Graphs 2

Keep the graph journey alive! In this interconnected world, we're tracing paths towards problem-solving treasures.

~~~~~~~~~~

Agenda :

~~~~~~~~~~

1. Rotten Oranges (Multi Source BFS) ← Application of BFS
2. DFS(depth first search) → traversal Approach on Graph (Recursive)
3. Connected Components ← technique in graph very helpful
4. No. of Islands
   - ↳ Application of Connected component

Rotten Oranges :

Given mat[N][M], where any cell can have one of the value :
0 -> Empty Cell
1 -> Fresh Orange
2 -> Rotten Orange
Every minute any fresh orange adjacent (top, right, bottom, left) to rotten orange becomes rotten. Find min time when all oranges become rotten.
If not possible to rott every orange, return -1.

Container:

	0	1	2	3	4
0	1 T=3	1 T=2	0	1 T=1	0
1	0	1 T=1	1 T=1	2 T=0	1 T=1
2	1 T=1	2 T=0	1 T=1	0	1 T=2
3	0	0	1 T=2	0	0

ans: 3 min

	0	1	2	3
0	0	1 T=2	0	1 T=1
1	0	1 T=1	1 T=1	2 T=0
2	1 T=1	2 T=0	1 T=1	0

T=0 → (1,3), (2,1)

T=1 → (0,3), (1,1), (1,2), (2,0), (2,2)

T=2 → (0,1)

Min time to rott all oranges = 2 min

Container:

	0	1	2	3	4
0	1	1	0	1	0
1	0	1	1	2	1
2	1	2	1	0	1
3	0	0	0	1	0

one orange is left

t=0 → (1,3), (2,1)

t=1 → (0,3), (1,2), (1,4) (1,1), (2,0), (2,2)

t=2 → (0,1), (2,4)

t=3 → (0,0)

→ if all oranges are not rotted
→ -1 Ans

	0	1	2	3
0	0	2	0	2
1	0	2	2	2
2	2	2	2	0

```
class pair {
    int i;   ] coordinate
    int j;   ]
    int t;   ] time
}
```

top
left ← (i, j) → Right
down

(1,3,0) | (2,1,0) | (0,3,1) | (1,2,1) | (1,1,1) | (2,0,1) | (2,2,1) | (0,1,2)

1,3 → 0
2,1 → 0
0,3 → 1
1,2 → 1
1,1 → 1
2,0 → 1

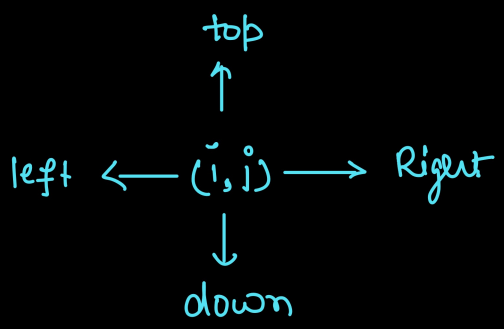
2,2 → 1
0,1 → 2

maximum time = 2 minute
rotted orge == total orge.
→ ans = 2 min

BFS
① Remove
② Print / work
③ Add unvisited neighbour

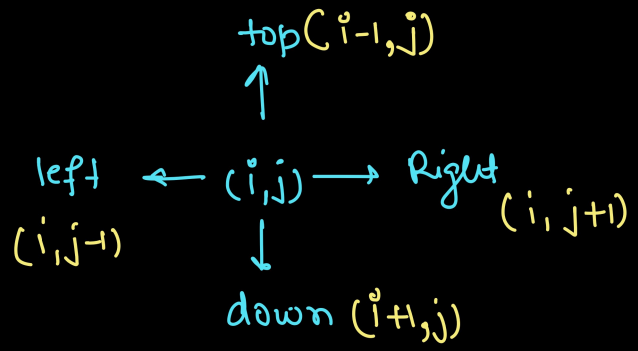
* Max time ← ②
* count orge which become rotted, Add from ③

Adding unvisited nb
→ mark
→ Add



Brute force Approach:

```
// top check
   ======
// left check
   ======
// down check
   ======
// Right check
   ======
```



	top	left	down	Right
xdir →	-1	0	1	0
ydir →	0	-1	0	1

d

i=2, j=3 (Assumption)

```
for(int d=0; d<4; d++){
    int r= i + xdir[d];
    int c= j + ydir[d];
    SOPln(r+","+c);
}
```

