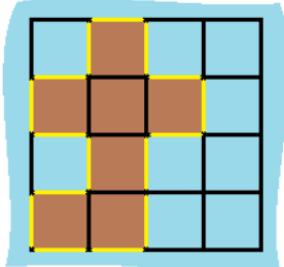
Easy   4873   254   Add to List   Share

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]]

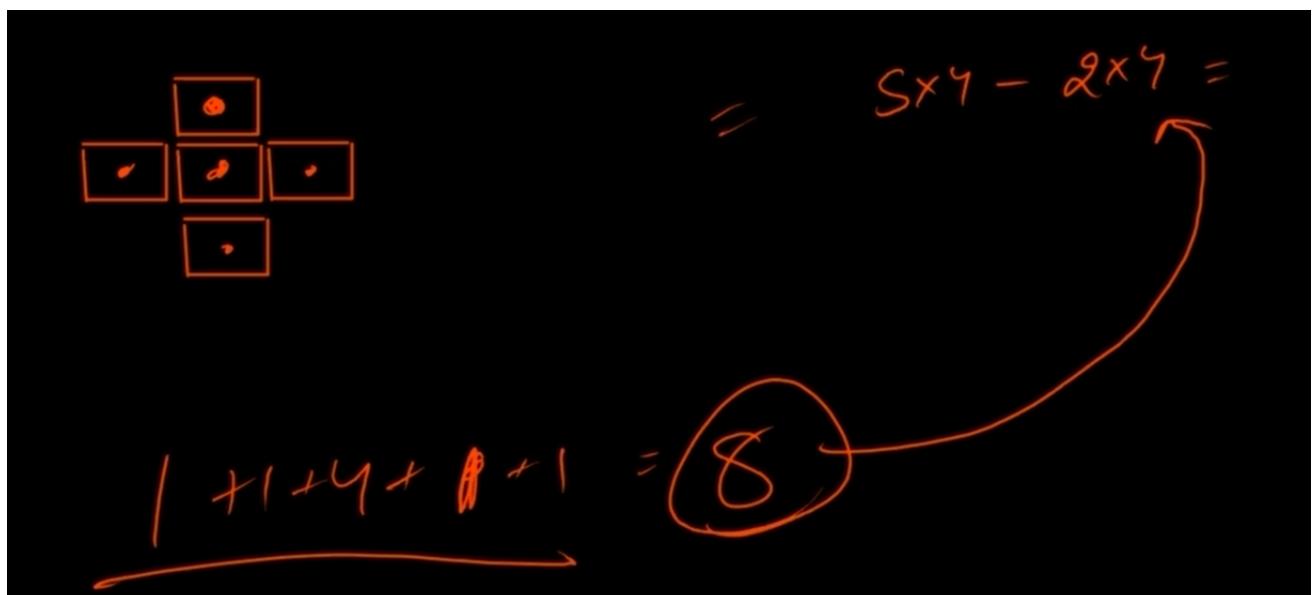
**Output:** 16

**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 dono 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 squares). 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 || r >= n || c < 0 || c >= m || grid[r][c] != 0) {
                        nbrCount++;
                    }
                }
            }
        }
    }
    return 4 * oneCount - nbrCount;
}
```

```

        int r = i + dir[a][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

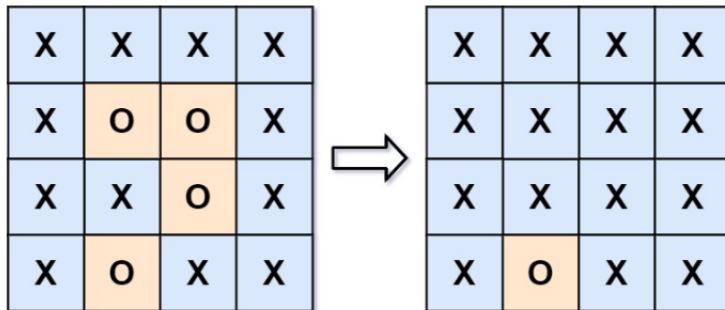
### 130. Surrounded Regions

Medium    5665    1336    Add to List    Share

Given an  $m \times n$  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.



```

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

// Agar koi basically boundary wale 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.

```

```

// 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 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);
        }
    }
}

```

```

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  $m \times n$  binary matrix  $\text{grid}$ . 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/or reflected

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:**

1	1	0	0	0
1	1	0	0	0
0	0	0	1	1
0	0	0	1	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:**

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

**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 RXDLXXX
// For the bottom test case the string will be RDLDXXXX

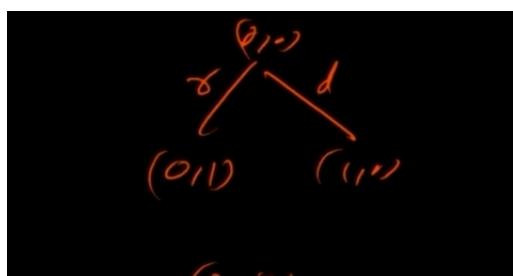
// Hence Denoting the two structures are different.

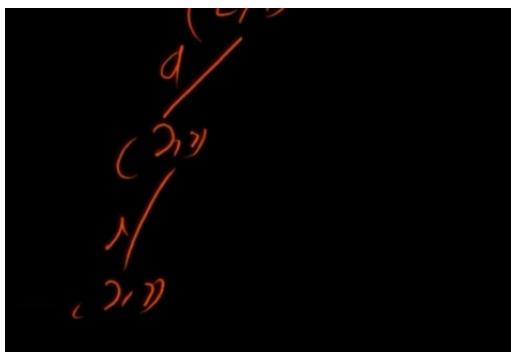
```

1 1  
1

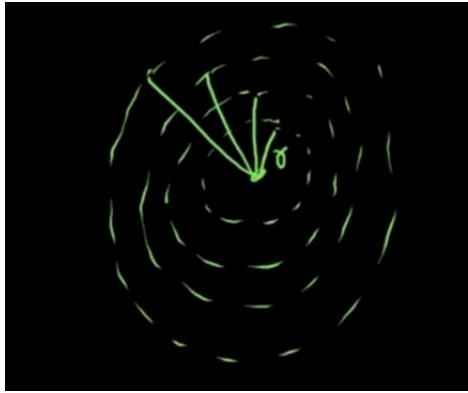
Is not same as 1 1  
1

Run like this





### 11. Cycle Bfs in Graph

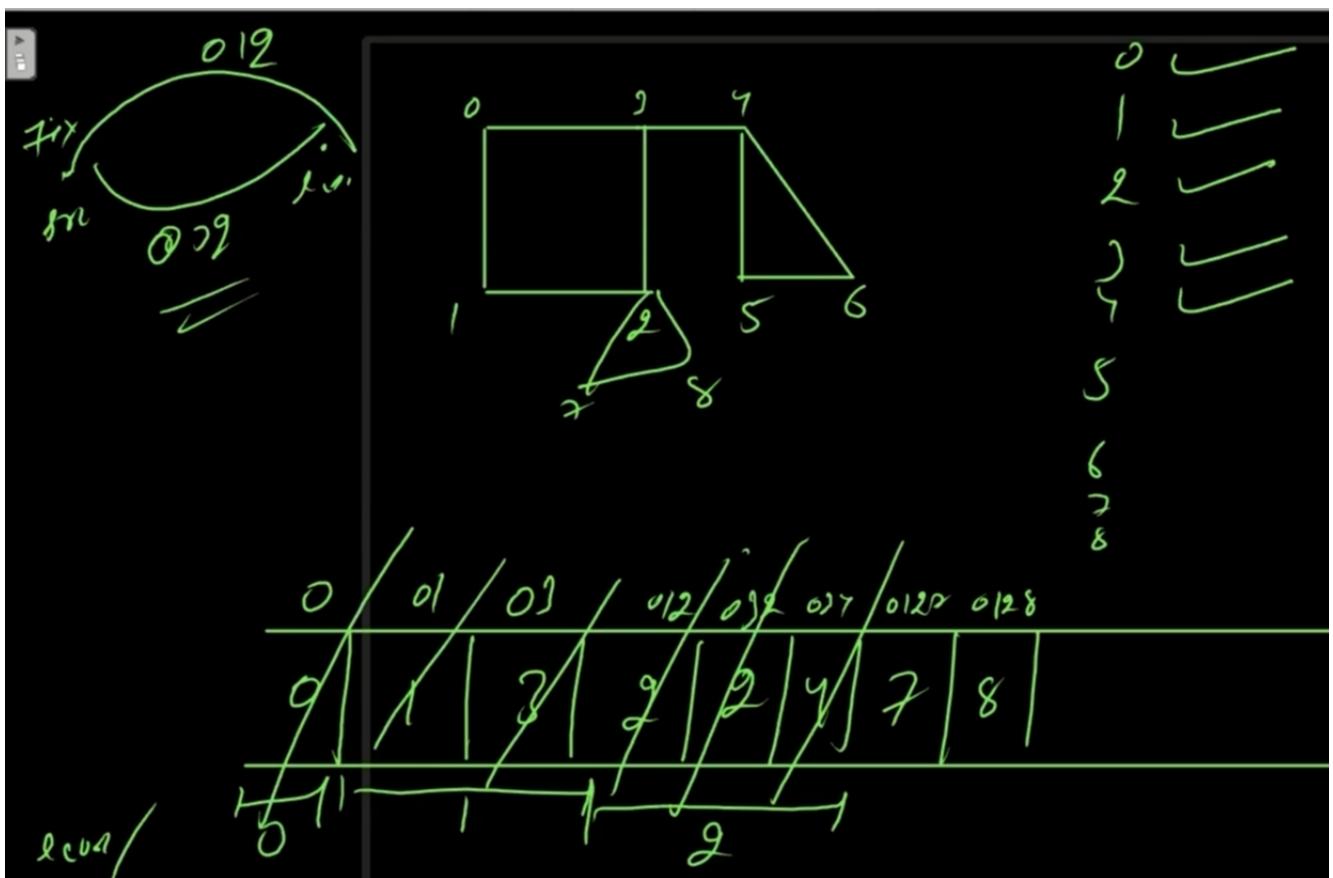


// ? It gives us the shortest path from the source vtx to the destination  
// ? vtx.

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

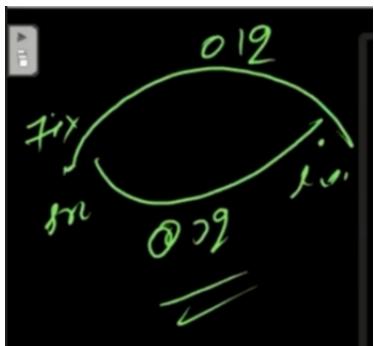
// Where radius is in terms of edges.

// 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 ...

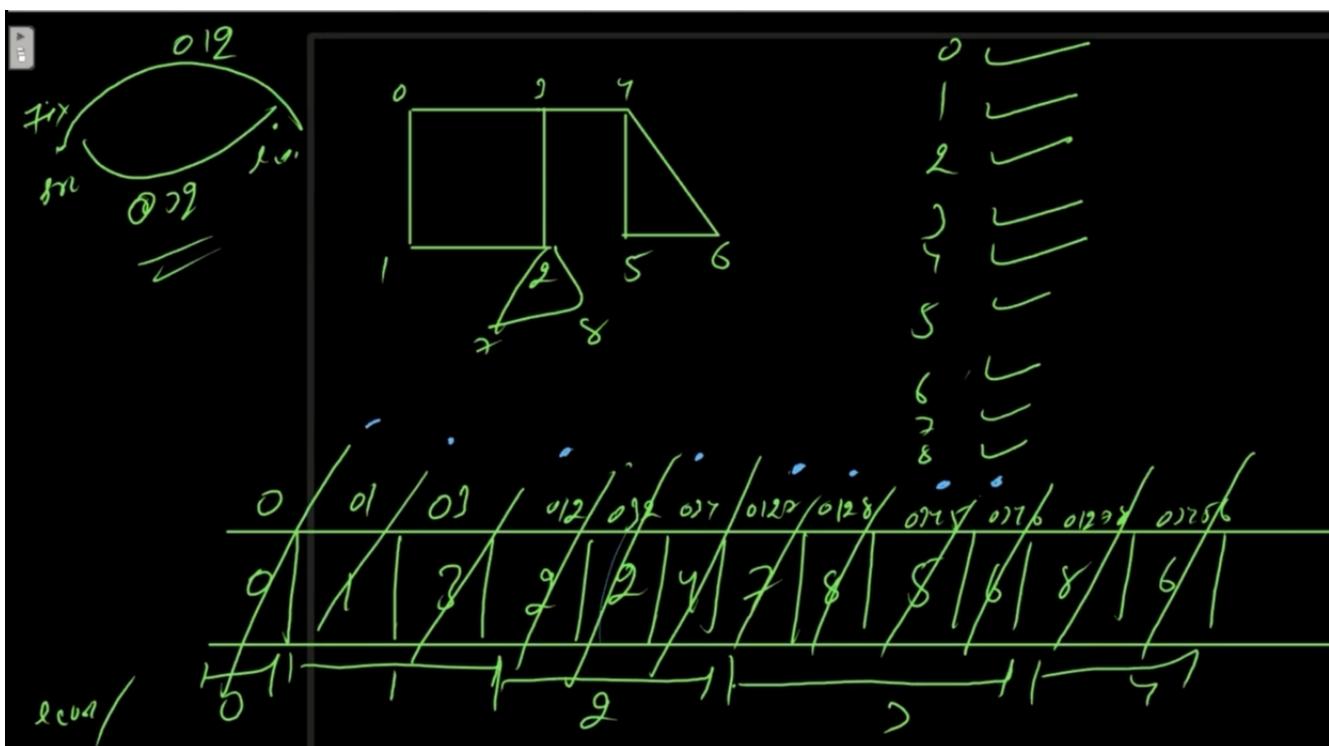


```
// # Agar koi vertex already visited hai aur wo dubara visit hoti hai, to us
// # graph mai cycle hai. Matlab us vertex tak pahunchne ke do raste hai from
// # the fixed source

// 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
```

```
// 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("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();
}

```

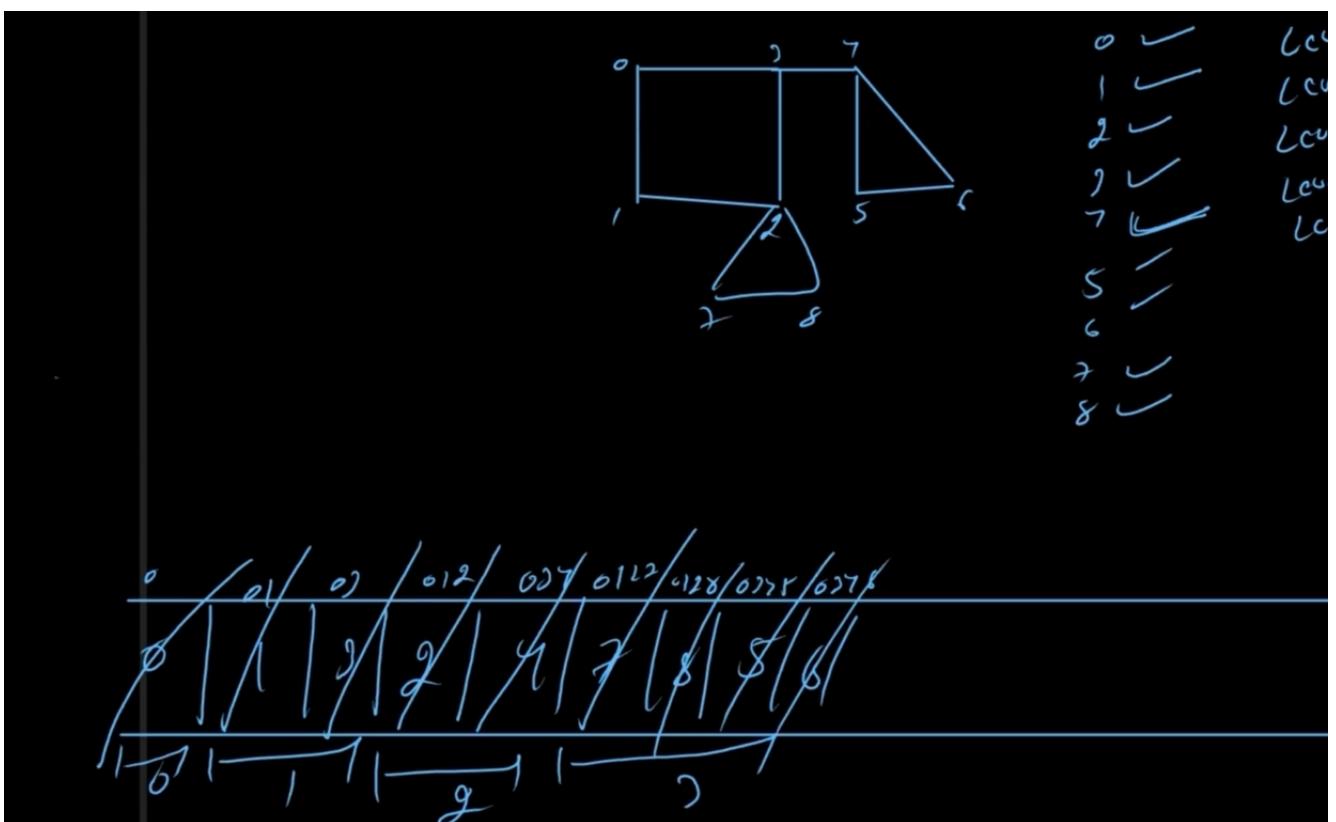
```

if (vis[vtx]) {
    System.out.println("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 therefore the  
// ? complexity is  $O(V)$ , means it is faster than the above one.

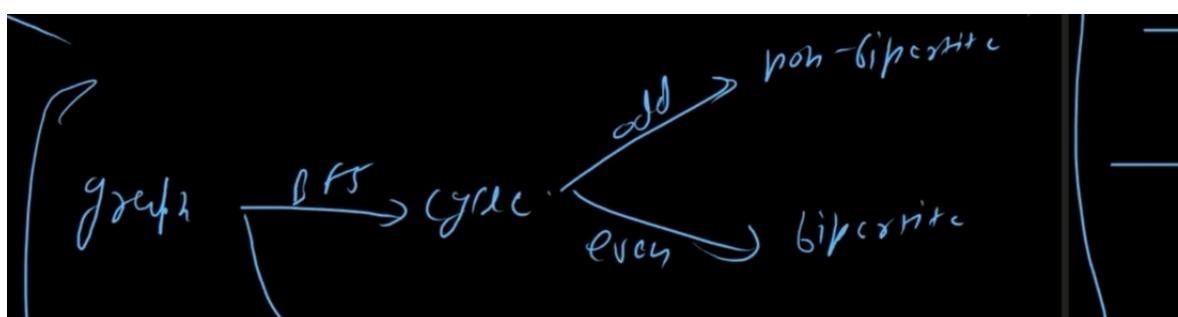
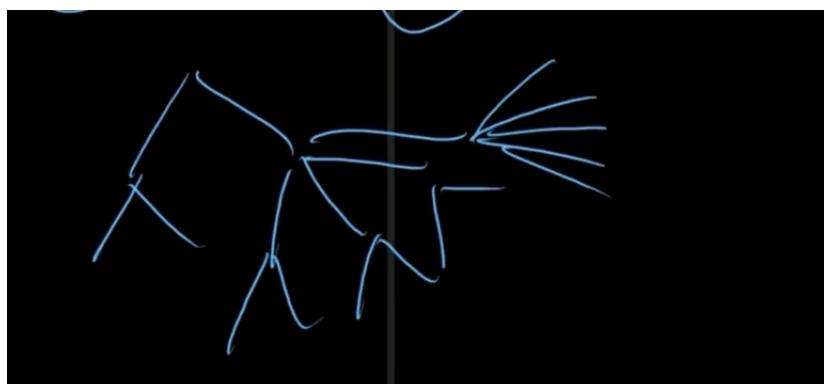
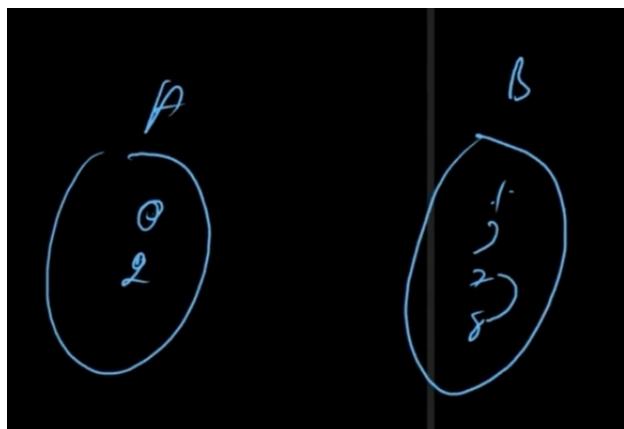
```
public static void bfs_withoutcycle(ArrayList<Edge>[] graph, int src, boolean[] vis) {
    LinkedList<Integer> que = new LinkedList<>();
    que.add(src);
    vis[src] = true;

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

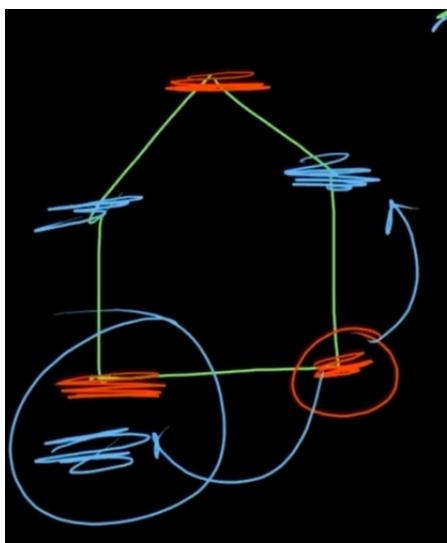
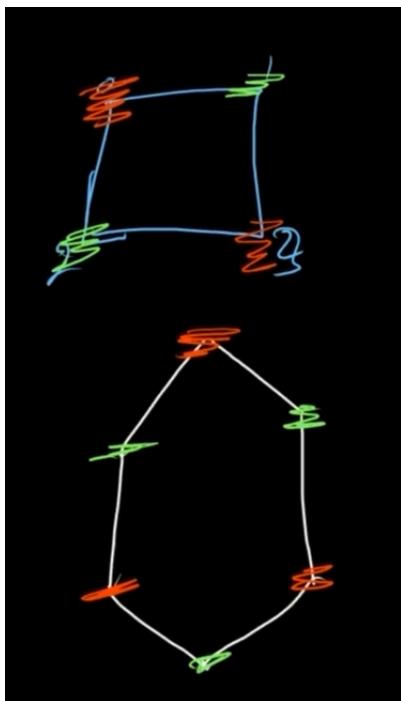
        while (size-- > 0) {
            int vtx = que.removeFirst();
            System.out.print(vtx + ", ");

            for (Edge e : graph[vtx]) {
                if (!vis[e.v]) {
                    vis[e.v] = true;
                    que.addLast(e.v);
                }
            }
        }
        level++;
        System.out.println();
    }
}
```

### 13. Bipartite Graph



non-cyclic graph  $\longrightarrow$  bipartite



```
// 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
// humesa bi-partite hogा.

