Agenda

1) Rotten oranges (multisource BFS)
2) DFS (Depth first search)
3) Connected component
4) No. of Islands

Q.1 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.

mat =

	0	1	2	3	4
0	~~1~~ ^{3}	~~1~~ ^{2}	0	~~1~~ ^{1}	0
1	0	~~1~~ ^{1}	~~1~~ ^{1}	2	~~1~~ ^{1}
2	~~1~~ ^{1}	2	~~1~~ ^{1}	0	~~1~~ ^{2}
3	0	0	~~1~~ ^{2}	0	0

ans = 3

mat =

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

ans = 2

mat =

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

ans = -1

→ we can apply BFS (multisource)

mat =

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

```
Class Pair {
        int r;
        int c;
        int t;
}
```

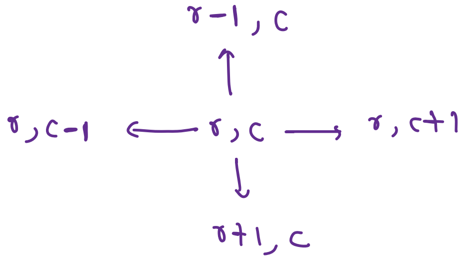
q: | ~~1,3,0~~ | ~~2,1,0~~ | ~~0,3,1~~ | ~~1,2,1~~ | ~~1,1,1~~ | ~~2,0,1~~ | ~~2,2,1~~ | ~~0,1,2~~ |

→ remove

→ add unvisited nbr

```
int  rottenoranges ( int [] [] mat) {
    int  N = mat.length;
    int  M = mat [0].length;
    Queue < Pair > q = new  ArrayDeque<>();

    // travel mat and add original rotted cells
    for (int i=0; i<N; i++) {
        for (int j=0; j<M; j++) {
            if ( mat [i] [j] == 2) {
                Pair np = new Pair (i, j, 0);
                q.add (np);
            }
        }
    }

    // apply BFS and find min time
    int ans = 0;
    while (q.size() > 0) {
        Pair rmv = q.remove();
        int r = rmv.r;
        int c = rmv.c;
        int t = rmv.t;

        ans = t;
        // add unvisited nbr
        // top nbr
        if (r-1 >= 0  && mat [r-1] [c] == 1) {
            mat [r-1] [c] = 2;
            q.add (new Pair (r-1, c, t+1));
        }
```

```
class Pair {
    int r;
    int c;
    int t;
    //constructor
}
```

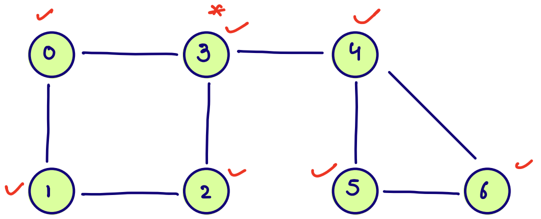


```
            // left nbr
            if (c-1 >= 0   &&  mat[r][c-1] == 1) {
                mat[r][c-1] = 2;
                 q.add (new Pair(r, c-1, t+1));
            }
            // bottom nbr
            if (r+1 < n   &&   mat[r+1][c] == 1) {
                mat[r+1][c] = 2;
                 q.add (new Pair(r+1, c, t+1));
            }

            // right nbr
            if ( c+1  < m   &&  mat[r][c+1] == 1) {
                mat[r][c+1] = 2;
                 q.add (new Pair(r, c+1, t+1));
            }
    }

    // travel and check any remaining fresh orange

     for (int i=0; i<N; i++) {
            for (int j=0; j<M; j++) {
                 if ( mat [i][j] == 1) {
                      return -1;
                 }
            }
     }
    return ans;
}
```

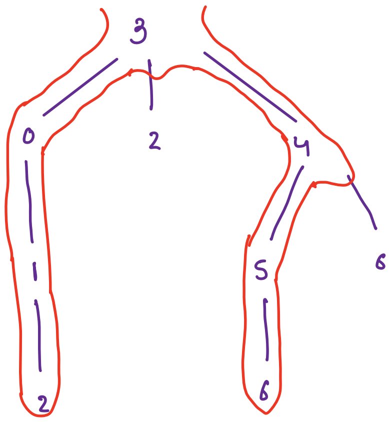
TC: O(N*M)

SC: O(1)