3

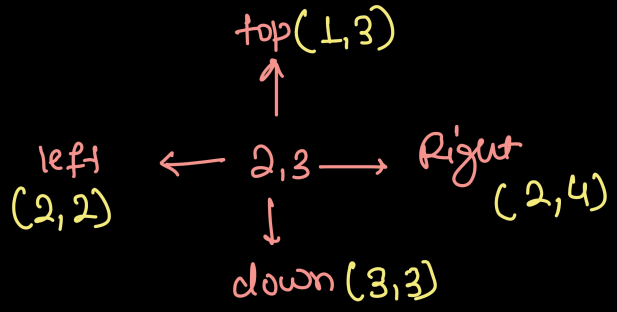
i=2, j=3

T d=0, r=2+(-1)=1, c=3+0=3

L d=1, r=2+0=2, c=3+(-1)=2

D d=2, r=2+1=3, c=3+0=3

R d=3, r=2+0=2, c=3+1=4



```java
public static class Pair {
    int i;
    int j;
    int t;
    public Pair(int i, int j, int t) {
        this.i = i;
        this.j = j;
        this.t = t;
    }
}

public static int[] xdir = {-1, 0, 1, 0};
public static int[] ydir = {0, -1, 0, 1};

public static int rottenOranges(int[][] arr) {
    /*
        Make a queue and add all oranges which are already rotted
        if they are already rotted, time for them is T=0
        make sure while adding rotted oranges in queue
        take the count of total oranges as well
    */
    int n = arr.length;
    int m = arr[0].length;

    Queue<Pair> qu = new ArrayDeque<>();

    int count = 0;
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < m; j++) {
            if(arr[i][j] == 2) {
                // (i,j) location is already rotted
                qu.add(new Pair(i,j,0));
            }

            // count the total orange as well
            // arr[i][j] == 1 || arr[i][j] == 2 => arr[i][j] != 0
            if(arr[i][j] != 0) {
                // rotted + fresh
                count++;
            }
        }
    }

    // Apply BFS Algorithm on que
    // multiple starting point available for BFS
    int time = 0;
    while(qu.size() > 0) {
        // remove
        Pair rem = qu.remove();
        // one orange rotted, decrease the count from total orange
        count--;
        // work  -> maximise the time
        time = rem.t;
        // add unvisited neighbour
        // iteration on 4 direction -> top, left, down, right
        for(int d = 0; d < 4; d++)  {
            int r = rem.i + xdir[d];
            int c = rem.j + ydir[d];

            if(r >= 0 && r < n && c >= 0 && c < m && arr[r][c] == 1) {
                // marking
                arr[r][c] = 2;
                // adding in que
                qu.add(new Pair(r,c, rem.t + 1));
            }
        }
    }
    if(count == 0) {
        return time;
    } else {
        return -1;
    }
}
```

Row and colm Count

→ Apply BFS Algo

→ to calculate total Orge.

→ If Rotted add with t=0

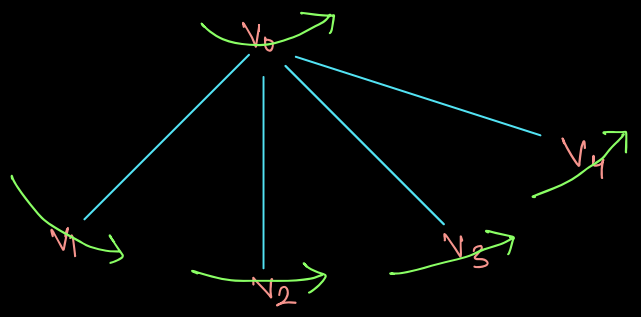
→ If it is orge add it in count

helping in making sure that location is valid

helping in making sure that added orge is fresh.

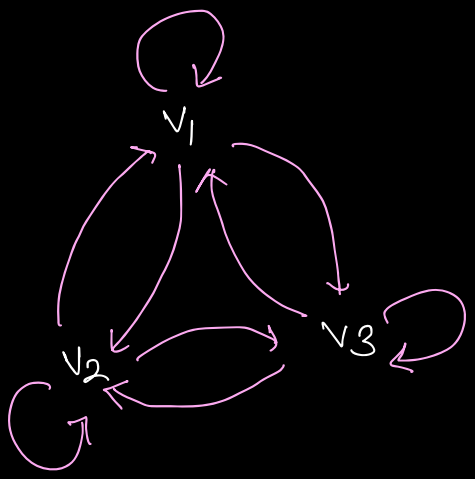
time complexity:

BFS Algorithm:



T.C: $O(V+E)$ → way of Representation for time complexity of Graph.