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

// # 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 hogा
```

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

```
// ? 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];

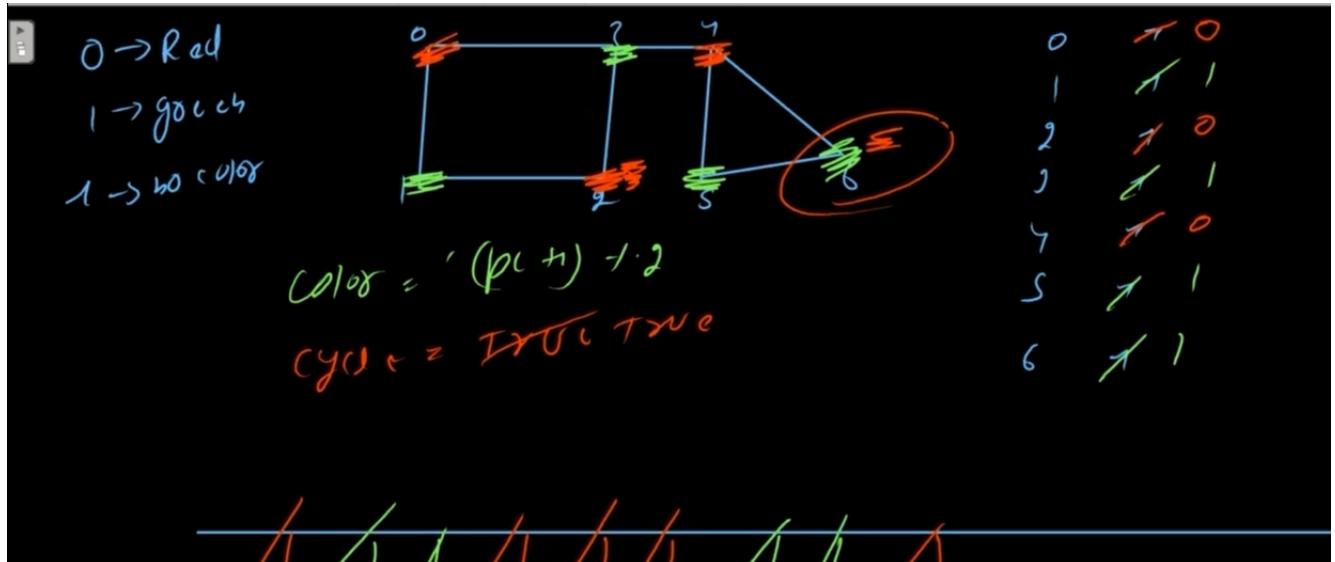
                if (colorShouldBe != actualColor) // conflict
                    return false;
                continue;
            }
            vis[vtx] = level % 2 == 0 ? 1 : 2;

            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-partite nhi hhoga
            return false;
        }
    }
    return true;
}
```

#### 14. Bi-partite and Even Odd Cycle





```
//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("Bipartite Graph with no cycle");
    else {
        if (isBipartite)
            System.out.println("Bipartite Graph with Even Length cycle");
        else
            System.out.println("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  $m \times n$  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],[0,1,1],[1,0,1]]
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 Parallel 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 phi 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 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 hogा 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

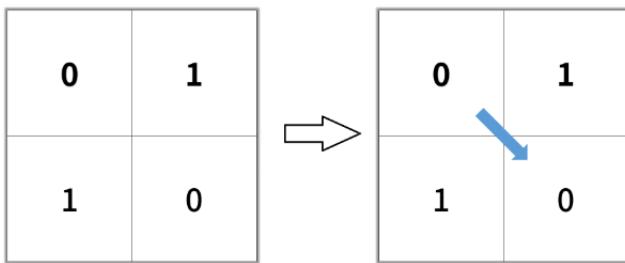
Given an  $n \times n$  binary matrix `grid`, return the length of the shortest **clear path** in the matrix. If there is no clear path, return `-1`.

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.

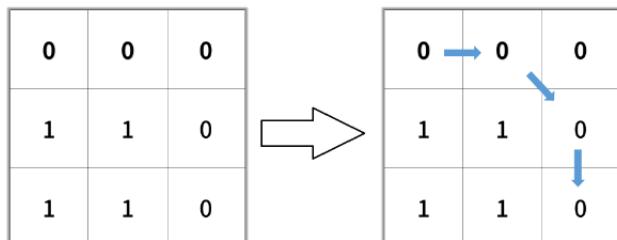
### Example 1:



**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
```

```
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(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 upper wale loop ke bahar aaya isska 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  $m \times n$  binary matrix `mat`, return the distance of the nearest `0` for each cell.

The distance between two adjacent cells is `1`.

#### Example 1:

0	0	0
0	1	0
0	0	0

**Input:** mat = [[0,0,0],[0,1,0],[0,0,0]]

**Output:** [[0,0,0],[0,1,0],[0,0,0]]

#### Example 2:

0	0	0
0	1	0
1	1	1

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

#### Constraints:

- $m == \text{mat.length}$
- $n == \text{mat[i].length}$
- $1 \leq m, n \leq 10^4$
- $1 \leq m * n \leq 10^4$
- $\text{mat[i][j]}$  is either 0 or 1.
- There is at least one 0 in  $\text{mat}$ .

0	0	0
0	1	0
1	1	1

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

T	T	T
T	T	T
T	T	T



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

// ? Again applied the concept of parallel 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.

```

```

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  $m \times n$  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:

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

```

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:**

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

```

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
            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

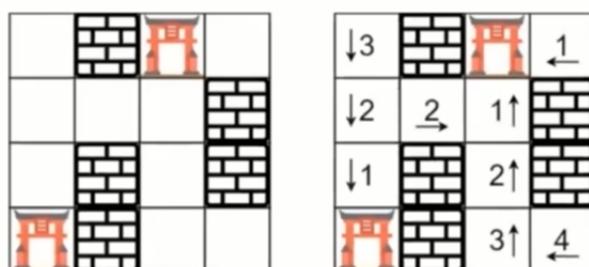
Medium    1761    27    Add to List    Share

You are given an  $m \times n$  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 without 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(r * m + c);
    }

}

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
                que.addLast(r * m + c); // distance pe hun usse + 1 mai hai mera neighbour isiliye
            }
        }
    }
    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

$n = 4$

$\text{astronaut} = [1, 2], [2, 3]$

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  $\text{astronaut}[p][2]$ : each element  $\text{astronaut}[i]$  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

```

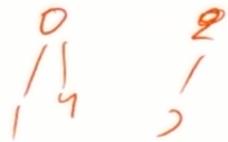
### Sample Output 0

6

### 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
```

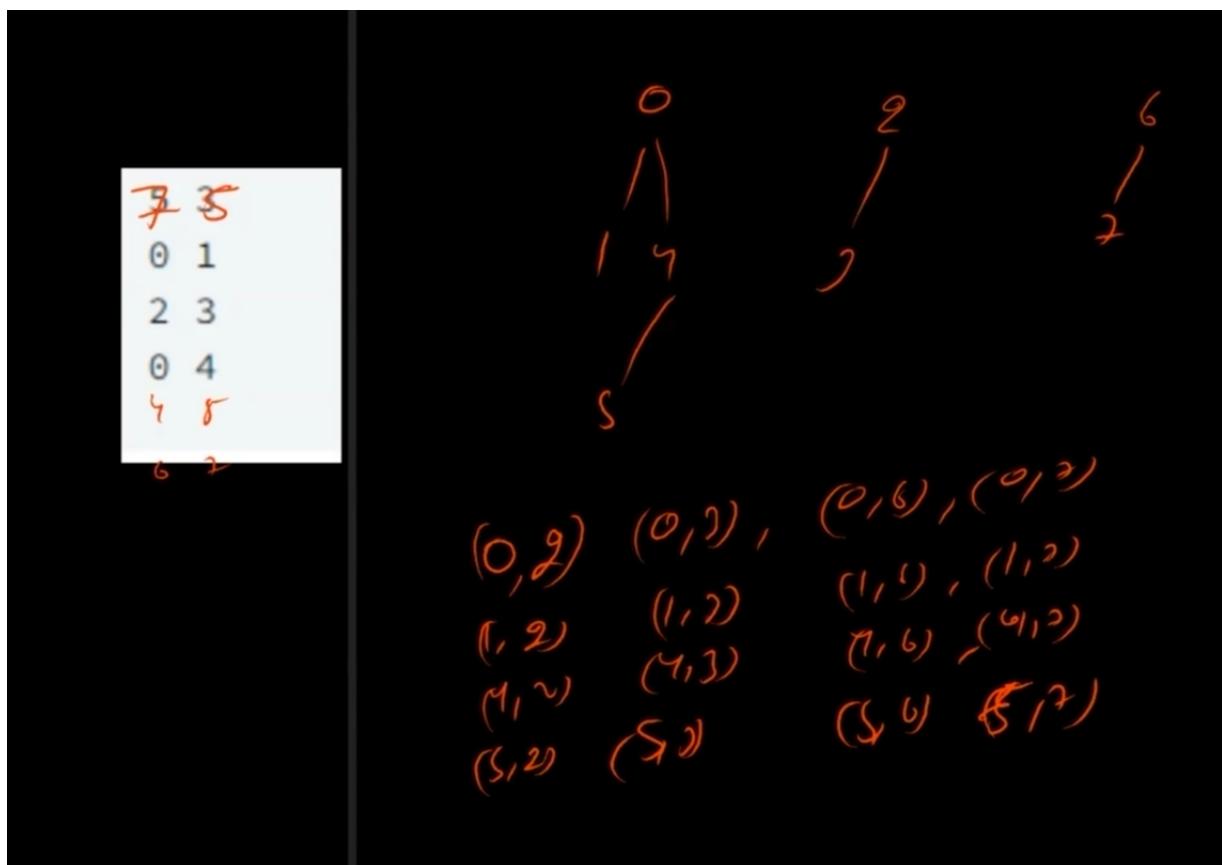
### Sample Output 1

```
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]



$$\begin{aligned}
 & (AB + A(C+AD+AC+AF) + BC + BD + BF + DF) + (CD + CE + \\
 & AC(B+C+D+F+E) + BC(C+D+E+F) + CD(C+D+E+F) + \\
 & BC(C+D+E+F)) \\
 \text{ans} &= f(0) + E(f) + D(E+f) + C(D+E+f) + B(C(D+E+f)) \\
 \text{sum} &= f + E + D + C + D + A
 \end{aligned}$$

```

// 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 component 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

```

```

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;
}

```

```

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

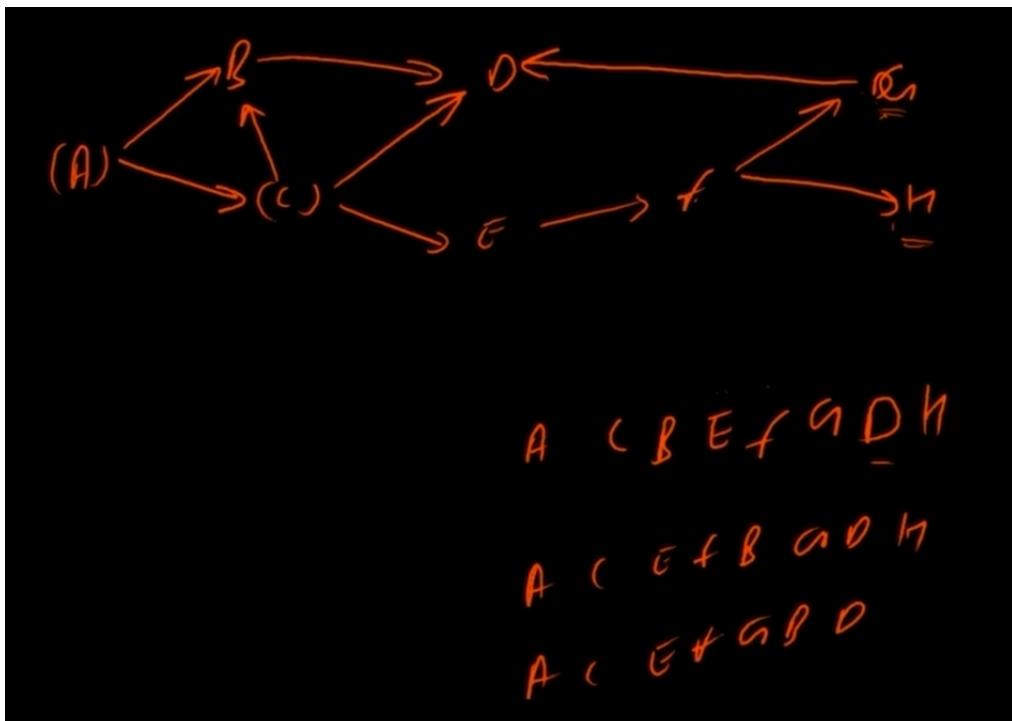
```

// 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 :

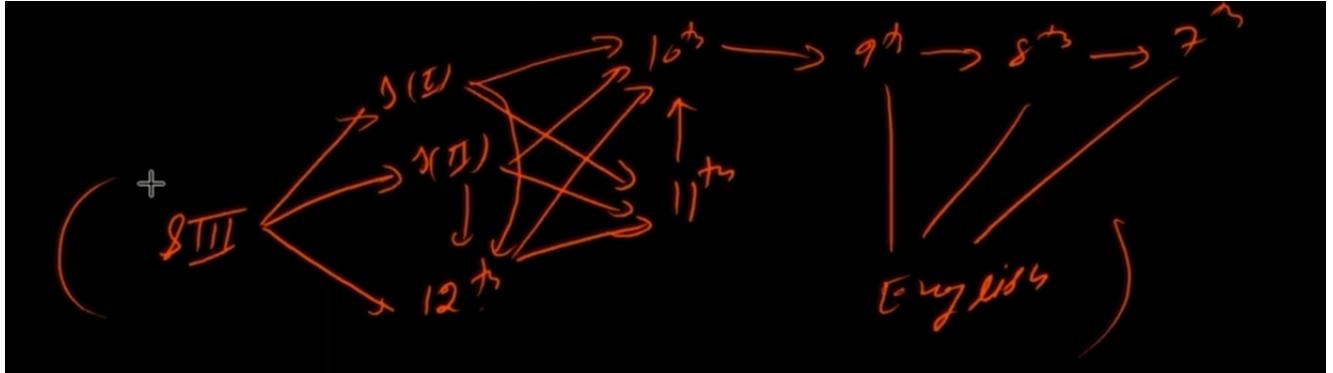
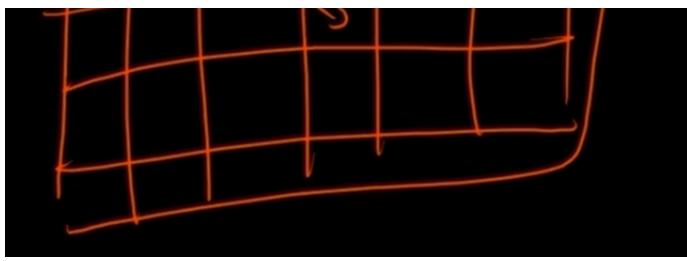
// 1. A Program execution during compilation. Aisa bahut bar hota hai ki kisi
// function ko pehle pure execute hone ke liye wo dusre function pe dependent
// hota hai

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

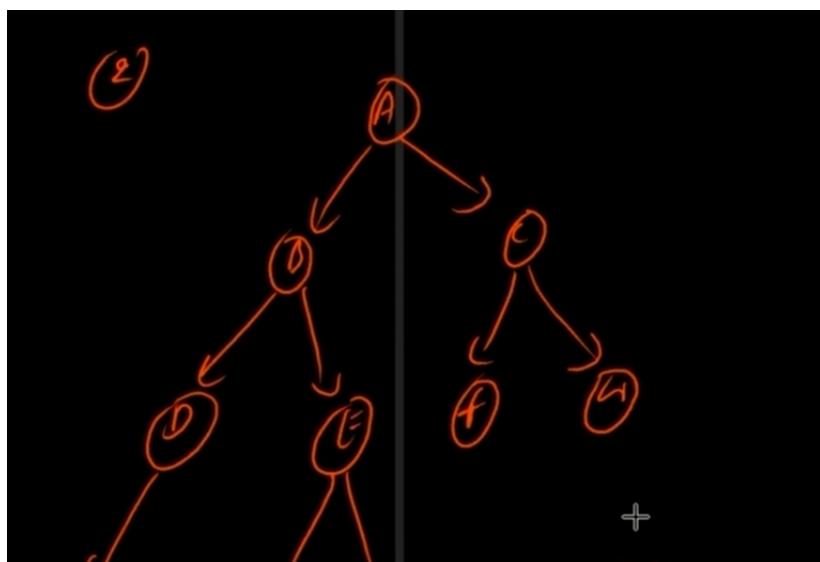
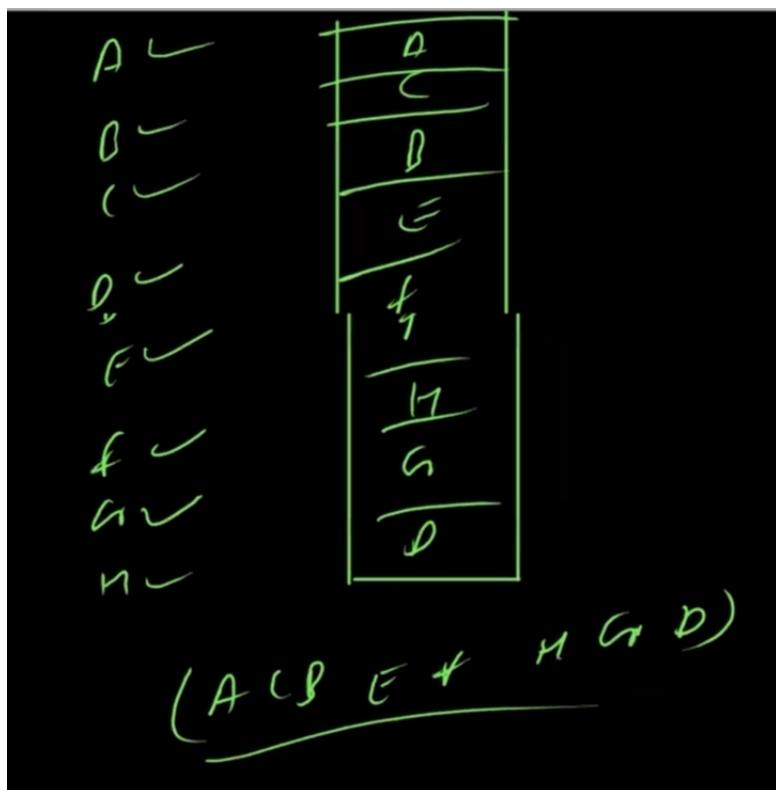
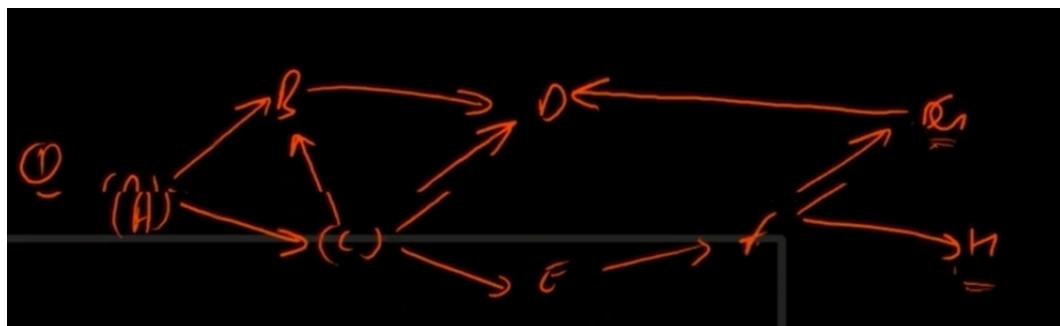
public static void topologicalOrder(int n, ArrayList<Edge>[] graph) {
    boolean[] vis = new boolean[n];
    ArrayList<Integer> ans = new ArrayList<>();
    for (int i = 0; i < n; i++) { // Hum yahan pe vertices ko randomly call bhi Laga sakte hai and tabhi bhi
        // answer same he aayega. Aisa 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));
    }
}
```

}

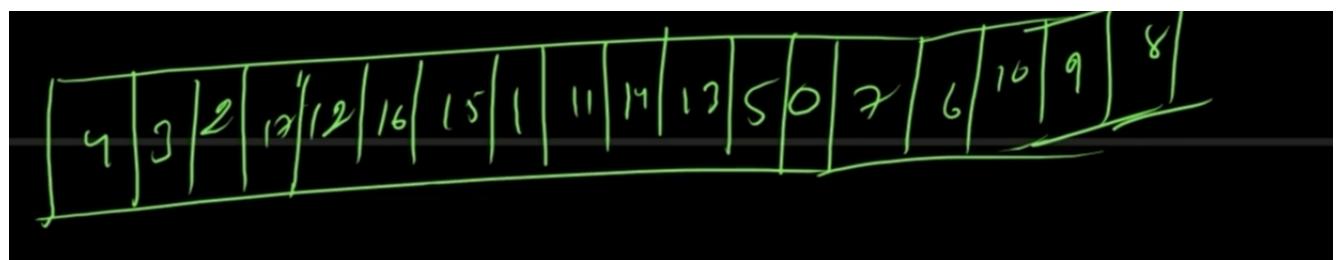
*// ! How to check if the topological order found is right or wrong :**// Humne edges bana ke dekha us order mai. aur agar sare edges same direction  
// mai aaye to order shi hai, nhi to order galat hai*

Ex :

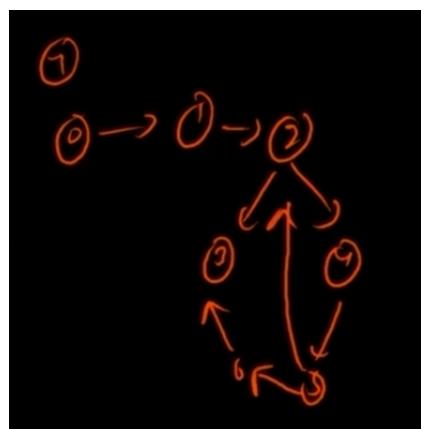




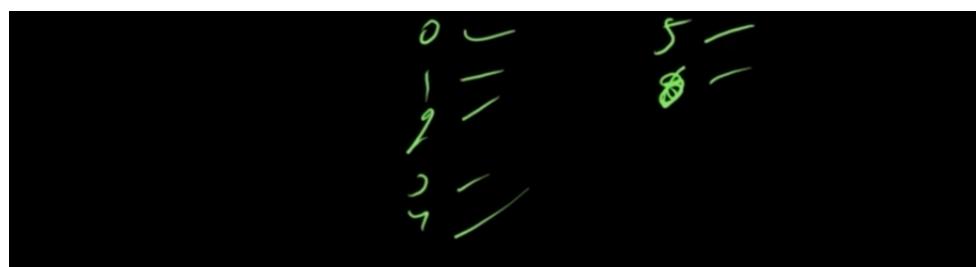
Normal Tree , just get the post order and check the topological order is right or wrong



$$O(E + V)$$



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 vtry = que.removeFirst();
```