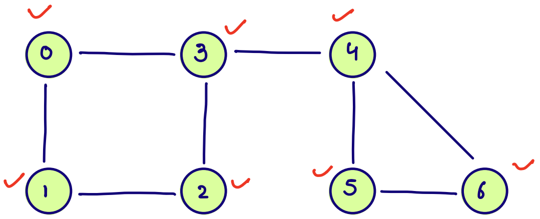
DFS (Depth First search)

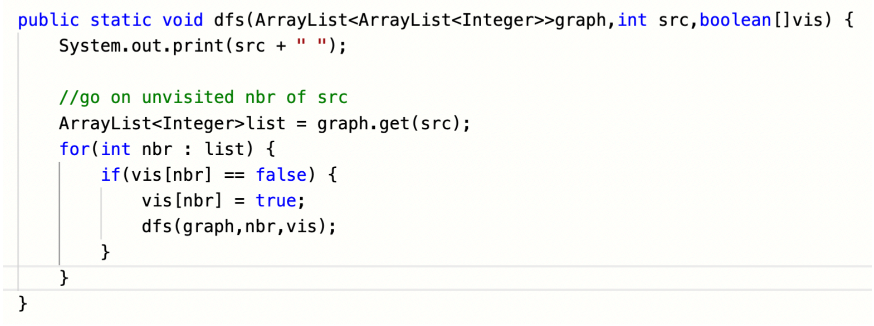


Src = 3

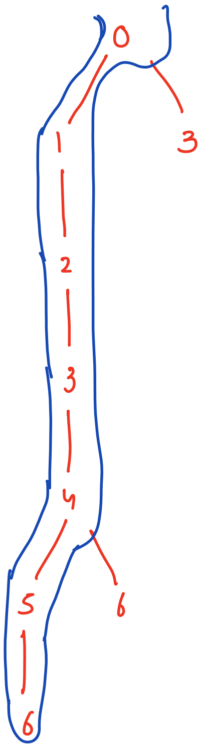


0 1 2 3 4 5 6



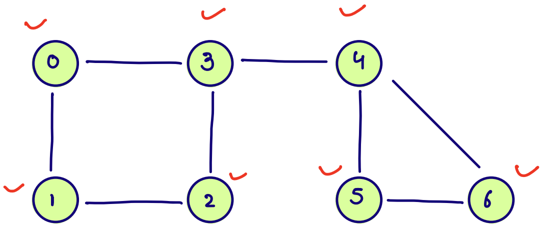


```java
public static void dfs(ArrayList<ArrayList<Integer>>graph,int src,boolean[]vis) {
    System.out.print(src + " ");

    //go on unvisited nbr of src
    ArrayList<Integer>list = graph.get(src);
    for(int nbr : list) {
        if(vis[nbr] == false) {
            vis[nbr] = true;
            dfs(graph,nbr,vis);
        }
    }
}
```



Q.2 Given a undirected graph, find total no. of connected components.

eg1.

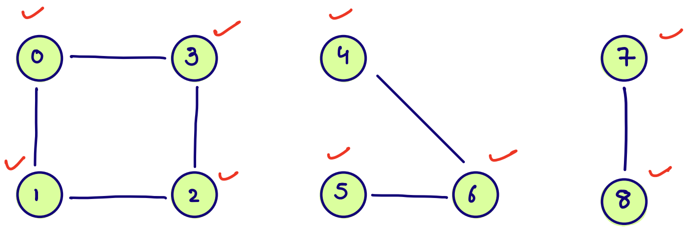


ans = 1

vis

once travelling from 0 : 0, 1, 2, 3, 4, 5, 6
is done

eg2.



ans = 3

vis

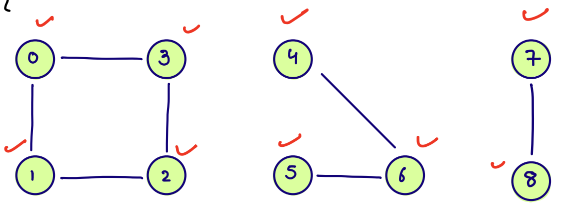
once toavelling from 0 : 0 1 2 3
is done

once toavelling from 4 : 4 5 6
is done

once toavelling from 7 : 7 8
is done

```
int  connectedComp ( AL < AL < Integer>> graph) {
     int  comps = 0;
     boolean[] vis = new  boolean [graph.size()];
     for (int  i=0; i< graph.size(); i++) {
             if (vis[i] == false) {
                 comp++;
                 //travel starting from i
                 vis[i] = true;
                 dfs(graph, i, vis);
             }
     }
     return  comp;
}


void  dfs ( AL < AL < Integer>>graph, int  src, boolean[]vis) {
      AL <Integer> list =  graph.get(src);
      for (int  nbr  :  list) {
             if (vis[nbr] ==false) {
                 vis[nbr] = true;
                 dfs( graph, nbr, vis);
             }
      }
}
```