Edge count in any graph:



Edge count = If vertex = 3
= 9 (worst case

In generic world for V vertex
Edge count = V^2

time Complexity = $O(V+E)$
= $O(V+V^2)$
= $O(V^2)$ → worst case
not necessary that Edge is always V^2.

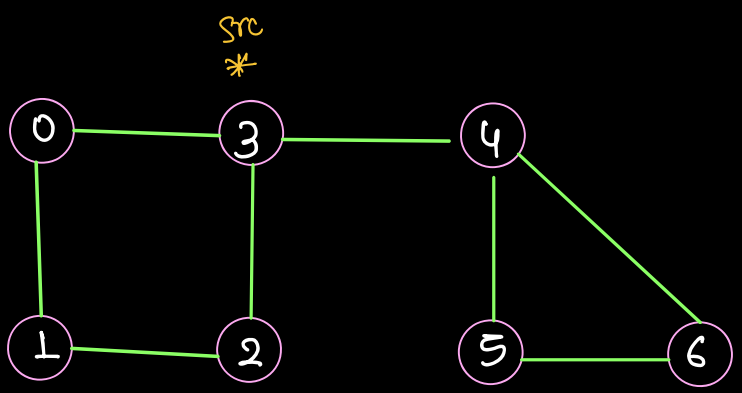
thatsway → T.C: $O(V+E)$

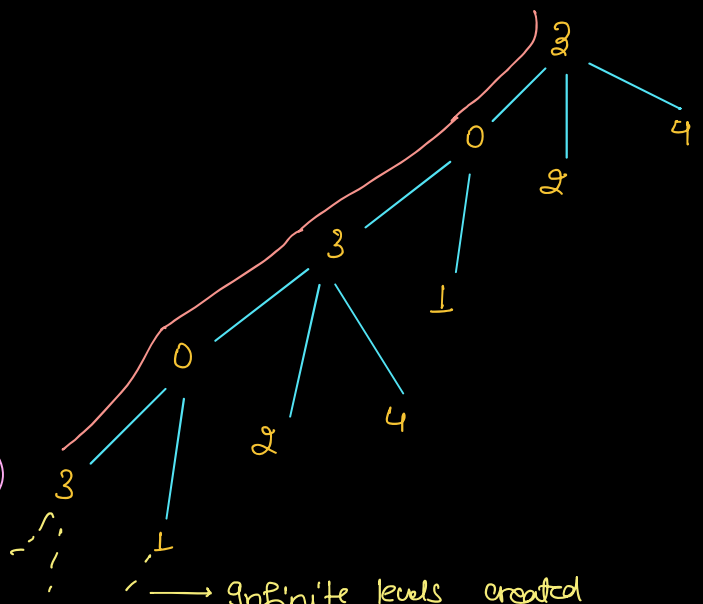
10:26 - 10:36 PM Break time

~~~~~~~~~~~~~~~~~~~~

Depth First Search :