```

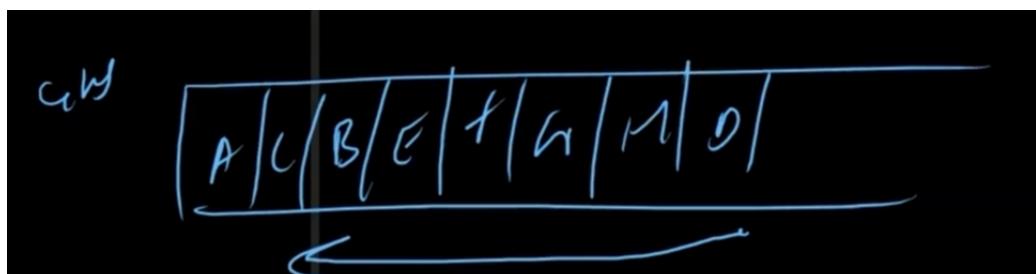
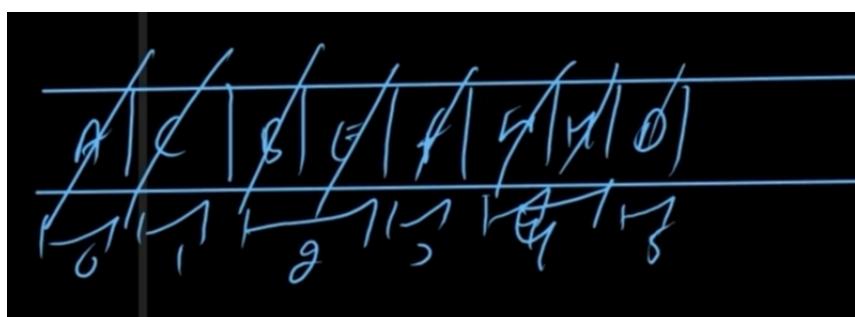
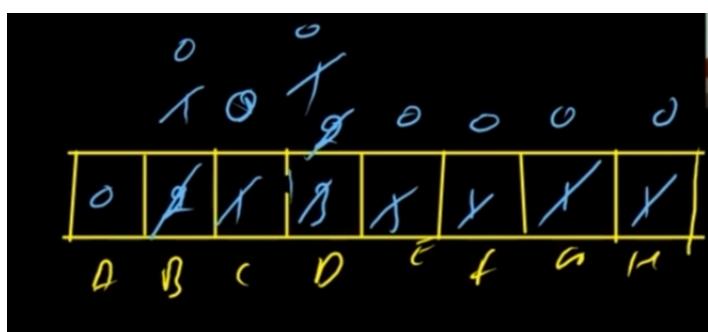
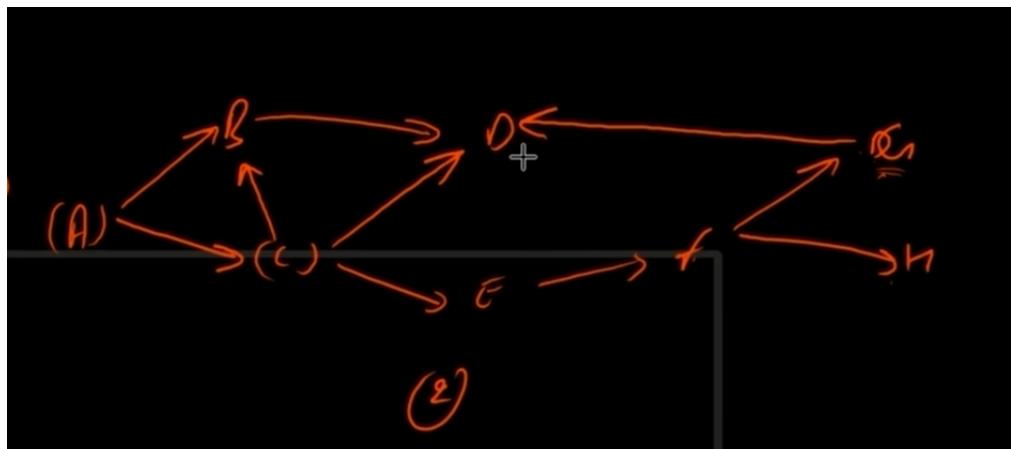
        while (vtx = que.removeFirst()) {
            ans.add(vtx);

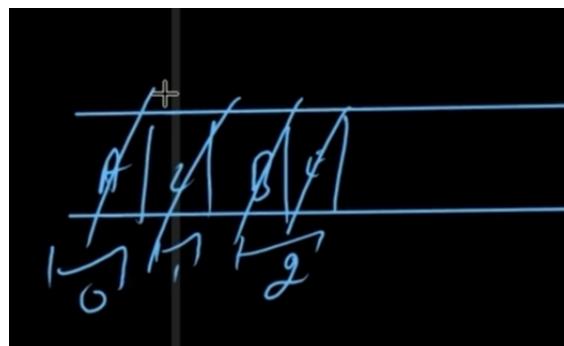
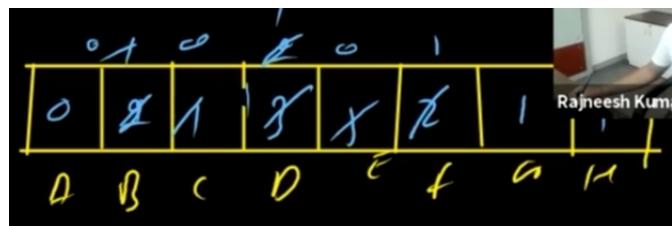
            for (Edge e : graph[vtx]) {
                if (-indegree[e.v] == 0)
                    que.addLast(e.v);
            }
        }
        level++;
    }

    if (ans.size() != n) {
        System.out.println("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 parallelly 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  

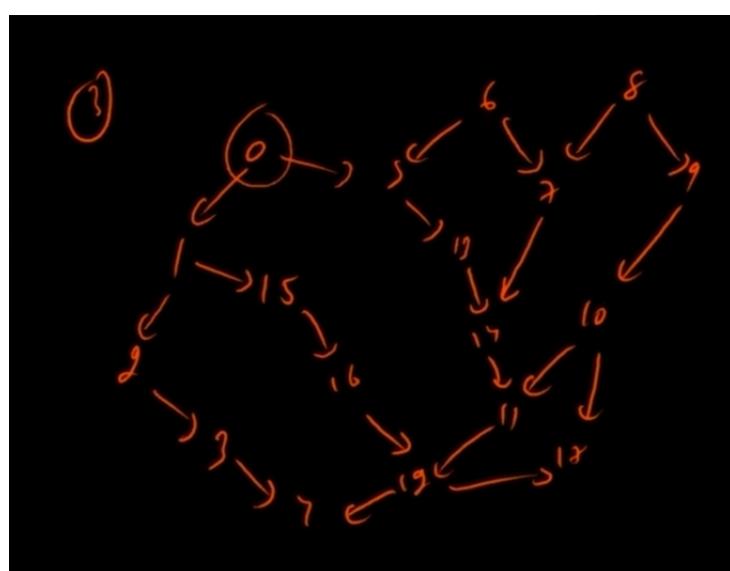
// level, they can be run parallel(simultaneously) with each other since they  

// are independent of each other considering that their dependencies are  

// resolved in the previous level. Hence we can save time for the whole process.  

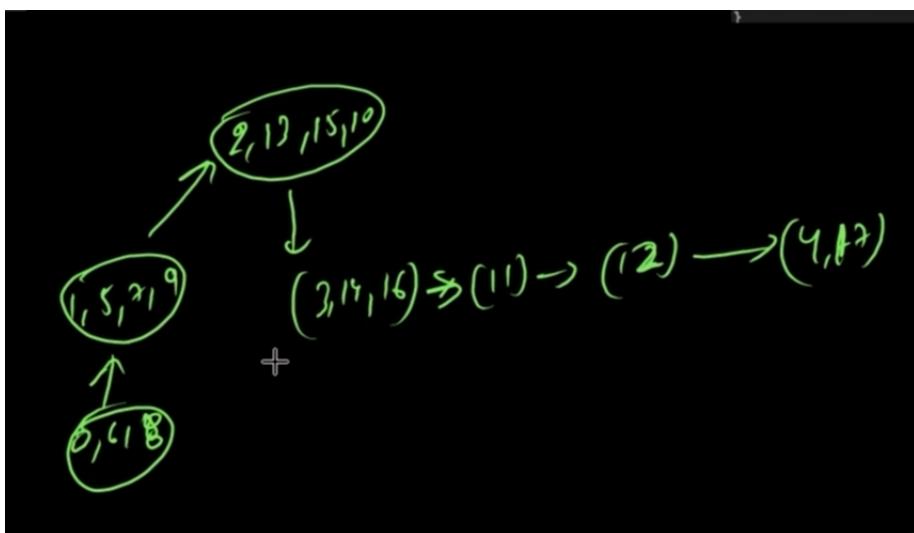
// ! Very important point, many questions can be resolved with this.  

// 
```



0	1	1	1	2	2	0	2	0	1	1	2	2	1	2	1	1	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

0	X	1	1	R	X	0	X	0	X	X	X	X	X	X	X	X	X
0	8	1	5	7	9	2	13	15	10	3	11	16	17	11	12	4	12
0	6	8	1	8	2	9	2	13	15	10	3	17	16	11	12	4	12
que	ans																



```

//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("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] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- 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 <= ai, bi < 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] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- 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 <= ai, bi < numCourses`
- `ai != bi`
- All the pairs `[ai, bi]` 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++;
    }

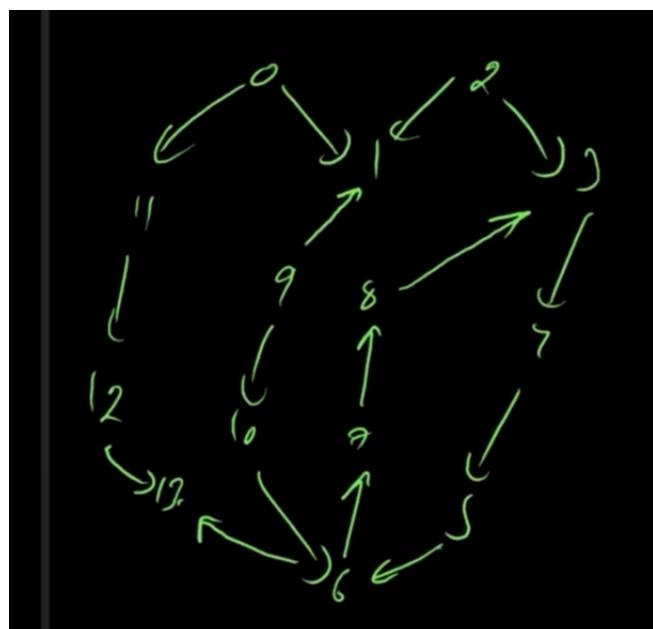
    if (ans.size() != n) {
        ans.clear();
    }
}

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



	VIS	path
0	✓	
1	✓	
2	✓	✓
3	✓	—
4	✓	—
5	✓	—
6	✓	—
7	✓	—
8	✓	—
9		
10		
11	—	
12	—	
13	—	

0	1
1	2+
2	1
3	1
4	1
5	1
6	1
7	1
8	1
9	
10	
11	2
12	2
13	2

```
//----- dfc cycle detection -----
```

```
// ----- wjs 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

Given an  $m \times n$  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:

9	9	4
6	6	8
2	1	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:

3	4	5
3	2	6
2	2	1

**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:

- $m == \text{matrix.length}$
- $n == \text{matrix[i].length}$
- $1 \leq m, n \leq 200$
- $0 \leq \text{matrix[i][j]} \leq 2^{31} - 1$

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 hogा, usme hum edges define
// karenge. Kyunki hume increasing path nikalna hai to agar koi meri adjacent
// value mujse kam hai, to wo edge use 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 karega in terms of node/vertex.

```

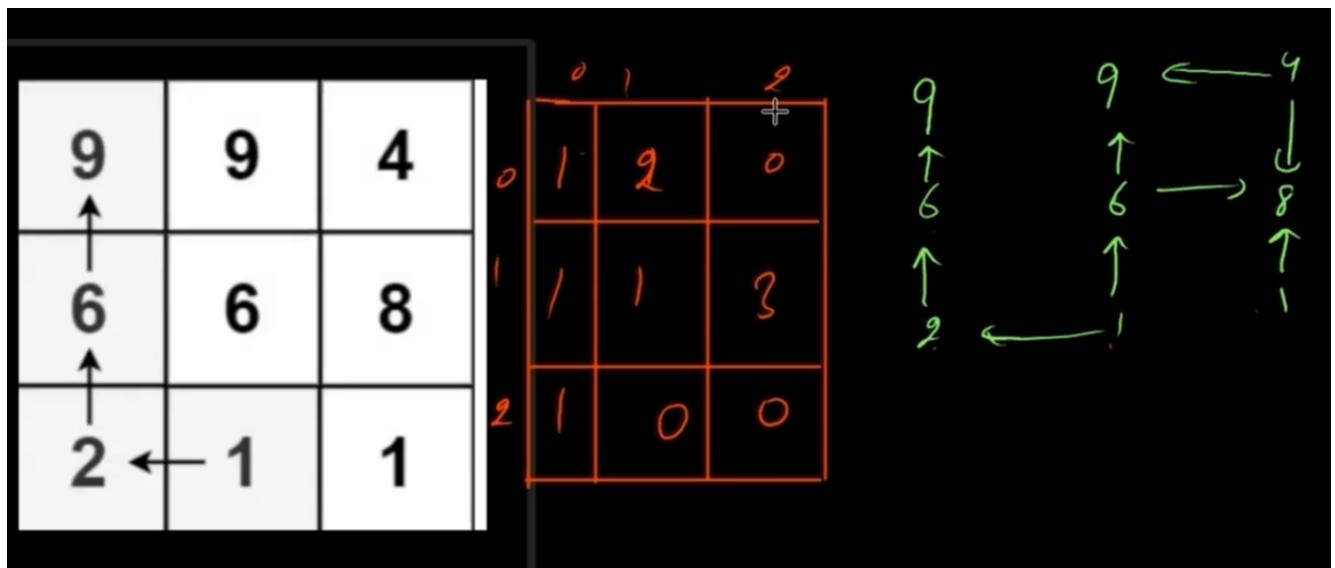
Test case to dry run on :

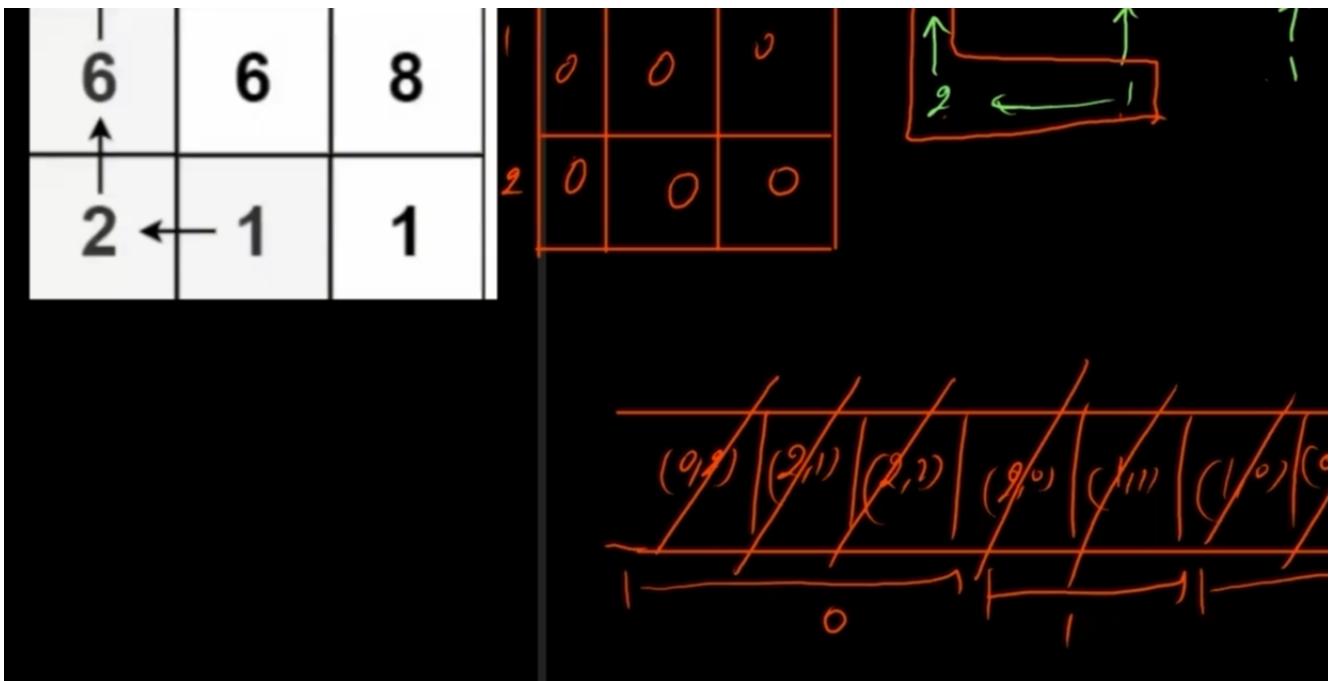
Accepted    Runtime: 0 ms

Your input    `[[0],[1],[5],[5]]`

Output    3     Diff

Expected    3





```

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);
        }
    }
}

```

```

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 wahan 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]] to
                // figure out the last
                // check
                if (--indegree[r][c] == 0)
                    que.addLast(r * m + c);
            }
        }
        level++;
    }
}

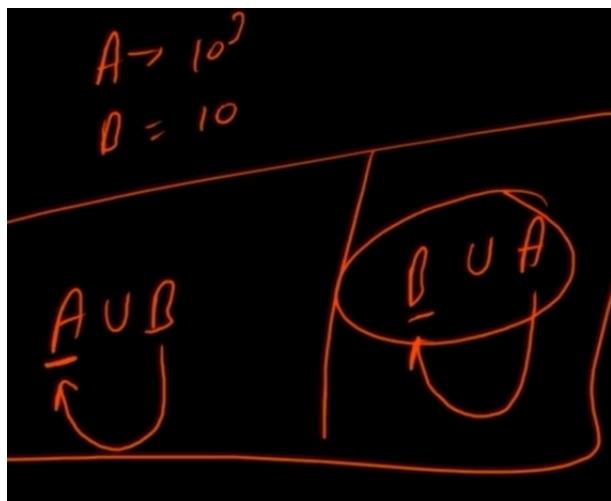
```

```

    return level;
}

```

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



```

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

// ? It works better when the edges are given. 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 a big operation and is of high complexity. So the answer remains
// the same but the complexity are different.