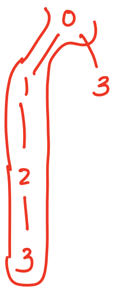
```
int  connected comp ( AL < AL < Integer>> graph) {

    int  comps = 0;

    boolean [ ] vis = new boolean [graph.size()];

    for (int i=0; i<graph.size(); i++) {
            if (vis[i] == false) {
                comp++;
                //travel starting from i
                vis[i] = true;
                dfs(graph, i, vis);
            }
    }

    return comp;
}

void dfs ( AL < AL < Integer>>graph, int src, boolean[]vis) {
    AL <Integer> list =  graph.get(src);

    for (int nbr : list) {
        if (vis[nbr]==false) {
            vis[nbr] = true;
            dfs( graph,nbr,vis);
        }
    }
}
```



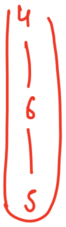
comp = ~~1~~ ~~2~~ 3

i	travel
0	✓
1	X
2	X
3	X
4	✓
5	X
6	X
7	✓
8	X

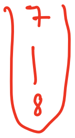
dfs(0)



dfs(4)



dfs(7)



Q.3 Given a mat[N][M] where 0 represents water cell and 1 represents land cell. Find total no. of islands.

Note: An island can be formed by connecting adjacent land cells.
↳ (T, L, B, R)

$$\text{mat} = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 0 & 0 & 1 & 1 \\ 4 & 1 & 1 & 0 & 1 \end{array}$$

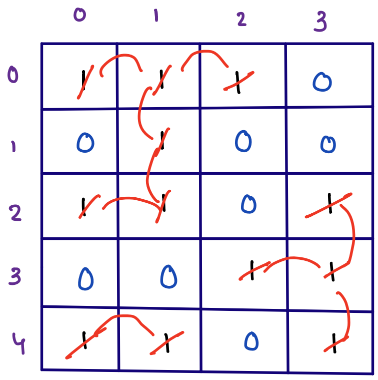
ans=3

$$\text{mat} = \begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 2 & 1 & 1 & 0 & 1 \\ 3 & 0 & 0 & 1 & 1 \\ 4 & 1 & 1 & 1 & 1 \end{array}$$

ans= 2

application of connected comps

(count of connected comps = no. of islands) mat =



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \cancel{1} & \cancel{1} & \cancel{1} & 0 \\ 1 & 0 & \cancel{1} & 0 & 0 \\ 2 & \cancel{1} & \cancel{1} & 0 & \cancel{1} \\ 3 & 0 & 0 & \cancel{1} & \cancel{1} \\ 4 & \cancel{1} & \cancel{1} & 0 & \cancel{1} \end{array}$$

comps = ~~0~~ ~~1~~ ~~2~~ 3

dfs (0,0)

dfs (2,3)

dfs (4,0)

```
int     islandCount ( int [ ] [ ] mat) {
        int  N = mat.length;
        int  M = mat[0].length;
         int  comps = 0;
         for (int  i=0;  i<N;  i++) {
                for( int j=0;  j<M;  j++) {
                       if  (mat[i][j] == 1) {
                          comps++;
                         //travel from  (i,j)
                          mat[i] [j] = -1;
                           dfs( mat, i, j);
                       }
                }
         }
         return  comps;
}
```

```
void  dfs ( int [ ] [ ] mat, int i, int j) {

    // top
    if( i-1 >= 0  &&  mat [i-1] [j] == 1) {

        mat [i-1] [j] =  -1;
        dfs ( mat, i-1, j);
    }

    // left
    if( j-1 >=0  &&  mat [i] [j-1] == 1) {

        mat [i] [j-1] =  -1;
        dfs ( mat, i , j-1);
    }

    // bottom
    if( i+1  <  mat.length  &&  mat [i+1] [j] == 1) {

        mat [i+1] [j] =  -1;
        dfs ( mat, i+1, j);
    }

    // right
    if( j+1  <  mat [0].length && mat [i] [j+1] == 1) {

        mat [i] [j+1] =  -1;
        dfs ( mat, i , j+1);
    }

}
```