~~~~~~~~~~~~~~~~~~~~

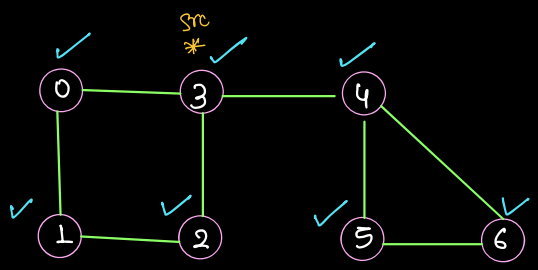


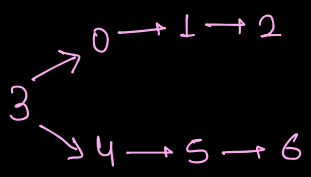


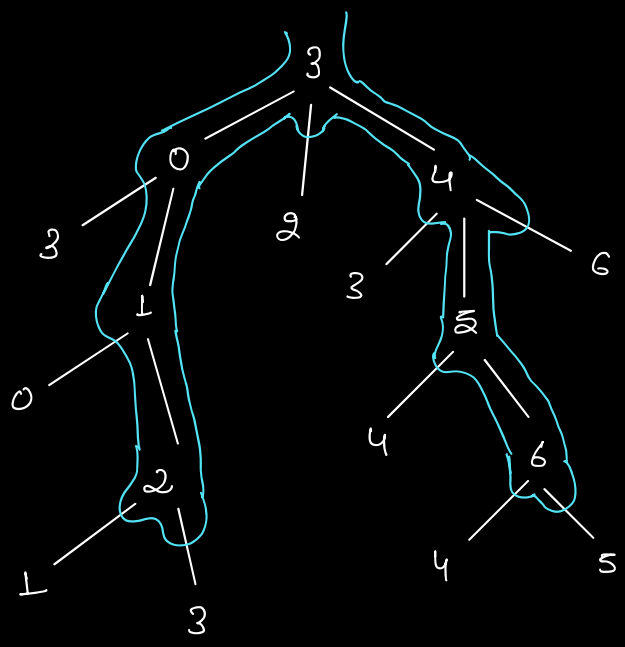
→ Infinite levels created

→ Don't visit already visited vertex

manage visited array

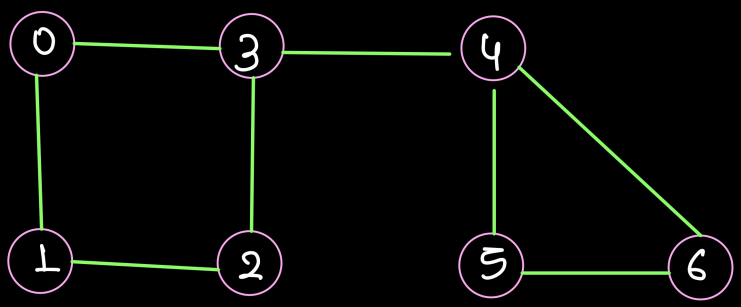






order of DFS : 3 → 0 → 1 → 2 → 4 → 5 → 6 (Depth first)