```

```

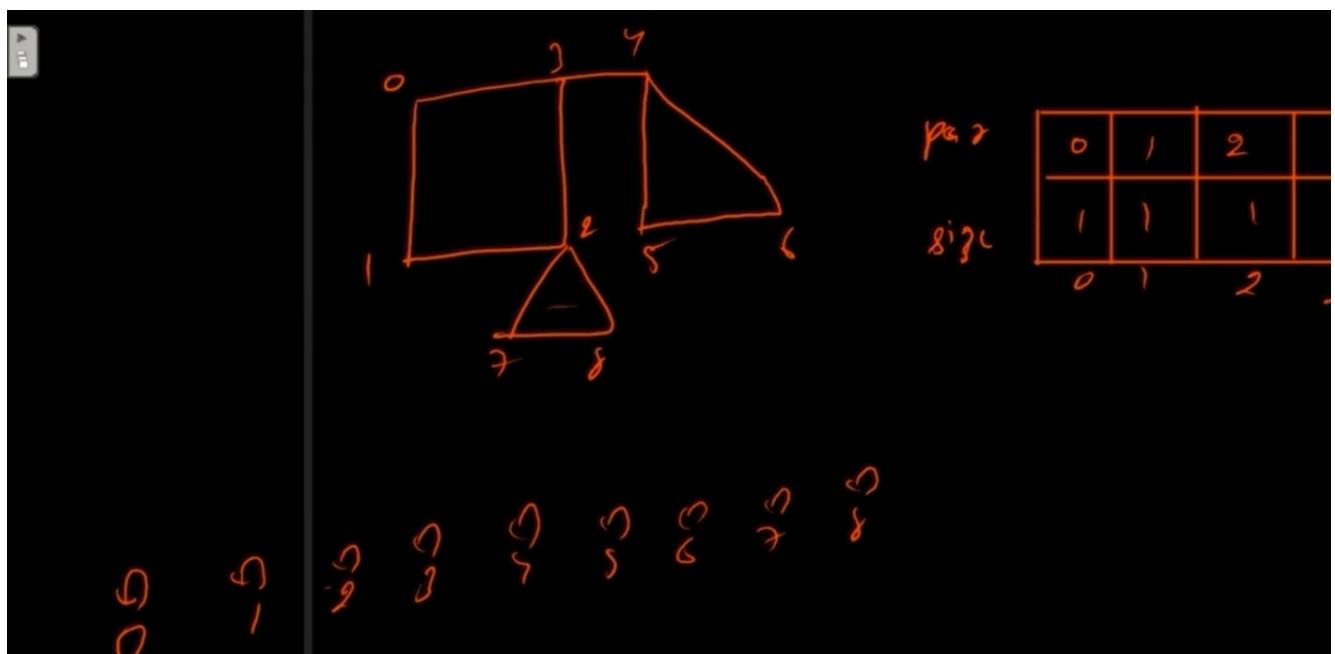
// # Ab union hogा kaise. Iske liye mai har ek set ka leader bana raha hun aur
// # jo bhi query hogi us set ke elements ki, wo leader se hogi

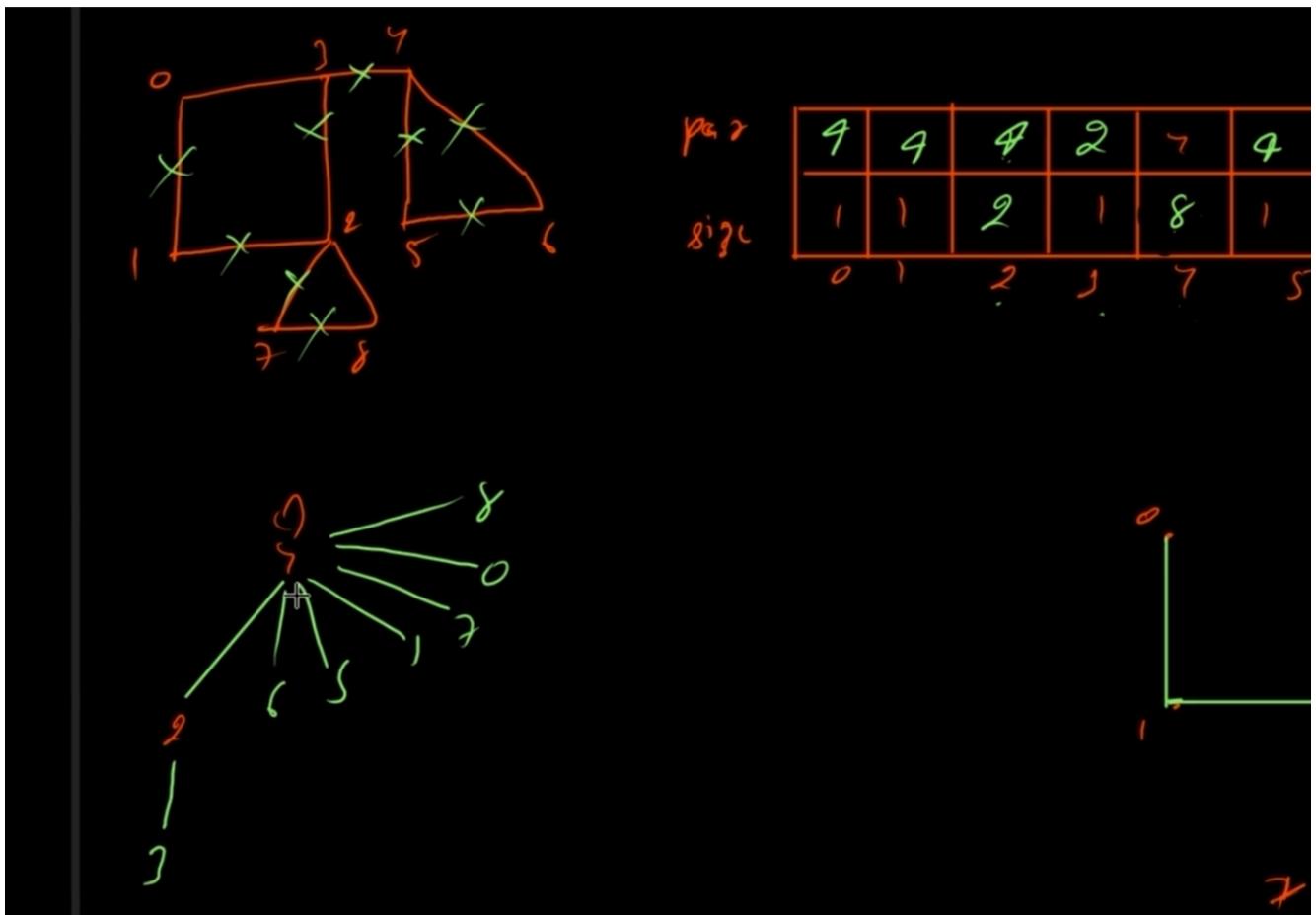
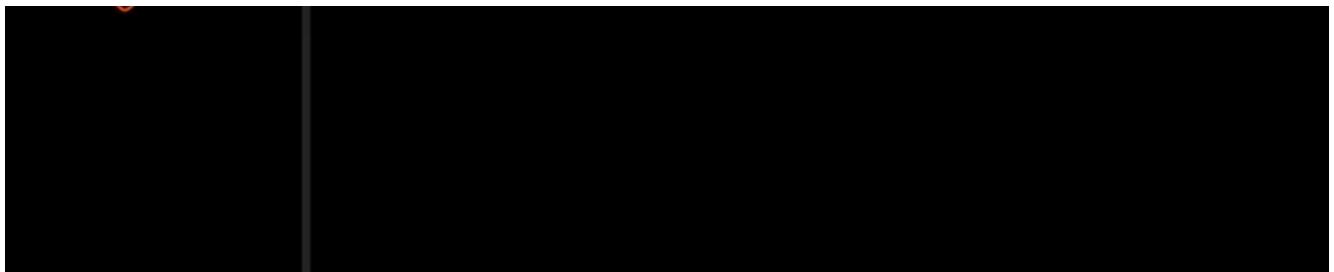
// So agar mai ek set ko country se denote karta hun to agar mai us country ka
// size puchta hun to mujhe leader answer karega. Aur koi ni.

// ! 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

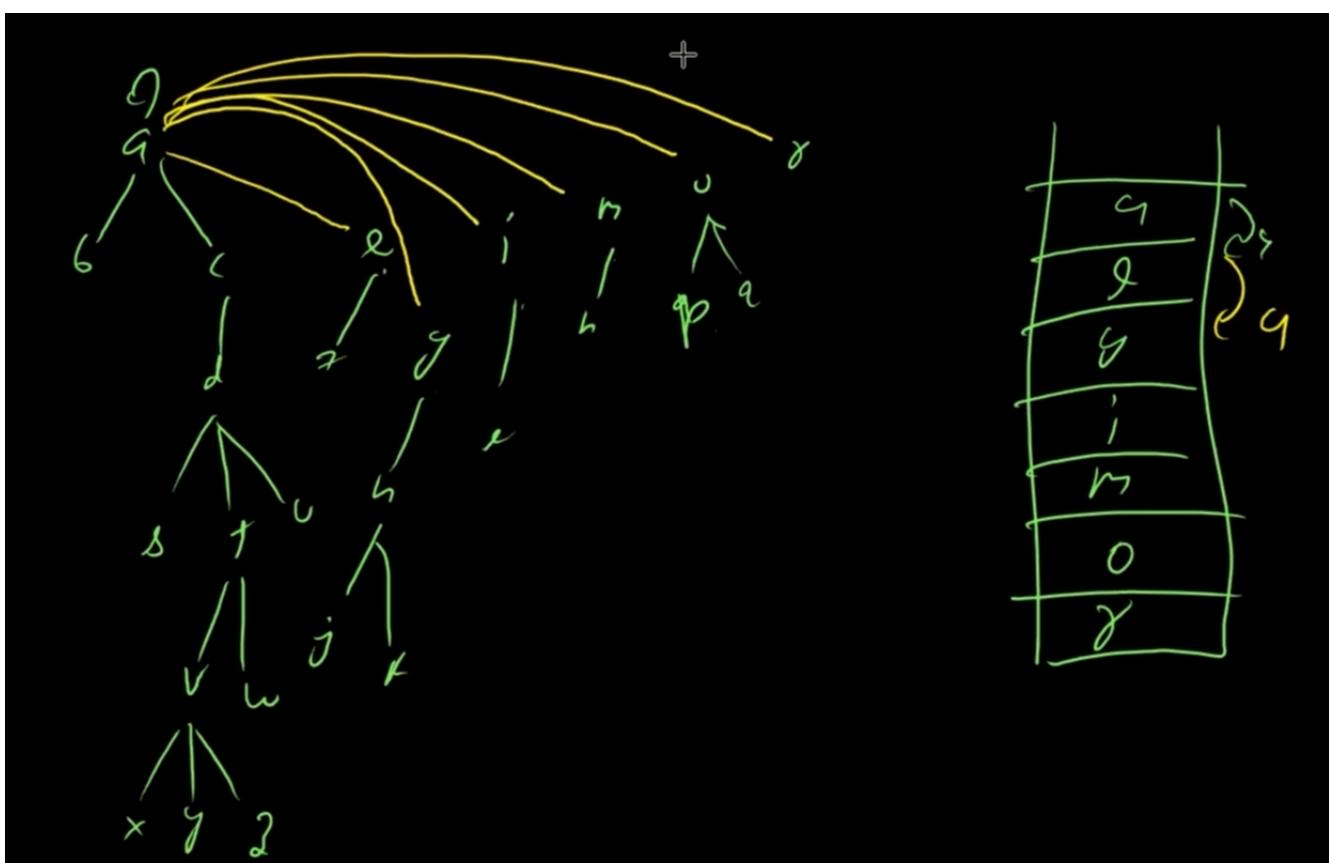
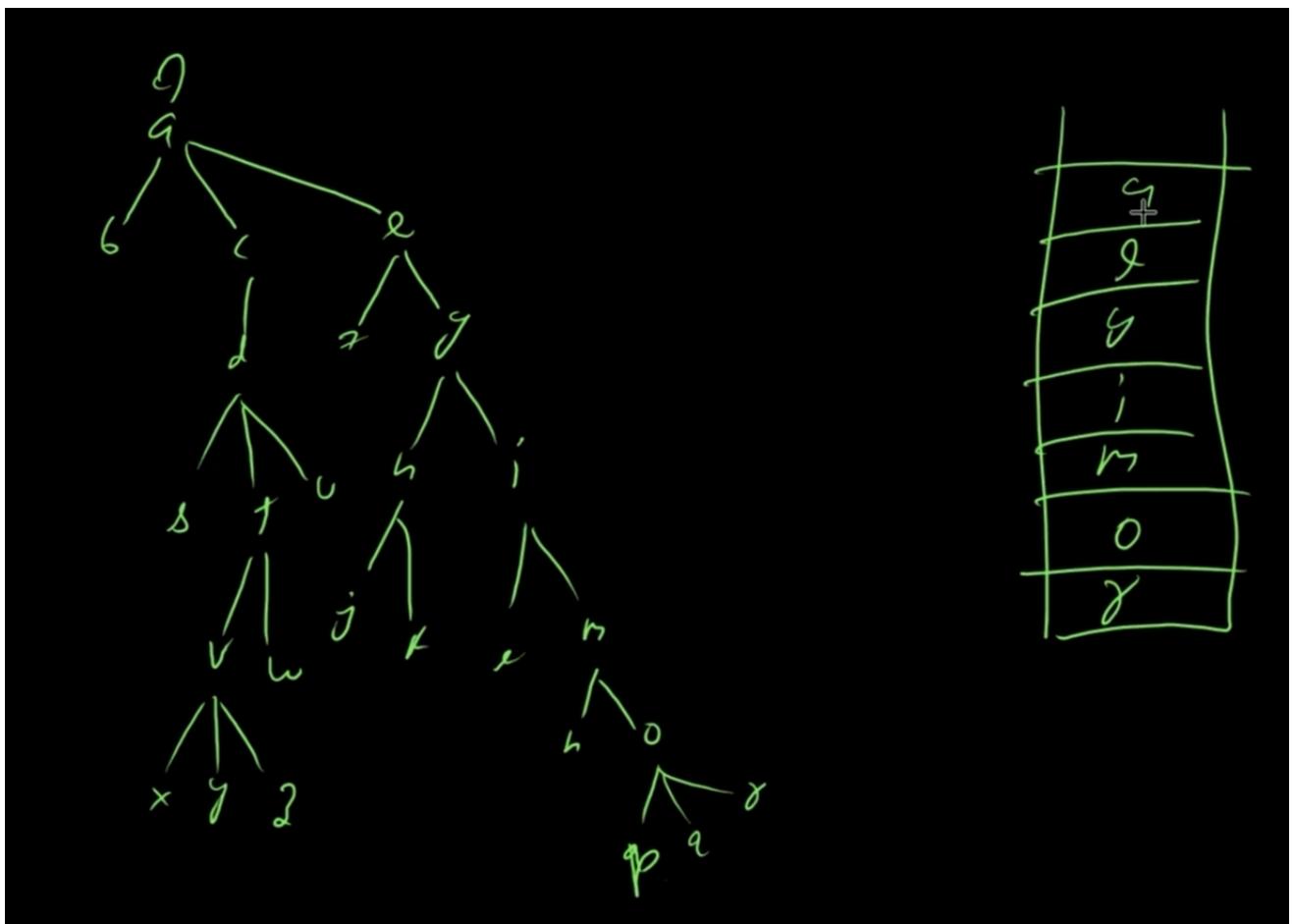
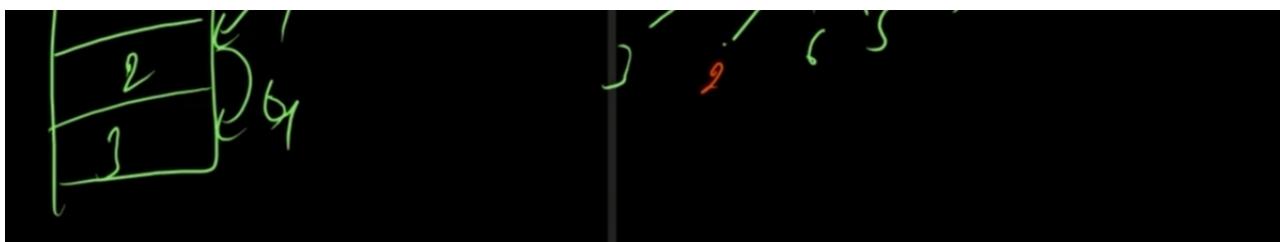
```
// # Now what we do is use path compression for to find the Leader of the
// # childs.

// Main 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 hogा
// 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)

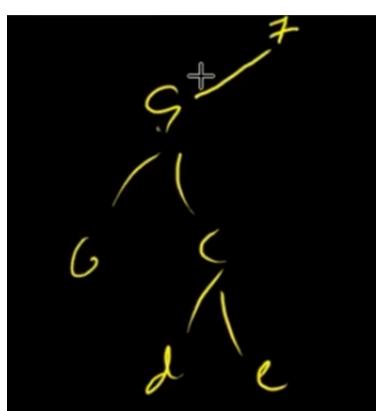
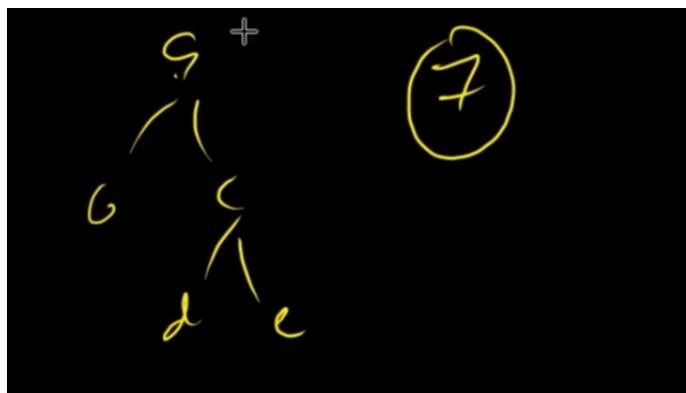
// ! Isi cheez ko path compression bolte hain. Jo humari complexity high hone se
// ! bacha leta hai.
```





```
// # 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 actually 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 leta hain (matlab tree ki height control
// karne ke liye). to in find ki complexity inti hai wo hoti hai log*(N) means
```

// Karte ke cycle, to jo jisme complexity jata hoga wo hoga log(N) means  
// amortized LogN. and this will be always less than O(4).

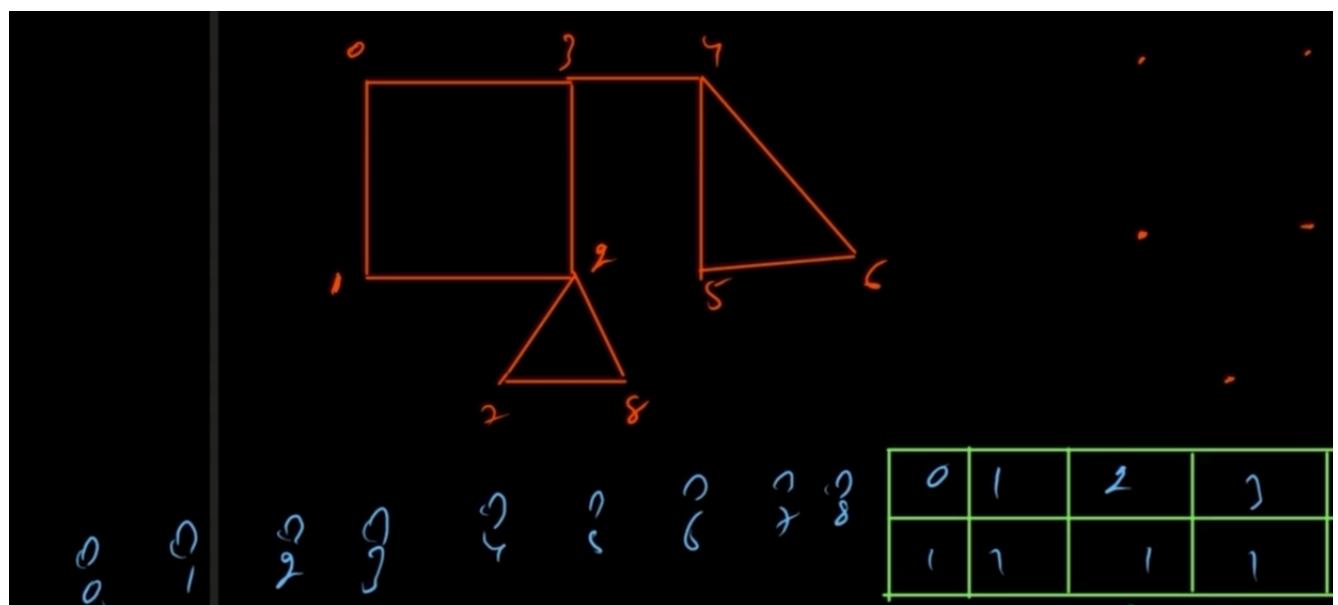
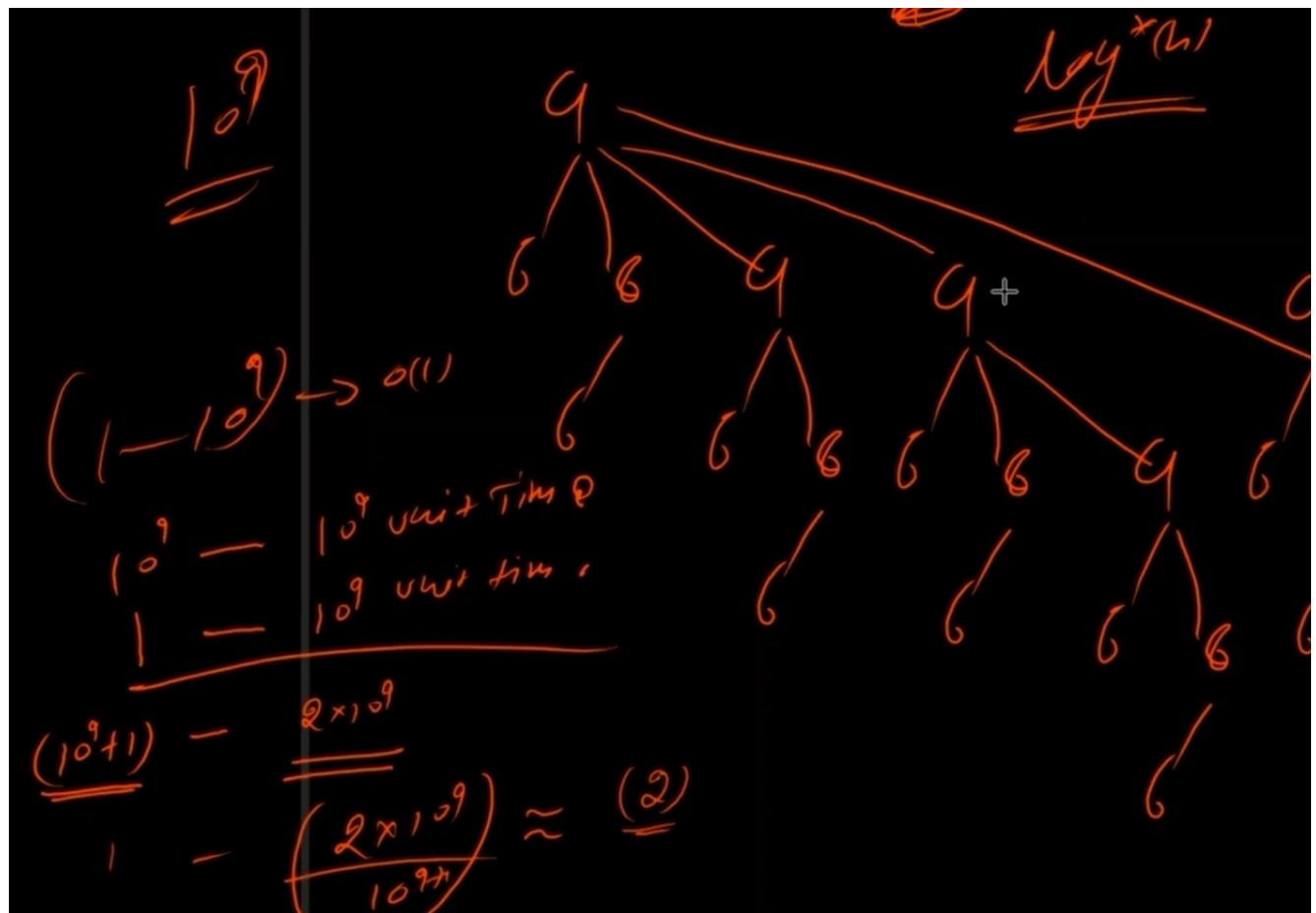
// # How less than 4 . Assuming the worst case here.

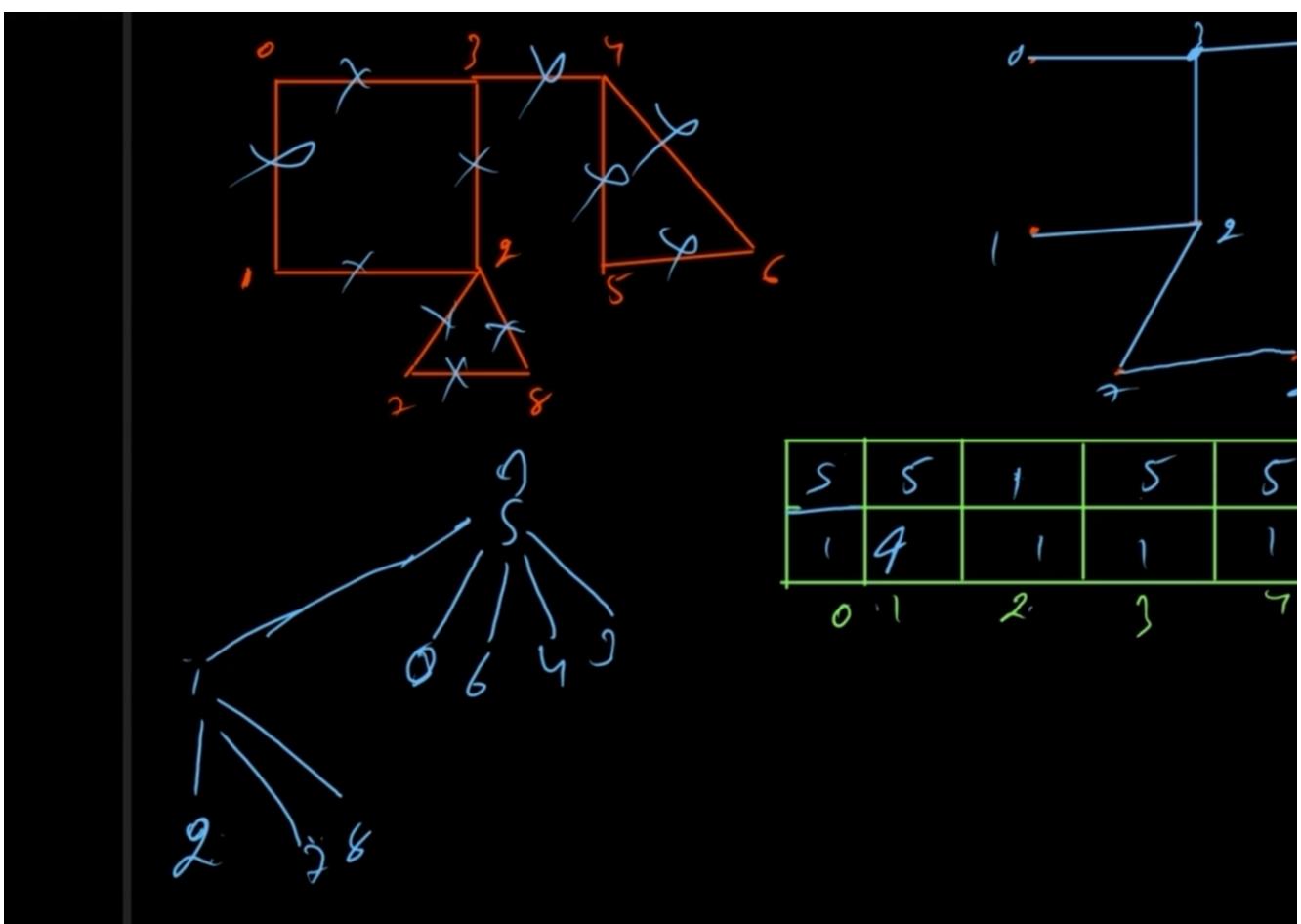
// Ab manle ki ek tree hai aue use similar tree mila, to humne jab uska union  
// kiya to simple leader ko join kar diya or ye sab O(1) mai hua. Aise he karte  
// karte agar height  $10^9$  ko reach kari to humko  $10^9$  unit of time laga tree  
// ko banane mai. Ab agar kisi leaf pe find parent ka 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 be 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).
```

```
// {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
        }
    }
}
```

```

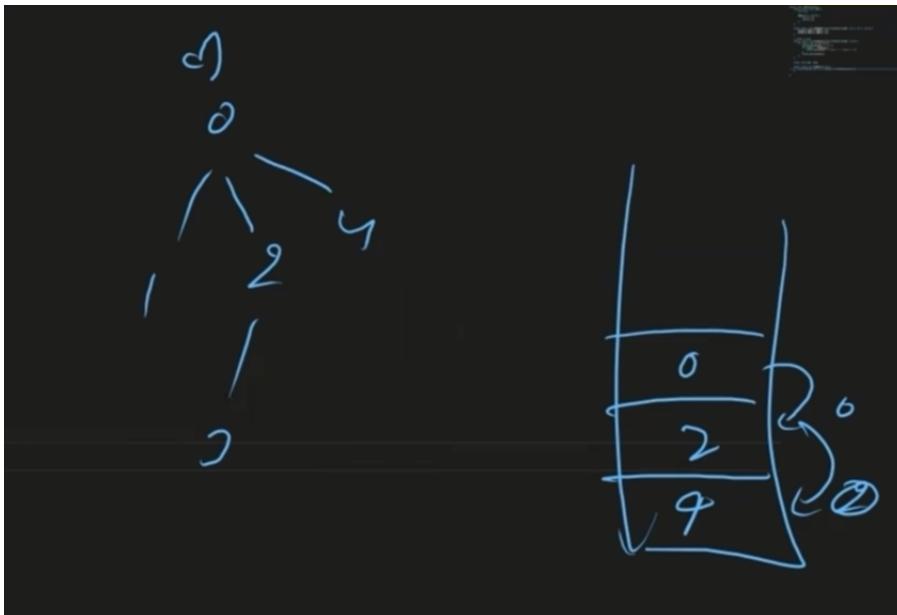
static int[] par, size;

// 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];
    }
}

```



$\mathcal{O}(V + E \log^{\chi}(h))$

```

// {{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);
        }
    }
}

```

$$V + E O(4) = V + E$$

$$= \underline{V + E \log^{\cancel{O}(n)}} \approx \underline{\underline{O(V+E)}}$$

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

## 30. 684. Redundant Connection

### 684. Redundant Connection

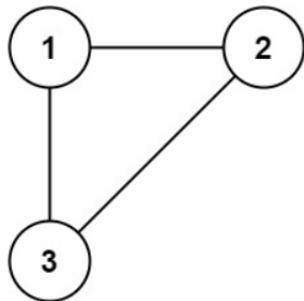
Medium    4452    317    Add to List    Share

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] = [ai, bi]` indicates that there is an edge between nodes  $a<sub>i</sub>$  and  $b<sub>i</sub>$  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.

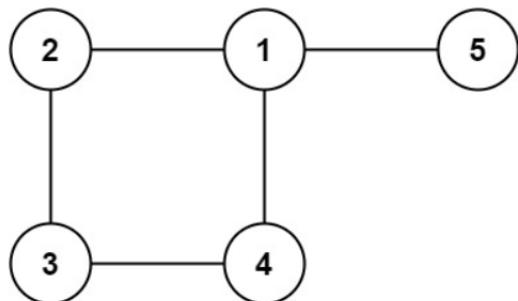
#### Example 1:



**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:

- $n == \text{edges.length}$
- $3 \leq n \leq 1000$
- $\text{edges}[i].length == 2$
- $1 \leq a_i < b_i \leq \text{edges.length}$
- $a_i \neq b_i$
- ...

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

```
// 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:**

- $1 \leq s1.length, s2.length, baseStr \leq 1000$
- $s1.length == s2.length$
- $s1, s2, \text{and } baseStr \text{ consist of lowercase English letters.}$

```
// 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 hogा, use uska Leader set kar denge to
// Lexographically order mil jayega kyunki sabse chota he chahiye 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 dhandhte 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;
}
```

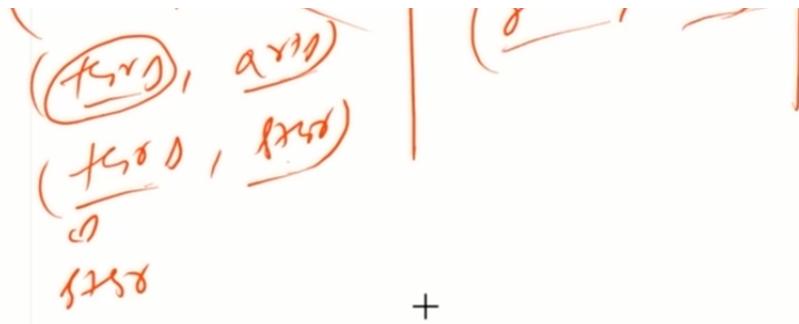
is (in different positions) of X, so that it equals  
 at positions 0 and 2), and "rats" and  
 "arts", or "arts".

```
2 *
3 *
4 *
5 *
6 *     public boolean isSimilar(String s1, String s2){
7 *
8 *         (t,a,r,s) (r,a,t,s) (r,a,t,s) (r,a,t,s)
```

(G)  
 (r,a,t,s) (r,a,t,s) (r,a,t,s) (r,a,t,s)

{"tars", "rats", "arts"} and {"star"}.  
Even though they are not similar, formally, each  
is similar to at least one other word in the group.

strs is an anagram of every other string in



### 33. Number of Islands(Union Find)

#### 200. Number of Islands

Medium 16967 389 Add to List Share

Given an  $m \times n$  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:**

- $m == \text{grid.length}$
- $n == \text{grid}[i].length$
- $1 \leq m, n \leq 300$
- $\text{grid}[i][j]$  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 bachege, utne he seprate island honge kyunki jo same
💡 // # island ko belong karte honge unka alag se set ban gaya hogा.
```

```
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 parent
                        // 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;
                }
            }
        }
    }
}

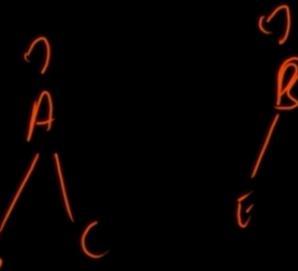
💡 // Cannot write par[p1]=p2 since agar aise Likha to pehle valid direction ke
// baad set ka Leader p2 ho jata. Agar aisa likhan hai to p1 ka duabra
// findLeaderParent call karne kee need ho jata hai
```

```

    }
}
}

return noOfSets;
}

```



`fun mergeSets(p1: Int, p2: Int): Int`

```

    var noOfSets = 1
    if (par[p1] == p1 && par[p2] == p2) {
        // If both p1 & p2 are leaders, then
        // they are in different sets. so increment
        // no of sets.
        noOfSets++
    } else {
        if (rank[p1] < rank[p2]) {
            // If p1's rank is less than p2's rank
            // then make p1 as leader and
            // attach p2 under p1.
            par[p2] = p1
            if (noOfSets == 1) {
                // If there is only one set
                // then both p1 & p2 are leaders.
                // Call finalLeaderParent and
                // update rank of p1.
                finalLeaderParent(p1)
            }
        } else {
            // If p1's rank is greater than
            // p2's rank then make p2 as
            // leader and attach p1 under p2.
            par[p1] = p2
            if (noOfSets == 1) {
                // If there is only one set
                // then both p1 & p2 are leaders.
                // Call finalLeaderParent and
                // update rank of p2.
                finalLeaderParent(p2)
            }
        }
        // If both p1 & p2 were leaders
        // then after merging them into
        // one set, they won't be leaders
        // anymore. So decrease the
        // count of sets by 1.
        noOfSets--
    }
    return noOfSets;
}

```

`finalLeaderParent(p1: Int): Int`

```

    var p1 = p1
    while (par[p1] != p1) {
        p1 = par[p1]
    }
    var p2 = p1
    while (par[p2] != p2) {
        par[p2] = p1
        p2 = par[p2]
    }
}

```

#### 34. Max Area of Island(Union Find)

##### 695. Max Area of Island

Medium 7889 176 Add to List Share

You are given an  $m \times n$  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 = [[0,0,1,0,0,0,1,0,0,0,0], [0,0,0,0,0,0,0,1,1,1,0,0], [0,1,1,0,1,0,0,0,0,0,0], [0,1,0,0,1,1,0,0,1,0,0], [0,1,0,0,1,1,1,0,1,0,0], [0,0,0,0,0,0,0,0,1,0,0], [0,0,0,0,0,0,0,1,1,0,0], [0,0,0,0,0,0,1,1,1,0,0], [0,0,0,0,0,0,0,1,0,0,0]]`  
`Output:` `6`

```
---r--- ^  
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:

- $m == \text{grid.length}$
- $n == \text{grid}[i].length$
- $1 \leq m, n \leq 50$
- $\text{grid}[i][j]$  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;  
}
```



```
// ? Ab kyunki mujhe grid2 wala island to pura mark karna he hogा, 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++;
            }
        }
    }

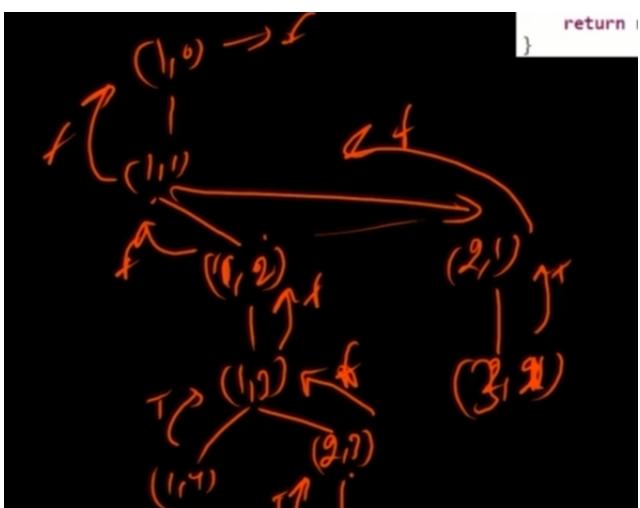
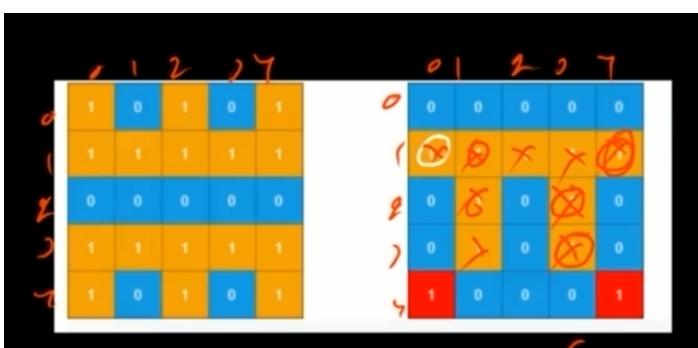
    return noOfSubIslands;
}
```

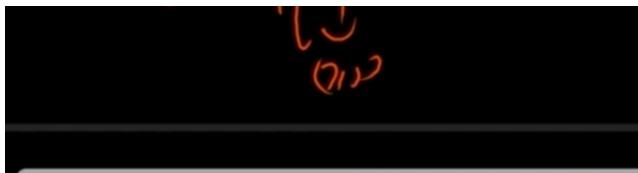
```
public static boolean dfs_countSubIslands(int[][] grid1, int[][] grid2, int[][] dir, int sr, int sc) {
    grid2[sr][sc] = 2; // Mark kiya

    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 angle
            // direction se true aay, to hume to false 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  $m \times n$ . 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).

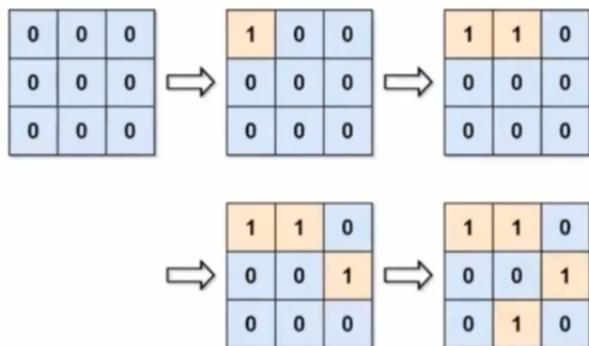
We may perform an add land operation which turns the water at position into a land. You are given an array `positions` where `positions[i] = [ri, ci]` is the position  $(r_i, c_i)$  at which we should operate the  $i^{th}$  operation.

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:

- $1 \leq m, n, \text{positions.length} \leq 10^4$
- $1 \leq m * n \leq 10^4$
- $\text{positions}[i].length == 2$
- $0 \leq r_i < m$
- $0 \leq c_i < n$

**Follow up:** Could you solve it in time complexity  $O(k \log(mn))$ , where  $k == \text{positions.length}$ ?

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

// ? Yahan pe kya 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 dekhenge, agar wo kisi aur island ka part
// ` ban saka 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;
    }

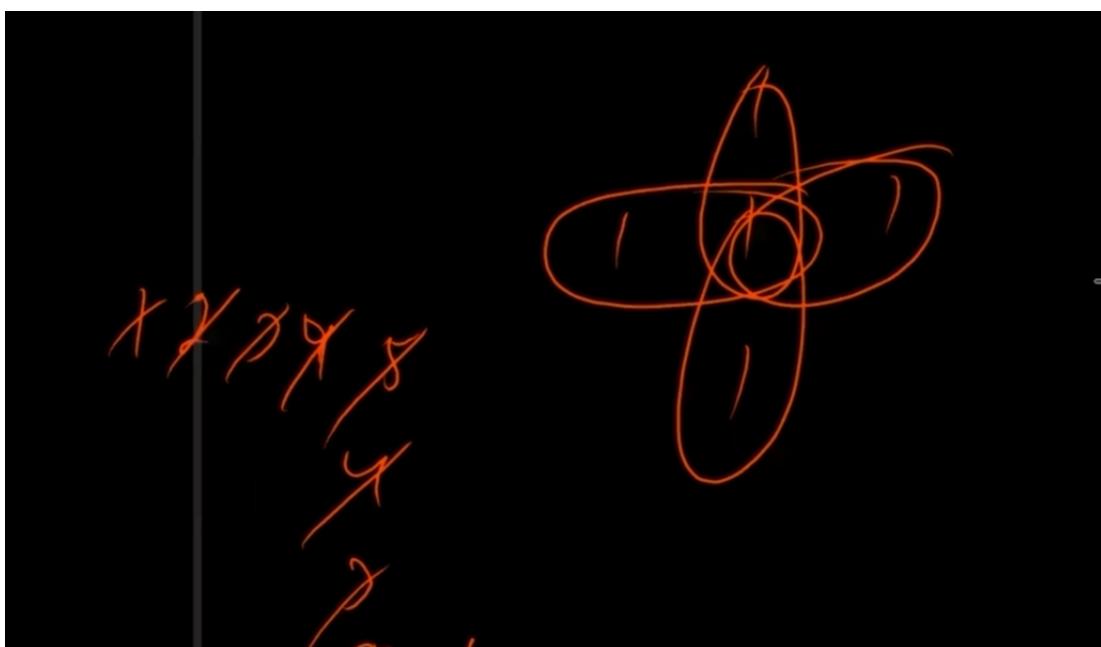
    int count = 0;
    int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };
    for (int[] p : operators) {
        int i = p[0], j = p[1];

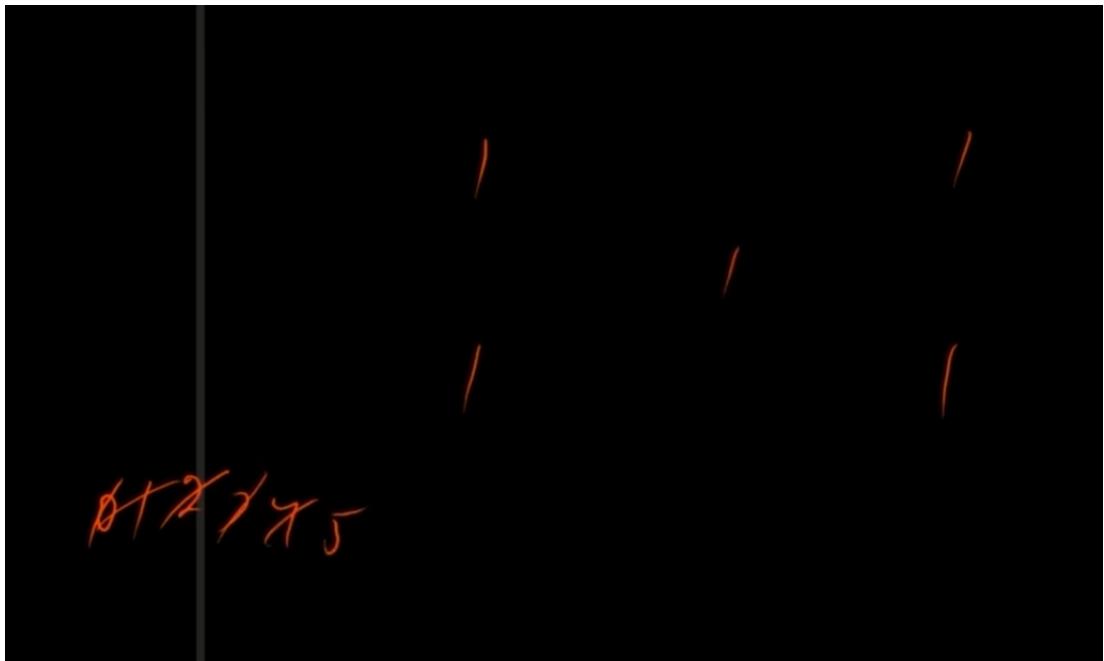
        if (grid[i][j] == 1) { // if there is already a Land where we are going then nothing should be updated.
            ans.add(count);
            continue;
        }
        grid[i][j] = 1;

        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;
}
```





```
// # 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)

scality	comparator/comparable/lambda	output
$q > b$	a - b	cine, $q > b$ , $q =$
$q \leq b$	a - b	ave, $q \leq b$ , $q =$
$q > b$	b - q	cine, $q < b$ , $\leftarrow$
$q \leq b$	b - q	cine, $q > b$ , $q =$

```

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 behaviour
        return b[2] - a[2]; // other - this, reverse of default behaviour
    });