DFS Algorithm



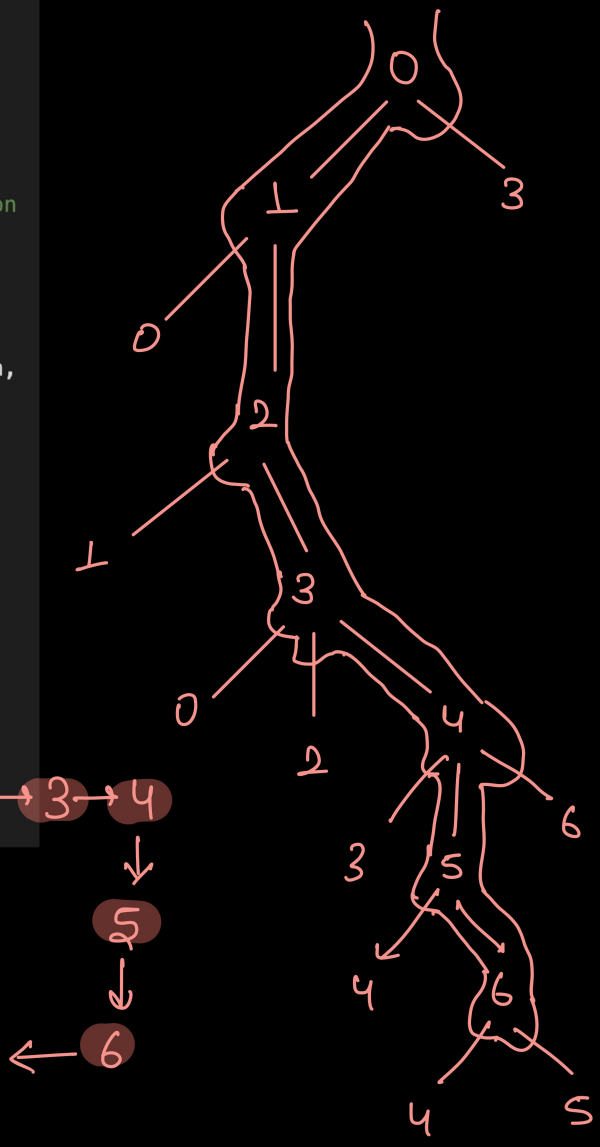
n=7

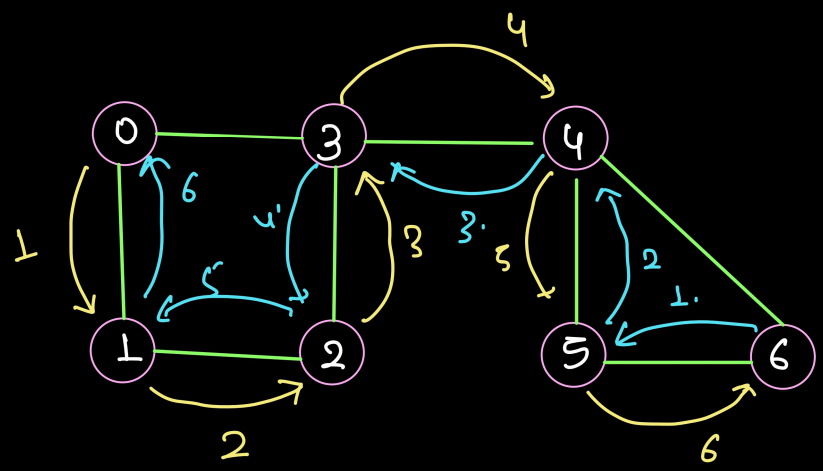
0	1	2	3	4	5	6
T	T	T	T	T	T	T

```java
public static void DFS(ArrayList<ArrayList<Integer>> graph) {
    // total vertex
    int n = graph.size();
    // prepare visited array
    boolean[] vis = new boolean[n];
    // For Now, assume source point
    int src = 0;
    // mark yourself and then make call to helper of DFS function
    vis[src] = true;
    dfsHelper(graph, src, vis);
}

public static void dfsHelper(ArrayList<ArrayList<Integer>> graph,
    int src, boolean[] vis) {
    // 1. Print
    System.out.print(src + " ");
    // 2. move toward unvisited neighbour
    for(int nbr : graph.get(src)) {
        // unvisited neighbour
        if(vis[nbr] == false) {
            // mark that neighbour
            vis[nbr] = true;
            // move toward that neighbour
            dfsHelper(graph, nbr, vis);
        }
    }
}
```

o/p: 0 → 1 → 2 → 3 → 4 → 5 → 6

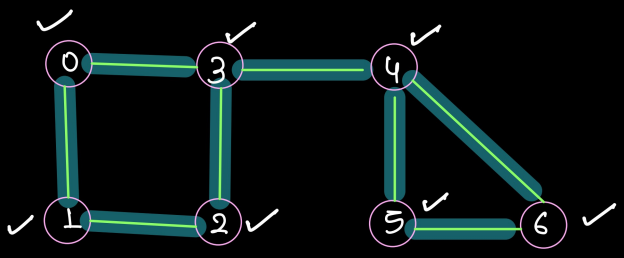




Connected Components :

Given an undirected graph, find total number of connected components.

Example1:

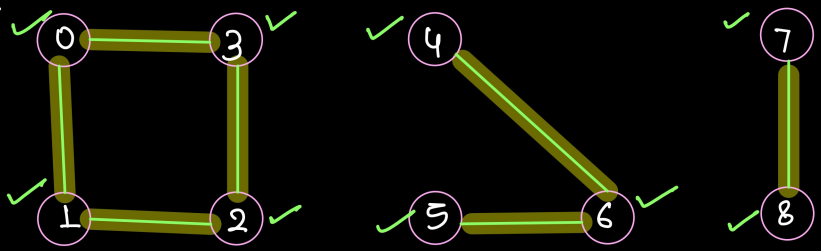


V= 0 1 2 3 4 5 6

graph is totally connected and making one component

no. of connected comp = 1

Example2:

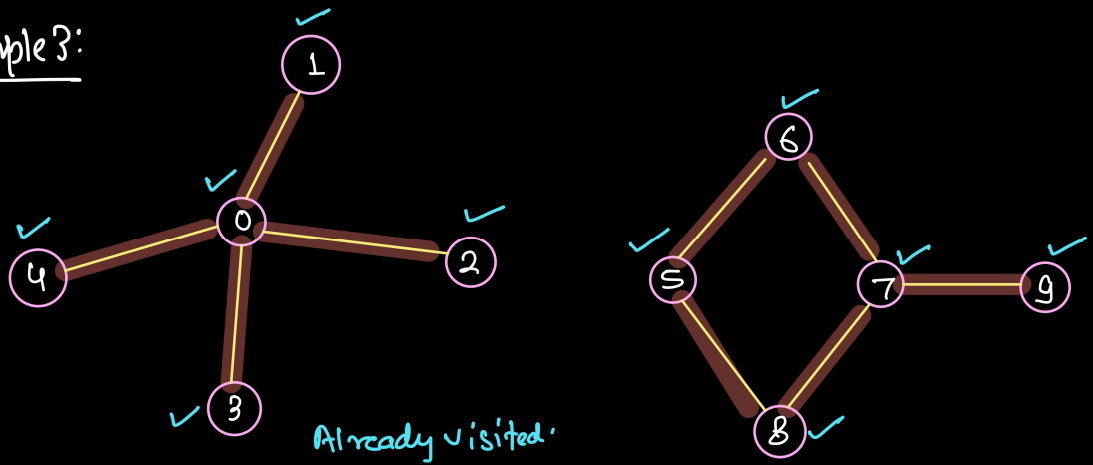


V= 0 1 2 3 4 5 6 7 8

no. of component = 3

dfs(0)
dfs(4)
dfs(7)
} 3 source point allowed.

Example3:



Already visited.

V= 0 1 2 3 4 5 6 7 8 9

dfs(0) → ✓
dfs(5) → ✓
} no. of connected component = 2

```java
public static int solveConnectedComponent(ArrayList<ArrayList<Integer>> graph) {
    int n = graph.size();
    boolean[] vis = new boolean[n];
    // try all vertex from graph
    int comps = 0;
    for(int v = 0 ; v < n; v++) {
        if(vis[v] == false) {
            vis[v] = true;
            connectedComponent(graph, v, vis);
            comps++;
        }
    }
    return comps;
}
public static void connectedComponent(ArrayList<ArrayList<Integer>> graph,
    int src, boolean[] vis) {
    for(int nbr : graph.get(src)) {
        // unvisited neighbour
        if(vis[nbr] == false) {
            vis[nbr] = true;
            connectedComponent(graph, nbr, vis);
        }
    }
}
```

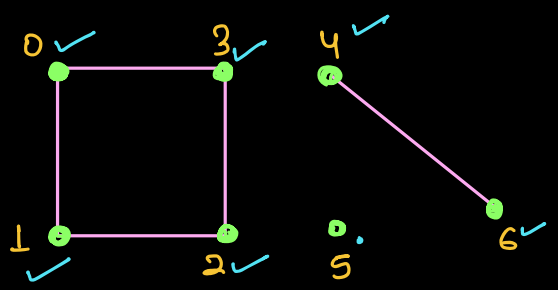
```java
public static int solveConnectedComponent(ArrayList<ArrayList<Integer>> graph) {
    int n = graph.size();
    boolean[] vis = new boolean[n];
    // try all vertex from graph
    int comps = 0;
    for(int v = 0 ; v < n; v++) {
        if(vis[v] == false) {
            vis[v] = true;
            connectedComponent(graph, v, vis);
            comps++;
        }
    }
    return comps;
}
public static void connectedComponent(ArrayList<ArrayList<Integer>> graph,
    int src, boolean[] vis) {
    for(int nbr : graph.get(src)) {
        // unvisited neighbour
        if(vis[nbr] == false) {
            vis[nbr] = true;
            connectedComponent(graph, nbr, vis);
        }
    }
}
```