    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

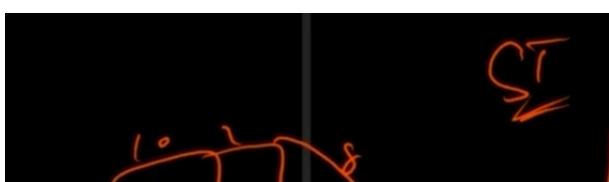
```

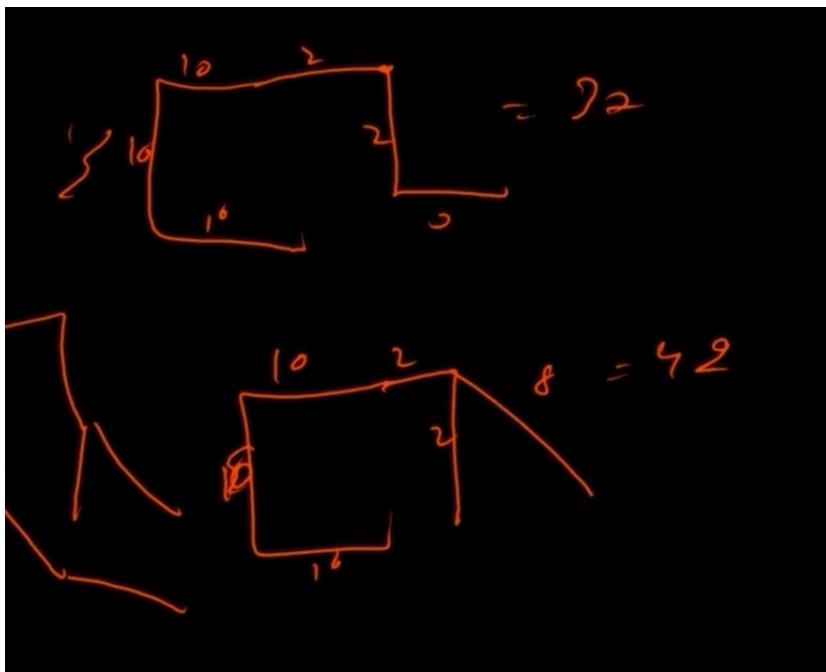
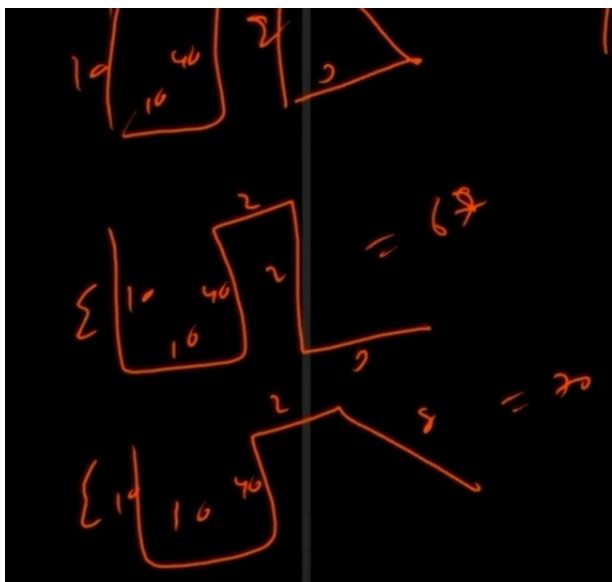
// B =====MST(Minimum Spanning Tree) =====
// Aisa spanning tree, jiska edges ki weight ka sum minimum ho, use bolte hain
// minumum spanning tree.

// Ek graph ke to bahut sare spanning tree ban sakte hain, to jo mimimum wala
// hogा 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.

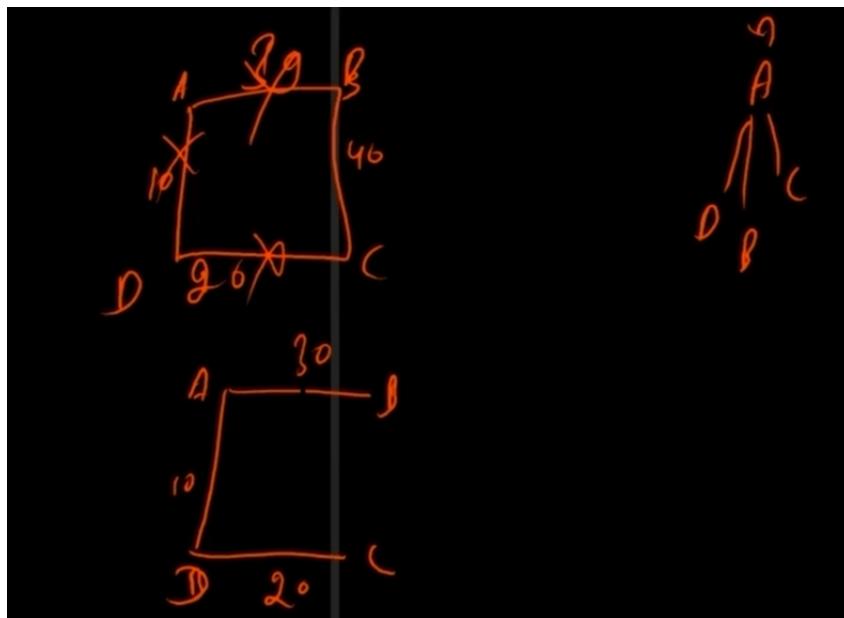
// Ek graph ke to bahut sare spanning tree ban sakte hain, to jo mimimum wala
// hogya with edges weight is called MST.

// ! 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 responsible 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 edges 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<sup>[1]</sup> 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.<sup>[2]</sup>

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

### 40. Optimize Water Distribution in a Village

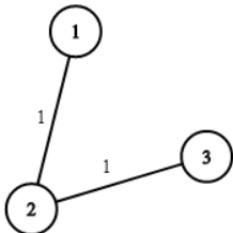
## 1168. Optimize Water Distribution in a Village

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 leta 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] });
    }

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

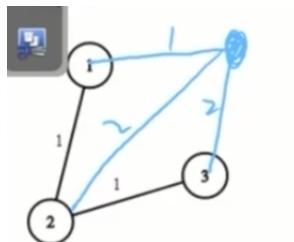
    // ! Baki pura union find algo hai
    par = new int[n + 1];
    for (int i = 0; i < n + 1; i++) {
        par[i] = i;
    }
    int minCost = 0;
    for (int[] edge : edges) {

        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;
}

```

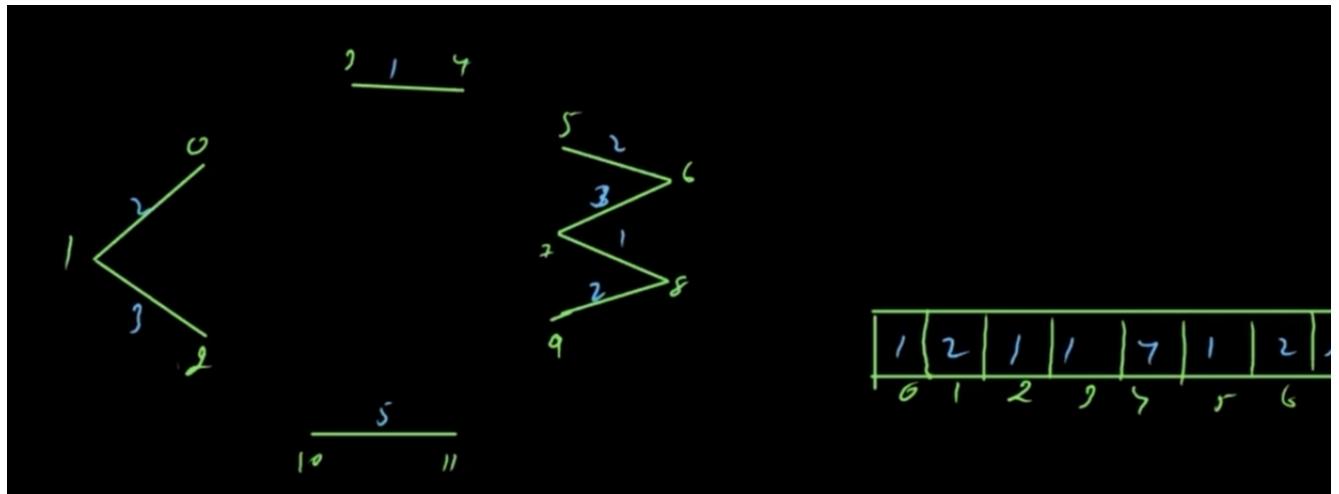


Input:  $n = 3$ , wells = [1,2,2], pipes = [[1,2,1],[2,3,1]]

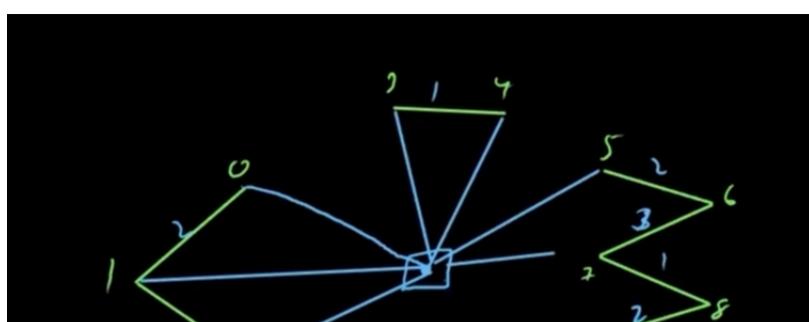
Output: 3

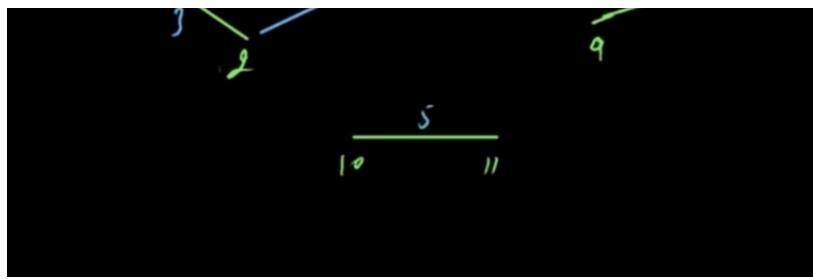
Explanation:

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.

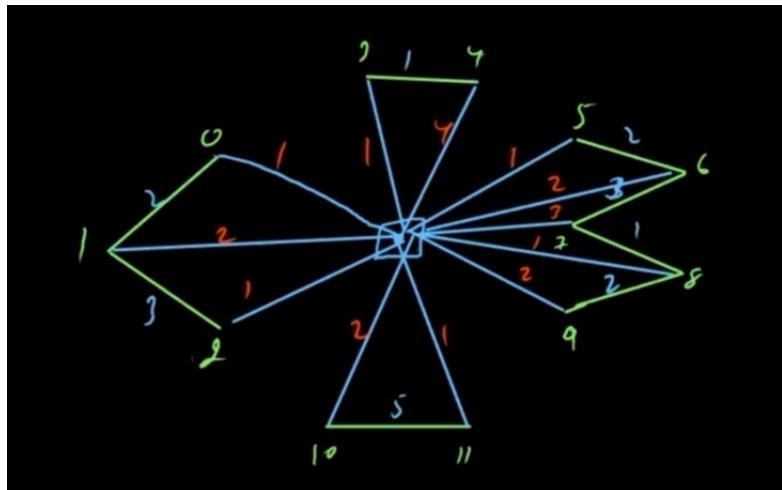


Pehle graph connected nhi tha

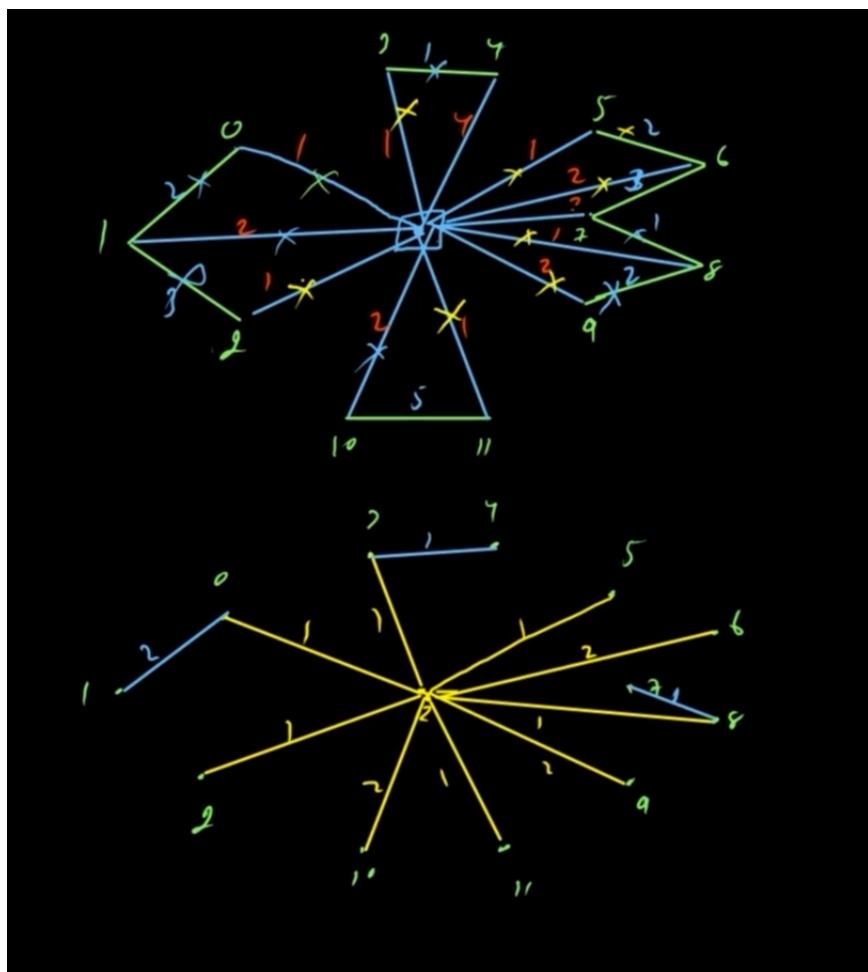




Ab connected hai to MST laga sakte hain



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 save 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:

$2 \leq N, M \leq 10^6$

$0 \leq K \leq 10^{18}$

$1 \leq A, B \leq N$  and  $A \neq B$

$1 \leq C \leq 10^6$

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.

### Contributors:



Pawe? Kacprzak



Bohdan Pryshchenko

```

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

// ! 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

```

```

public static int mr_president() {

    // N = > Cities
    // M => bidirectional Roads
    // K => Maintenance 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;
    }
}

```

```

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];

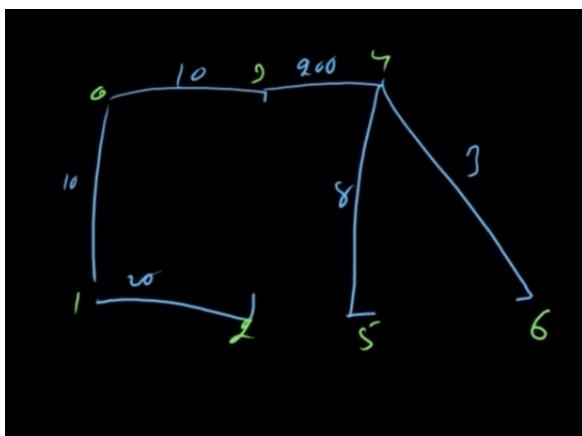
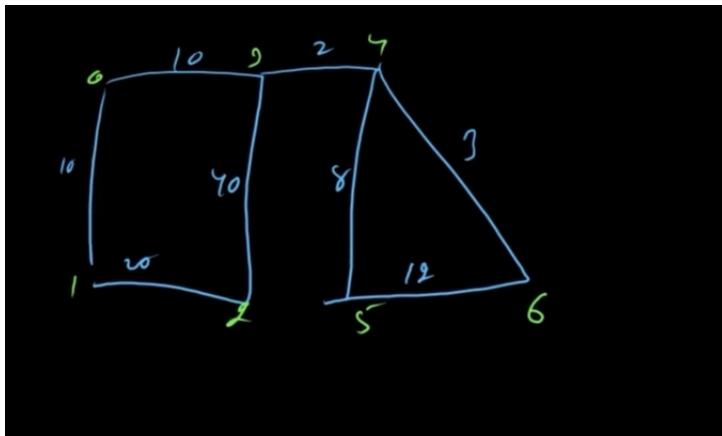
    int p1 = findLeaderParent(u);
    int p2 = findLeaderParent(v);

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

if (numberOfComponents > 1) { // Agar graph mai he number of components 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;
}

```



$$10 \quad 10 \quad 10 \quad 20 \quad 20 \quad 20 \quad 20 \quad \Sigma = 30$$

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

30

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

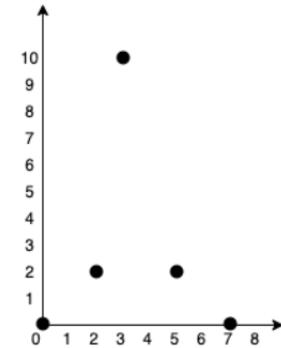
You are given an array `points` representing integer coordinates of some points on a 2D-plane, where `points[i] = [xi, yi]`.

The cost of connecting two points `[xi, yi]` and `[xj, yj]` is the **manhattan distance** between them:  $|x_i - x_j| + |y_i - y_j|$ , 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.

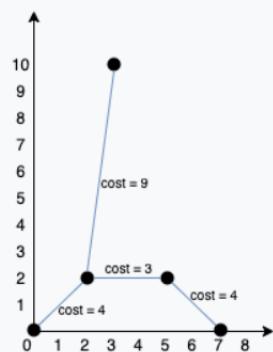
### Example 1:



**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:

- $1 \leq \text{points.length} \leq 1000$
- $-10^6 \leq x_i, y_i \leq 10^6$
- All pairs  $(x_i, y_i)$  are distinct.

💡 // ! 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.

```
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);

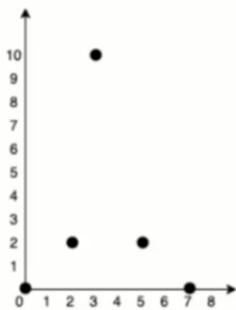
        if (p1 != p2) {
            par[p2] = p1;
            minCost += w;
        }
    }

    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.

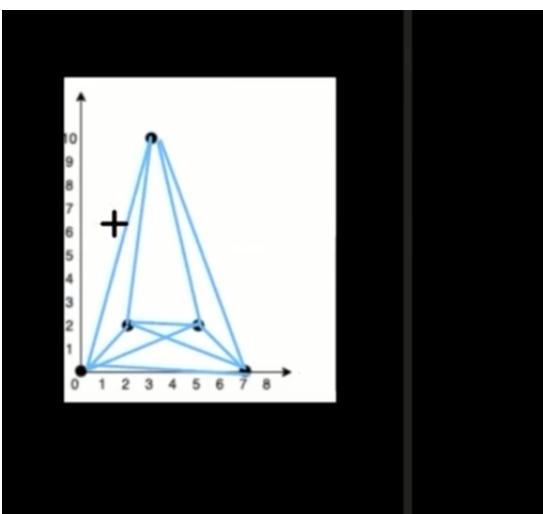
**Example 1:**



12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31

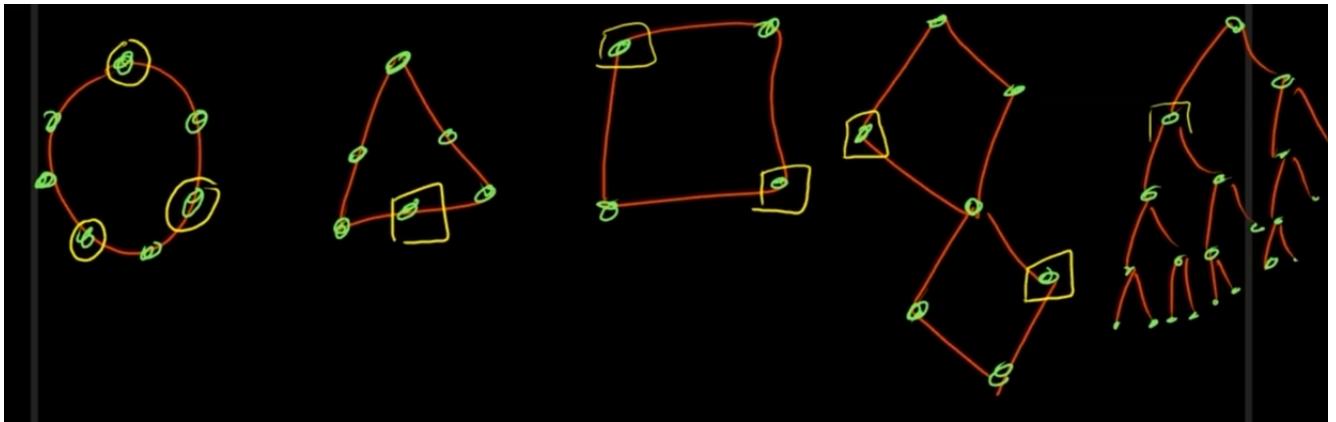
8 9 2 3 7

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 sakte 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 malware
    // 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

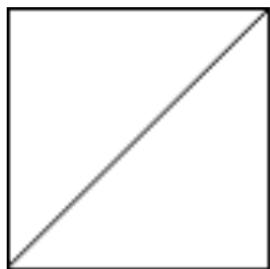
Medium    2431    465    Add to List    Share

An  $n \times n$  grid is composed of  $1 \times 1$  squares where each  $1 \times 1$  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.

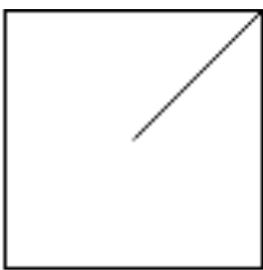
Note that backslash characters are escaped, so a `'\'` is represented as `'\\'`.

##### Example 1:

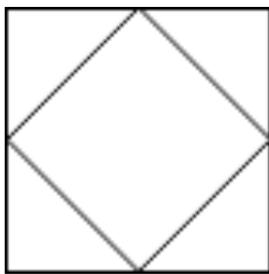


**Input:** grid = [" /","/ "]  
**Output:** 2

##### 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 hogा

// ? 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 storing 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;
    }
}

```

```

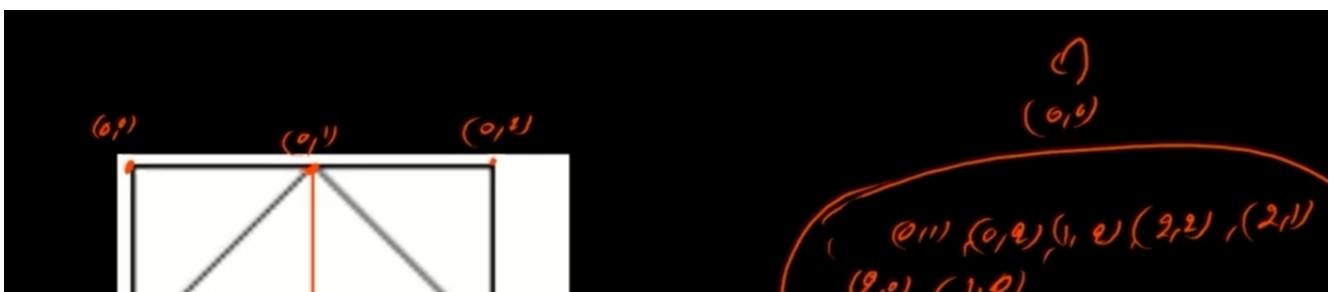
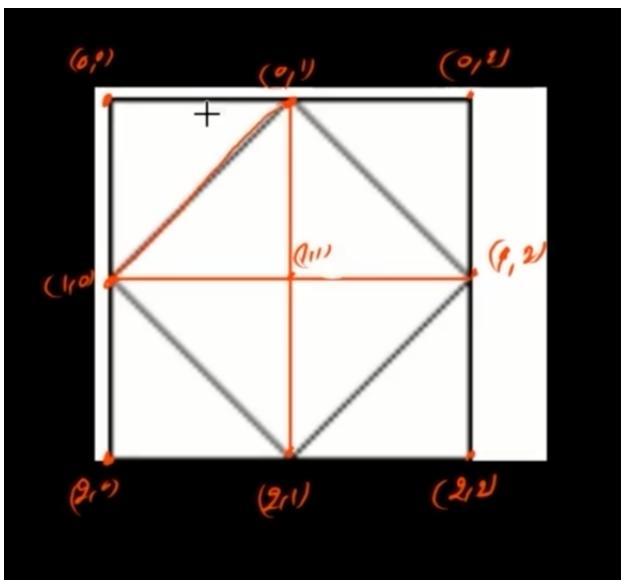
public int regionsBySlashes(String[] grid) {
    int n = grid.length;
    int N = n + 1;

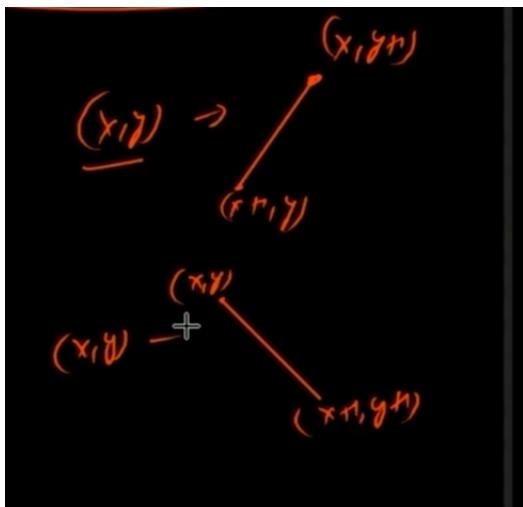
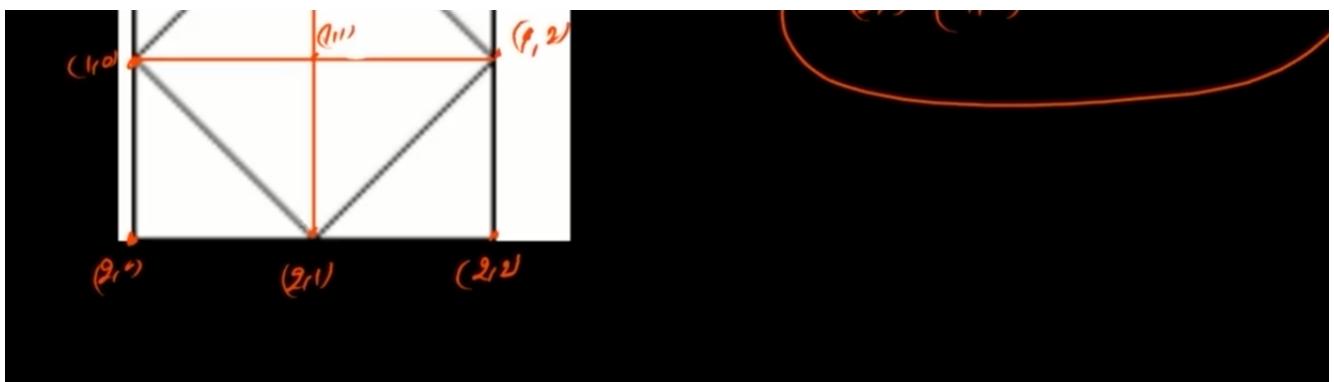
    par = new int[N * N];
    for (int i = 0; i < N * N; i++) {
        if (i / N == 0 || i % N == 0 || i / N == N - 1 || i % N == N - 1) { // Sare boundary elements ka parent 0 ko
        // maan liya kyunki wo ek sath connected
        // hain
        par[i] = 0;
    } else {
        par[i] = i;
    }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

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);
        }
    }
}

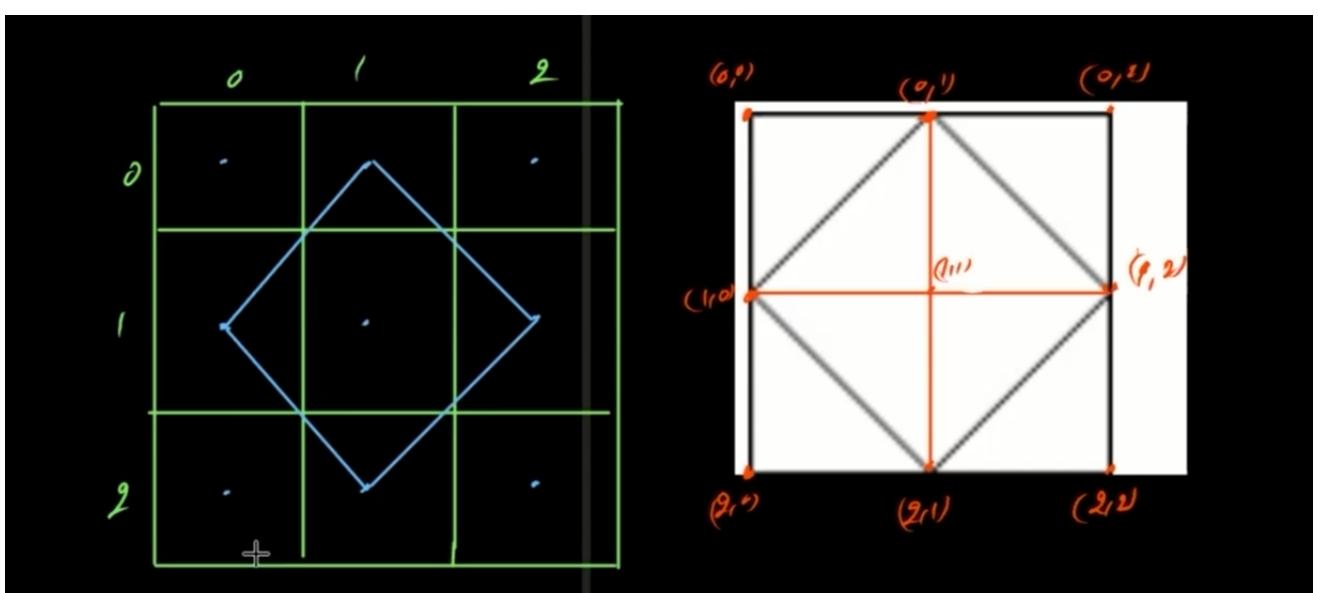
return numberOfCycles;
}

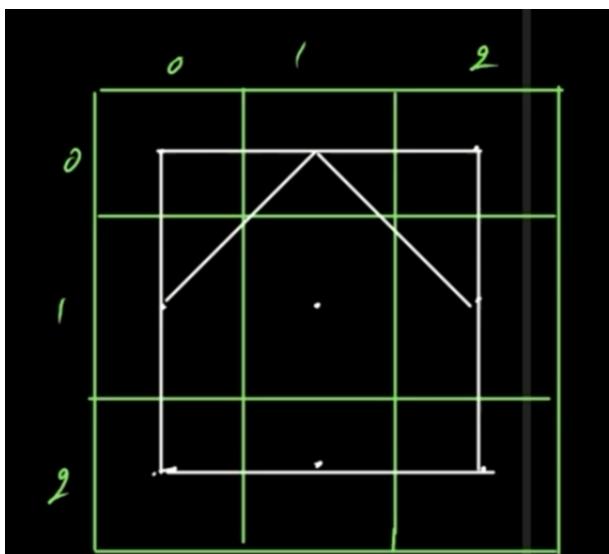
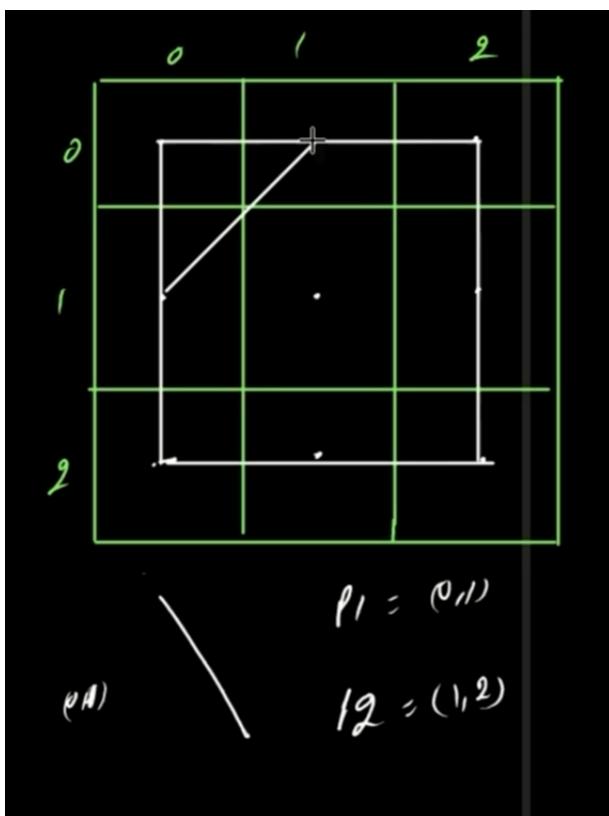
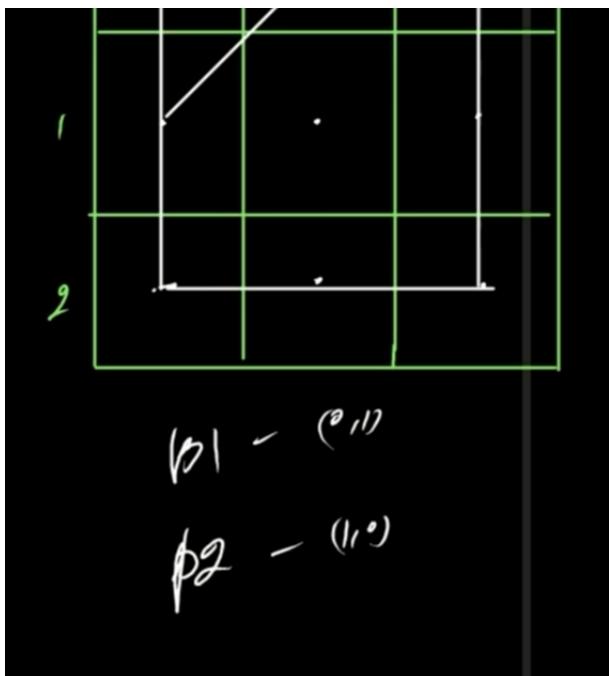
```

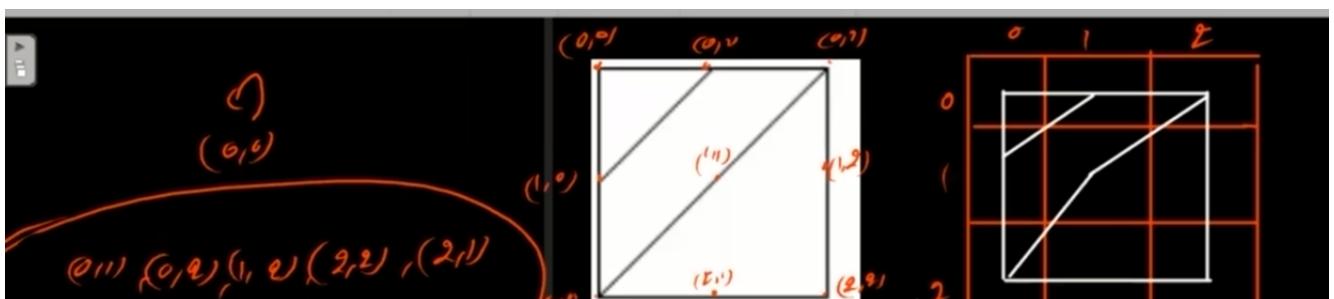
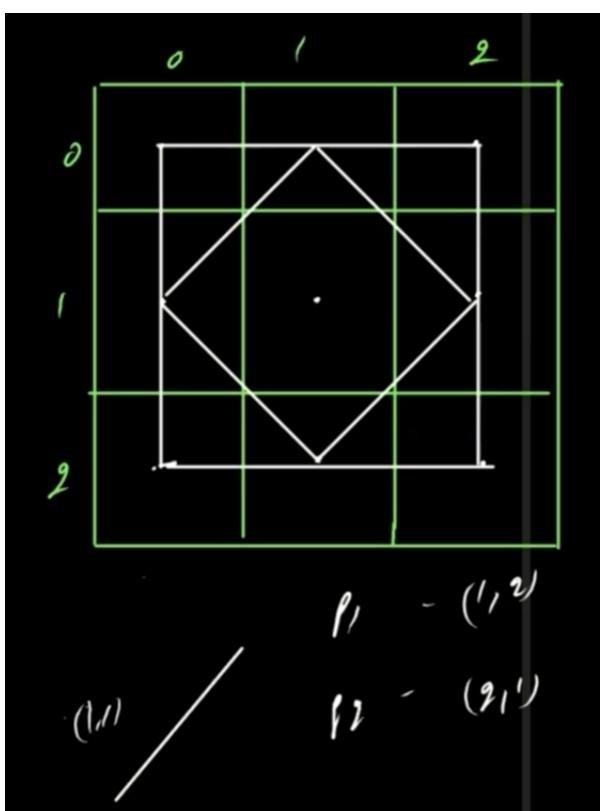
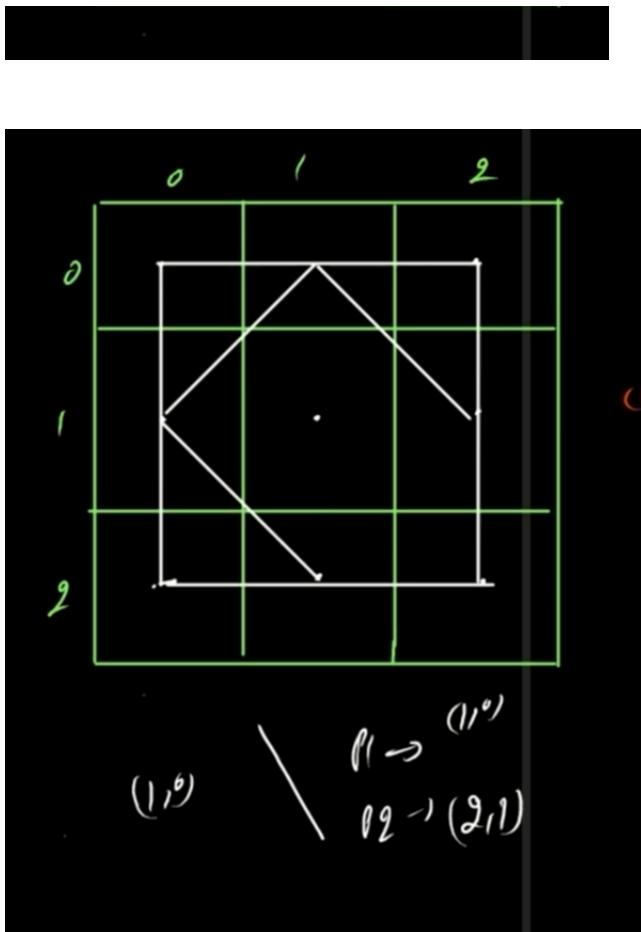


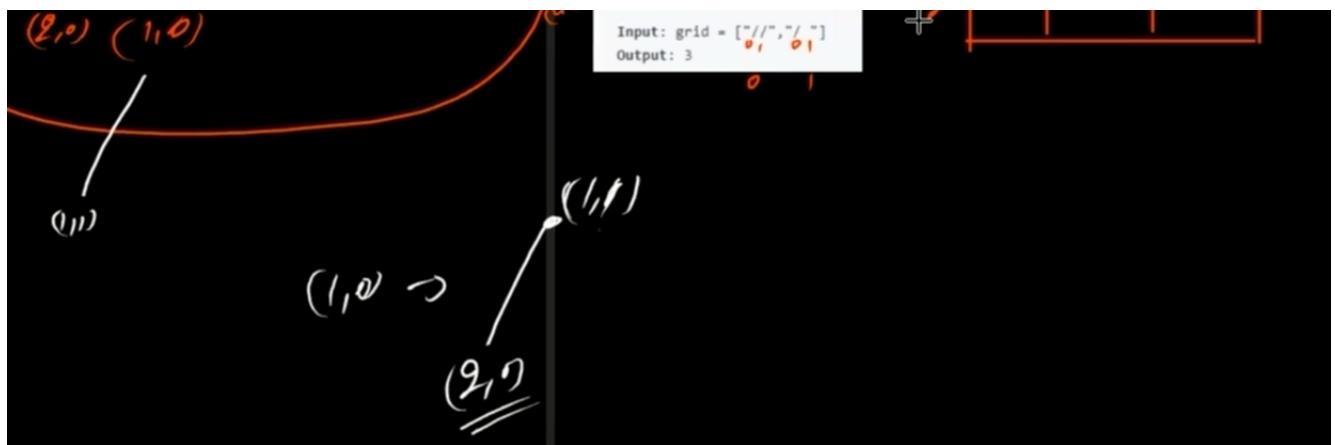


```
grid = ["/\\", "\\\\"]
```









## 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: " $x_i == y_i$ " or " $x_i != y_i$ ". Here,  $x_i$  and  $y_i$  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:

- $1 \leq \text{equations.length} \leq 500$
- $\text{equations}[i].length == 4$
- $\text{equations}[i][0]$  is a lowercase letter.
- $\text{equations}[i][1]$  is either '=' or '!'.
- $\text{equations}[i][2]$  is '='.
- $\text{equations}[i][3]$  is a lowercase letter.

💡 // 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 millega 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. Dijkstra

```

// b <<<===== Dijkstra(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(Dijkstra) 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 dijkstra(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(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;

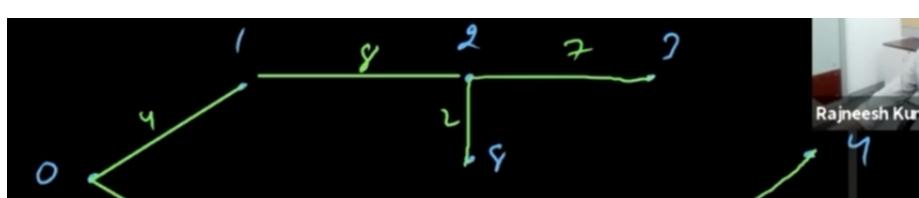
    // dijkstra
    pair(int src, int par, int w, int wsf) {
        this.src = src;
        this.par = par;
        this.w = w;
        this.wsf = wsf;
    }

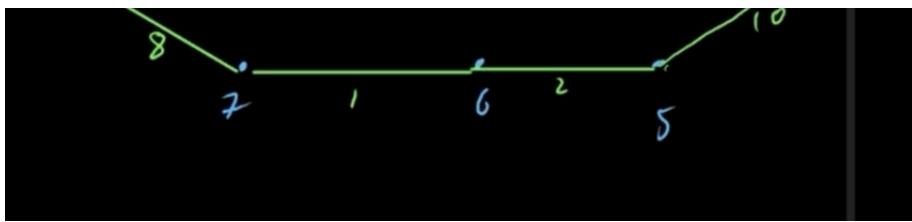
    // This one to be used in prim's algo
    pair(int src, int par, int w) {
        this(src, par, w, wsf: 0); // constructor chaining
    }
}

```

The image shows a digital whiteboard interface with the following elements:

- Left Panel:** A large oval containing handwritten notes: "src", "par", "w", and "wsf".
- Middle Panel:** A graph with 9 nodes (0-8) and various edges with weights. Node 0 is the source (marked with a blue arrow). Edges include (0,1:w4), (0,7:w8), (1,2:w8), (1,7:w11), (2,3:w7), (2,8:w2), (3,4:w9), (3,5:w14), (4,5:w10), (5,6:w2), (6,7:w1), (6,8:w6), and (7,8:w7).
- Right Panel:** A diagram showing node 0 connected to nodes 1, 2, and 7. The edges are labeled with weights: (0,1:w4), (0,2:w8), and (0,7:w2).
- Bottom Left:** A table with columns "v[i]" and "j". The rows are numbered 0 to 8. The "v[i]" column has checkmarks in the first seven rows. The "j" column has checkmarks in the first four rows.
- Bottom Right:** Handwritten notes: "7, 1, 11, 15" in a green oval, "2, 2, 8" in a green oval, and "8, 7, 2, 15" crossed out in a green oval.
- Bottom Center:** A toolbar with various icons: eraser, highlighter, selection, zoom, etc.





## 47. Network Delay Time

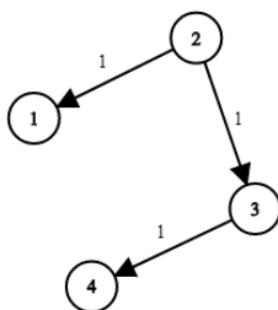
### 743. Network Delay Time

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  $\text{times}[i] = (u_i, v_i, w_i)$ , where  $u_i$  is the source node,  $v_i$  is the target node, and  $w_i$  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:

- $1 \leq k \leq n \leq 100$
- $1 \leq \text{times.length} \leq 6000$
- $\text{times}[i].length == 3$
- $1 \leq u_i, v_i \leq n$
- $u_i \neq v_i$
- $0 \leq w_i \leq 100$
- All the pairs  $(u_i, v_i)$  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.
```

```
....
```

```

// ! 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];
    });

```

```

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

```

// b <===== Dijkstra Better =====>
// Jaise he mai elements ko queue mai dalunga, waise he mai apna distance ka
// array update marunga

// Initially mera distance ka array 0 hogा. 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 rakhni hai wo usse better hai ki nhi

// ? So basically ye distance ka array kuch elements jinki need nhi hai unko
// ? dalne ne nhi data 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