n=7

T	T	T	T	T	T	T
0	1	2	3	4	5	6

return comps; → 3

Same as DFS



comps = 0 → 1 → 2 → 3

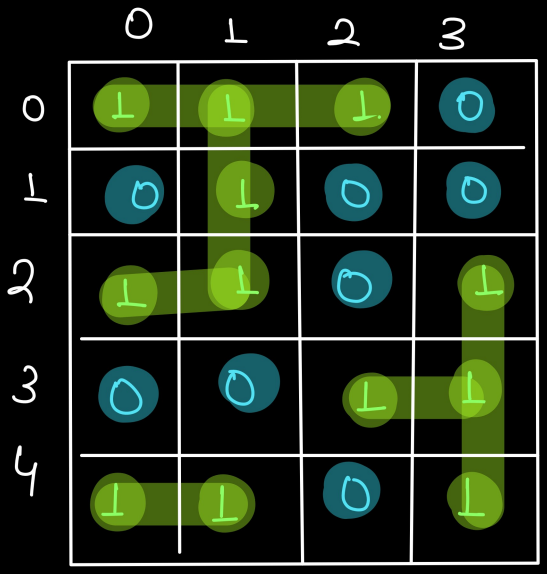
- v = 0 → DFS → 0, 1, 2, 3
- x = 1 → Already marked
- x = 2 → " "
- x = 3 → " "
- v = 4 → DFS → 4, 6
- v = 5 → DFS → 5
- x = 6 → Already marked

~~~~~~~~~~~~~~~~

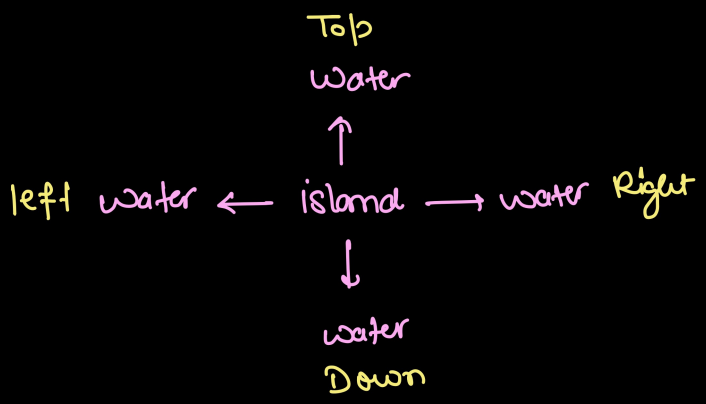
No. of Islands :

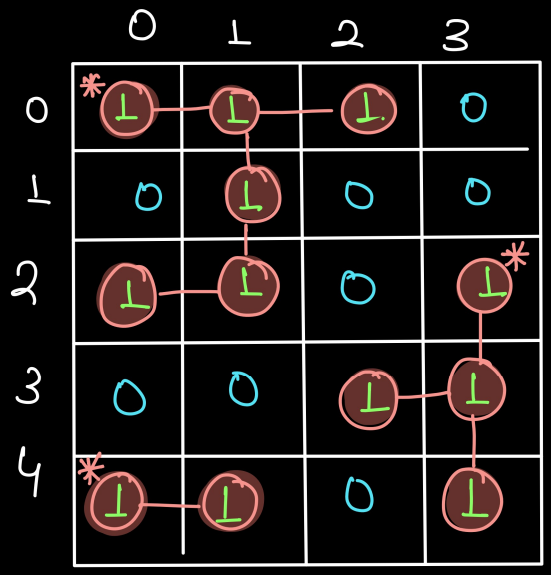
~~~~~~~~~~~~~~~~

Given mat[N][M], where 0 represent water cell and 1 represent land cell.
Find total number of Islands.
Note : An Islands can be formed by connecting adjcent land cells (T,L,D,R).



No. of island = 3





No. of connected Component

No. of calls → 3 calls.

No. of island → connected component

DFS → we will write

BFS → TODO

```java
public static void IslandComps(int[][] arr, int i, int j) {
    // iterate on all possible neighbours -> T,L,D,R
    for(int d = 0; d < 4; d++) {
        int r = i + xdir[d];
        int c = j + ydir[d];

        if(r >= 0 && r < arr.length &&
            c >= 0 && c < arr[0].length &&
            arr[r][c] == 1) {
            // mark
            arr[r][c] = -1;
            // make call toward that land
            IslandComps(arr, r, c);
        }
    }
}
```

```java
// number of island
public static int numberOfIsland(int[][] arr) {
    int count = 0;

    for(int i = 0; i < arr.length; i++) {
        for(int j = 0; j < arr[0].length; j++) {
            if(arr[i][j] == 1) {
                // mark
                arr[i][j] = -1;
                // call
                IslandComps(arr, i, j);
                // increment in count
                count++;
            }
        }
    }
    return count;
}
```

29th January 2024
next Session

25
26
27
28
} try to cover backlog.

[Data Dump.]

100% → 4 days.