```

```

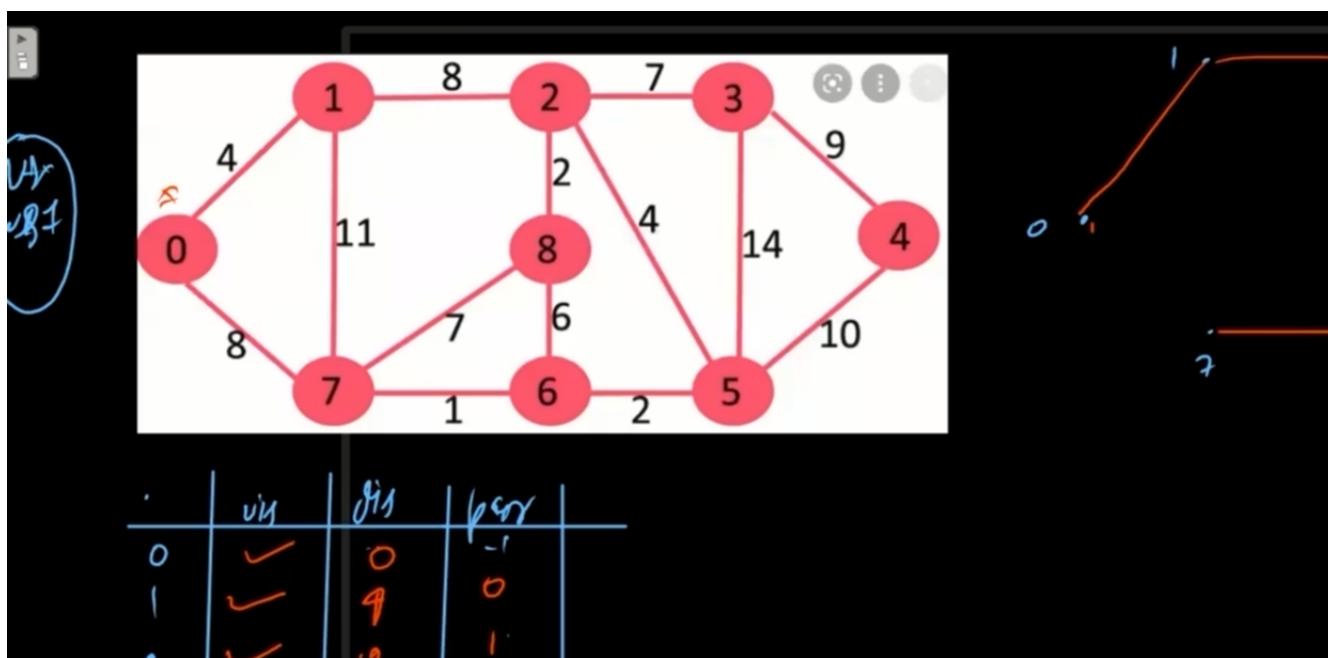
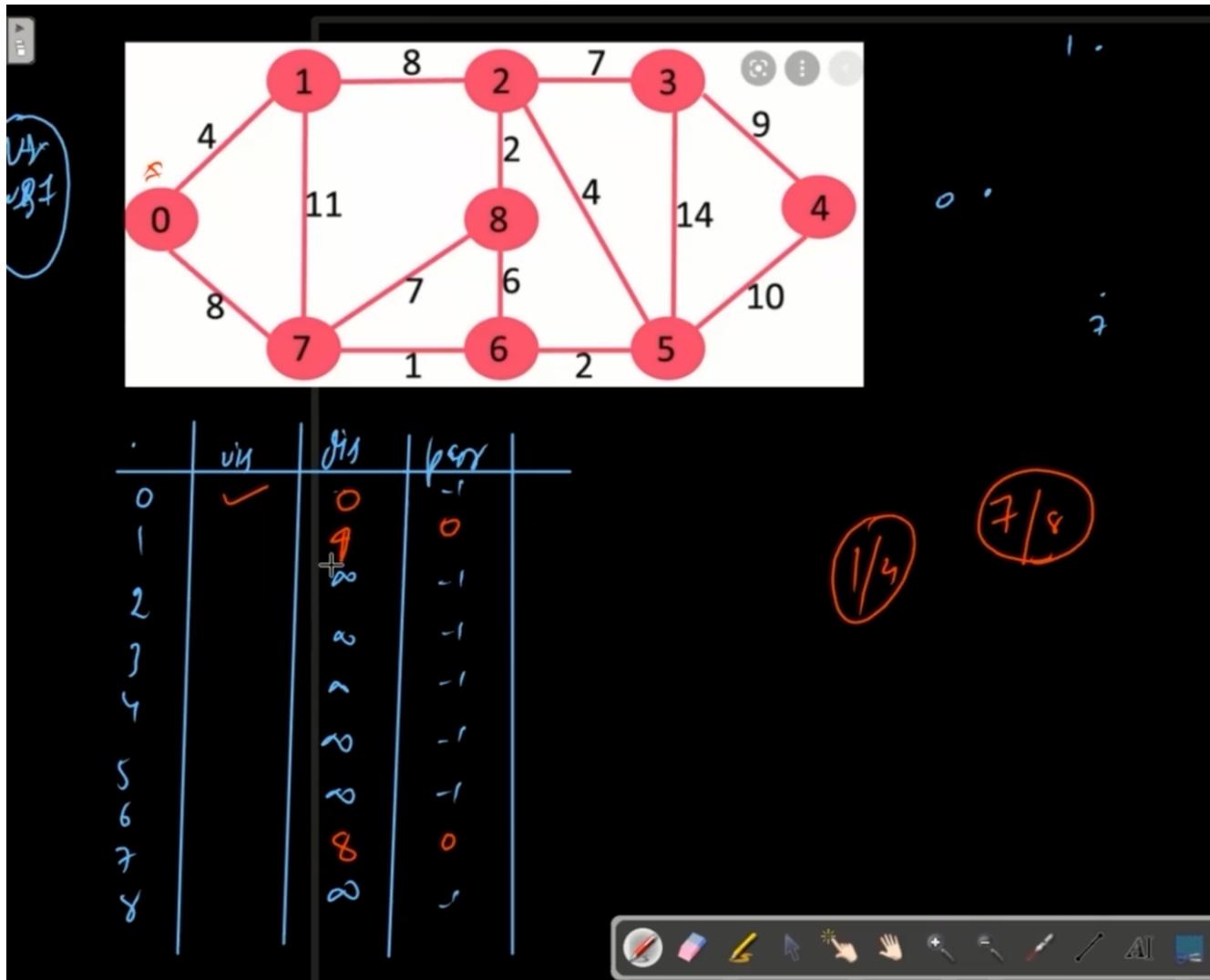
// # mai help karta hai.

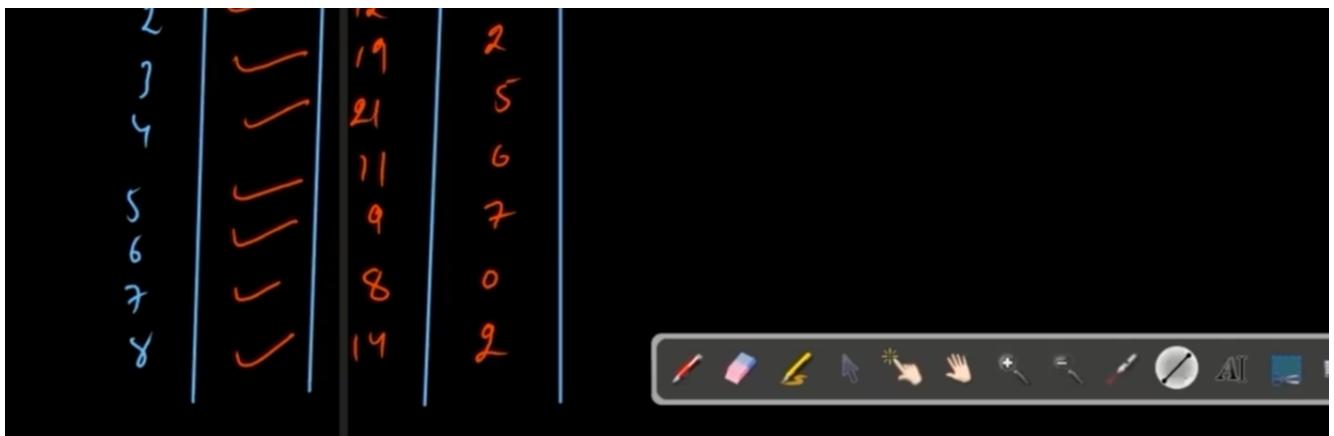
// Agar mujhe kisi vertex tak ka path chahiye, to mai usse puchunga ki tera
// parent kaun hai. Tab wo jo answer dega mai usse dubara question karrunga ki
// tera parent kaun hai. Aisa mai tab tak karta rahunga jab tak mai source ke
// parent tak nhi pahunch jata. joki hai -1. Aise mujhe pura path mil 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  
 // data hai. Per worst case mai dono ki same hoti hai.



Finding path using the parent array.

```

public static void dijkstra_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.vtx, 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, dis[v] });
            }
        }
    }

    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

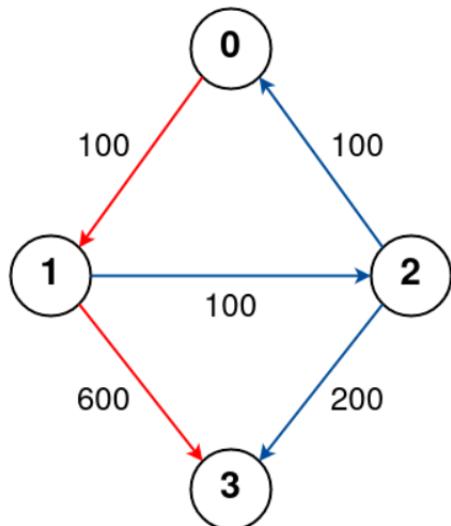
### 787. Cheapest Flights Within K Stops

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] = [fromi, toi, pricei]` indicates that there is a flight from city `fromi` to city `toi` with cost `pricei`.

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

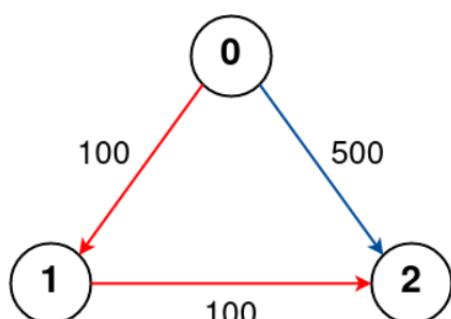
**Output:** 700

**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

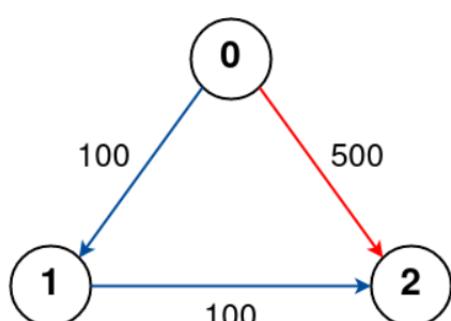
**Output:** 200

**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:**

- $1 \leq n \leq 100$
- $0 \leq \text{flights.length} \leq (n * (n - 1) / 2)$
- $\text{flights}[i].length == 3$
- $0 \leq \text{from}_i, \text{to}_i < n$
- $\text{from}_i \neq \text{to}_i$
- $1 \leq \text{price}_i \leq 10^4$
- There will not be any multiple flights between two cities.
- $0 \leq \text{src}, \text{dst}, k < n$
- $\text{src} \neq \text{dst}$

```
// 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 hogा 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 increased
// ? 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 =====>

// Dijkstra hume agar -ve edges hote hain to shi answer nikal ke nhi deta hai.
// Aur 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.

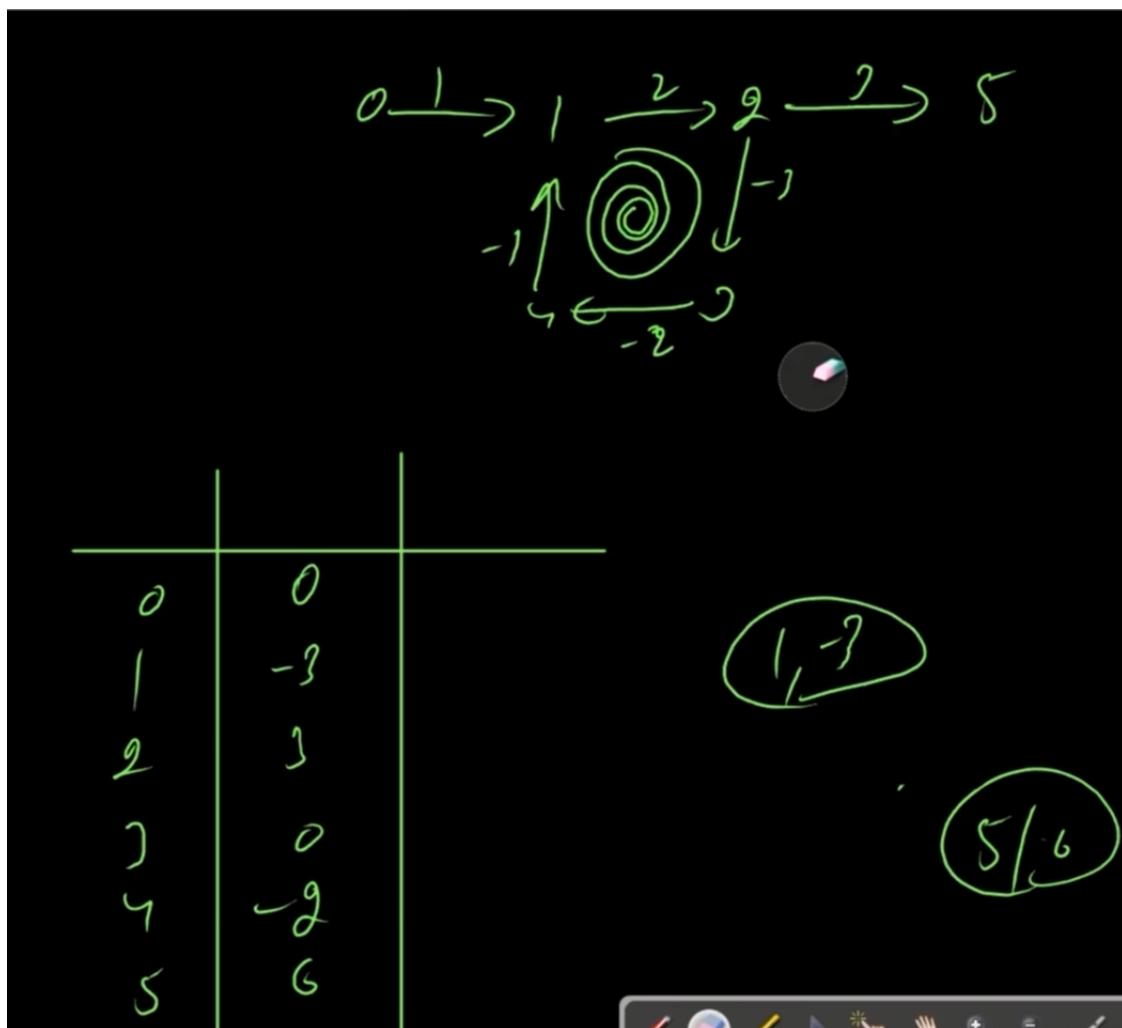
// # Dijkstra may aur may not correct answer dede in -ve edges.
// This may be true when the cycle overall weight will be +ve. Sirf correct
// answer aane ki possibility.

// 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 humesa ek spanning 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 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.

```

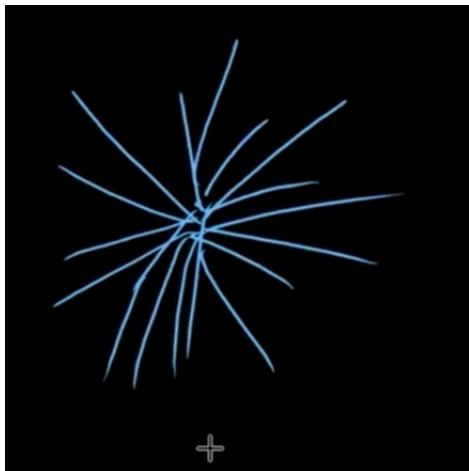
// # How to dry run
// Jitni vertex hain, utni kar sari edges ko traverse karna hai
// Ab manle AB edge aayi with wt = -1. To hum prev mai check karenge ki A kis
// weight ke sath aaya hai. Aur usme AB edge ki wt add kardenge to B mai us
// value ke sath pahunchenge. To Agar kam weight ke sath B pahunchenge to B ko
// current wale array mai update kar denge.

// # If current array does not update
// Agar Kahan 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.

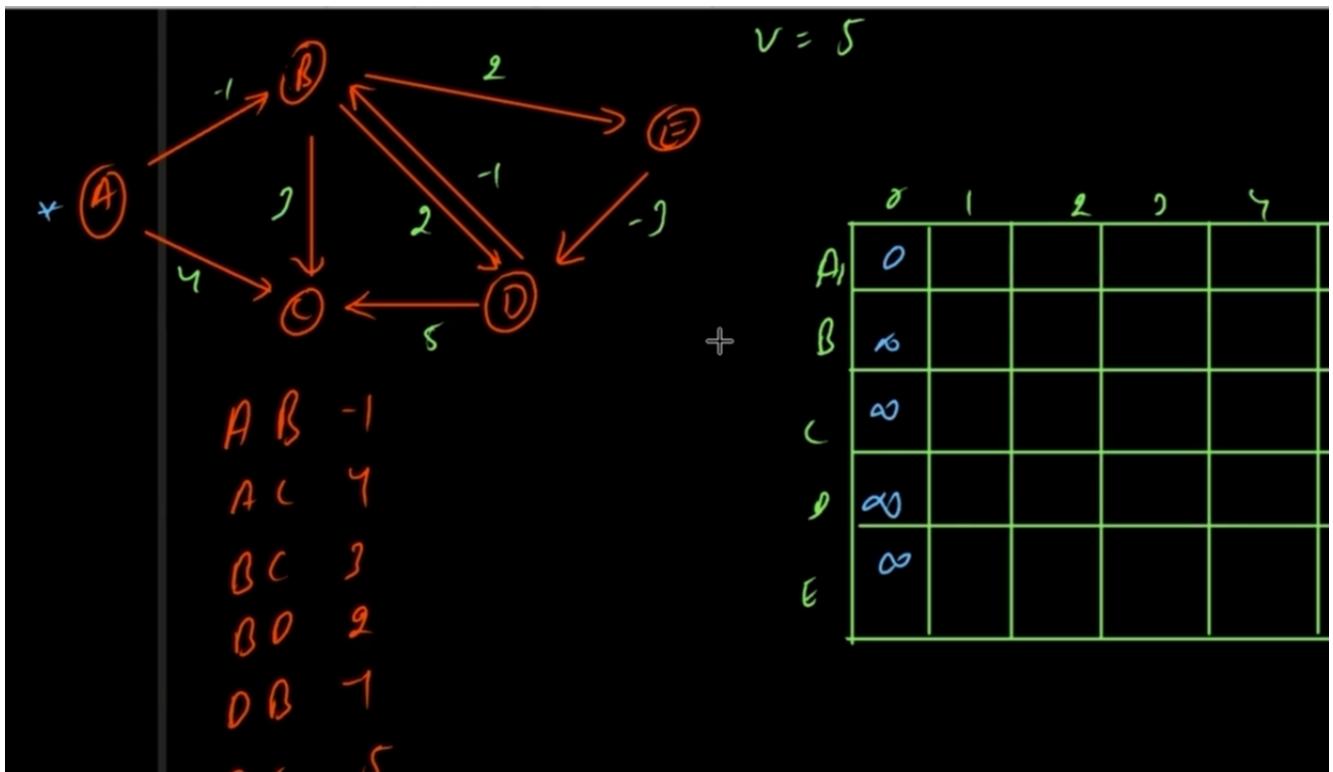
// # For Negative Cycle
// Agar last edge jo hum extra traverse kar rahe hain. (extra isiliye kyunki
// hume sirf V - 1 edges he chahiye hote hain), agar usme current wala array ek
// ^ bar bhi update hota hai, to iska matlab ye hai ki negativecycle Exist karti
// hai. To bas tab hume sahi answer nhi 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



	$D$	$C$	$B$	$A$	
	$G$	$D$	$-3$		
	$D$	$E$	$2$		

	$\sigma$	$1$	$2$	$3$	$4$	$5$
$A_1$	$0$	$0$				
$B$	$\infty$	$-1$				
$C$	$\infty$	<del><math>4</math></del>				
$D$	$\infty$	$\infty$				
$E$	$\infty$	$\infty$				
	$p$	$c$				

	$\sigma$	$1$	$2$	$3$	$4$	$5$
$A_1$	$0$	$0$	$0$			
$B$	$\infty$	$-1$	$1$			
$C$	$\infty$	<del><math>4</math></del>	$2$			
$D$	$\infty$	$\infty$	$1$			
$E$	$\infty$	$\infty$		$1$		
	$p$	$c$				

	$\sigma$	$1$	$2$	$3$	$4$	$5$
$A_1$	$0$	$0$	$0$	$0$		
$B$	$\infty$	$-1$	$1$	$-1$		
$C$	$\infty$	<del><math>4</math></del>	$2$	$2$		
$D$	$\infty$	$\infty$	$1$	$-2$		
$E$	$\infty$	$\infty$	$1$	$1$		
	$p$	$c$				

	$\sigma$	1	2	3	4	5
A	0	0	0	0	0	
B	$\infty$	-1	1	1	-2	
C	$\infty$	4	2	2	2	
D	$\infty$	0	1	-2	-2	
E	$\infty$	$\infty$	1	1	1	
		p	c			



	$\sigma$	1	2	3	4	5
A	0	0	0	0	0	0
B	$\infty$	-1	1	1	-2	-2
C	$\infty$	4	2	2	2	<del><math>\infty</math></del>
D	$\infty$	0	1	-2	-2	-2
E	$\infty$	$\infty$	1	1	1	1
		p	c			

```

// # 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];
    }
}

```

```

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 update tabhi hogा 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 hogा 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.

    }
}

```

## 52. Cheapest Flights Within K Stops

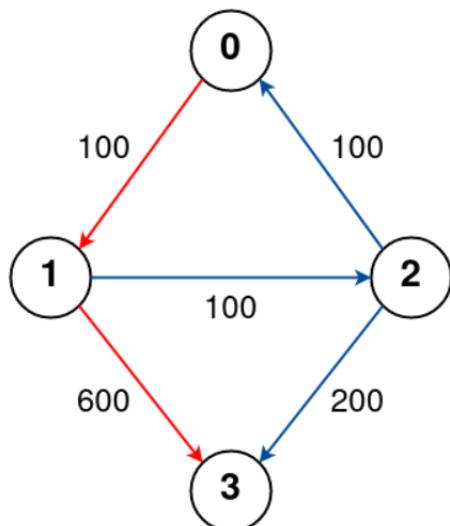
### 787. Cheapest Flights Within K Stops

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] = [fromi, toi, pricei]` indicates that there is a flight from city `fromi` to city `toi` with cost `pricei`.

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

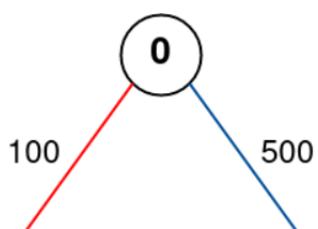
**Output:** 700

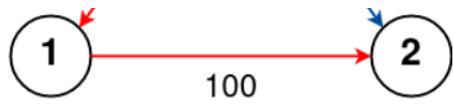
**Explanation:**

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  $100 + 600 = 700$ .

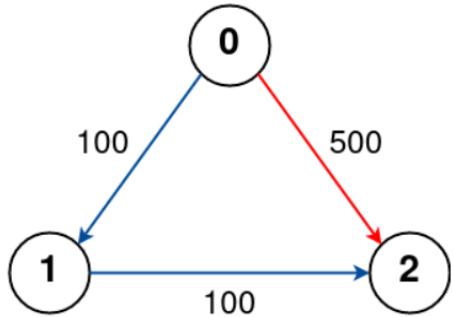
#### Example 2:





**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:

- $1 \leq n \leq 100$
- $0 \leq \text{flights.length} \leq (n * (n - 1)) / 2$
- $\text{flights}[i].length == 3$
- $0 \leq \text{from}_i, \text{to}_i < n$
- $\text{from}_i \neq \text{to}_i$
- $1 \leq \text{price}_i \leq 10^4$
- There will not be any multiple flights between two cities.
- $0 \leq \text{src}, \text{dst}, k < n$
- $\text{src} \neq \text{dst}$

```
// 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. G
- 54. Fa
- 55. Ags
- 56. Ag
- 57. Agd
- 58. Asgd
- 59. Asgd
- 60. Asdg
- 61. Asgdadsg
- 62. Sfd
- 63. Dsf
- 64. Hdhsd
- 65. Ssdfh
- 66. Hfsd
- 67. Shdsdfh
- 68. Shdfhsds
- 69. Hdsfhsd