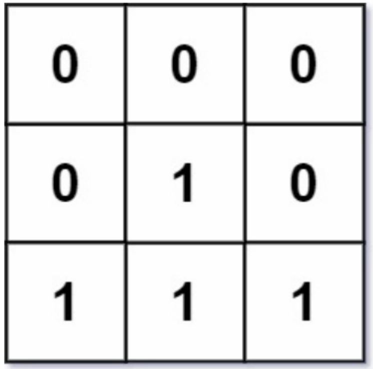
542. 01 Matrix

BFS

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

The distance between two adjacent cells is `1`.



0	0	0
0	1	0
1	1	1

0	0	0
0	1	0
1	2	1

(i) Start from zero

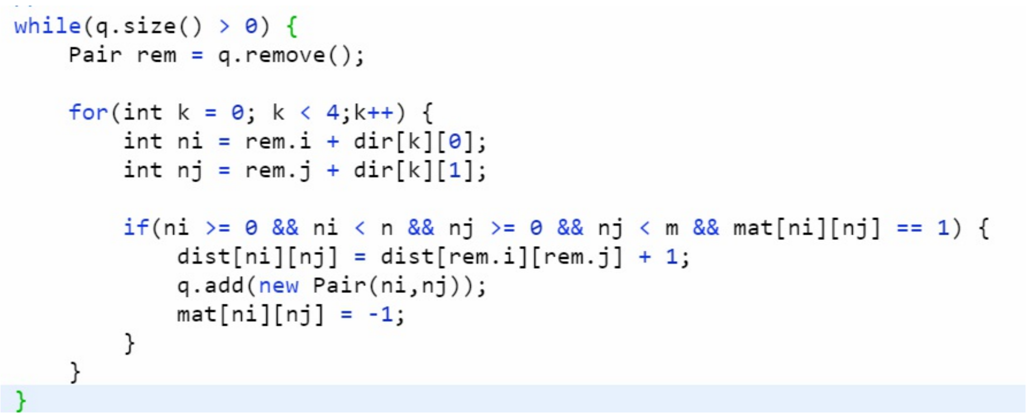
	0	1	2	3
0	-1	-1	0	-1
1	-1	-1	-1	-1
2	-1	-1	-1	0
3	-1	-1	-1	0

dist

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

```
Pair {
 int i;
  int j;
}
```

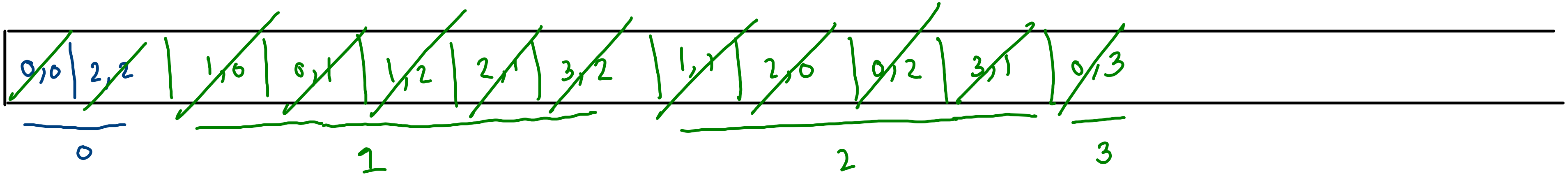
$$\underbrace{\cancel{0,2} \mid \cancel{2,3} \mid \cancel{3,3}}_{0} \mid \underbrace{\cancel{0,1} \mid \cancel{1,2} \mid \cancel{0,3} \mid \cancel{2,2} \mid \cancel{1,3} \mid \cancel{3,2}}_{1} \mid \underbrace{\cancel{0,0} \mid \cancel{1,1} \mid \cancel{2,1} \mid \cancel{3,1}}_{2} \mid \underbrace{\cancel{1,0} \mid \cancel{2,0} \mid \cancel{3,0}}_{3}$$



```
while(q.size() > 0) {
    Pair rem = q.remove();

    for(int k = 0; k < 4;k++) {
        int ni = rem.i + dir[k][0];
        int nj = rem.j + dir[k][1];

        if(ni >= 0 && ni < n && nj >= 0 && nj < m && mat[ni][nj] == 1) {
            dist[ni][nj] = dist[rem.i][rem.j] + 1;
            q.add(new Pair(ni,nj));
            mat[ni][nj] = -1;
        }
    }
}
```

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

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



1162. As Far from Land as Possible

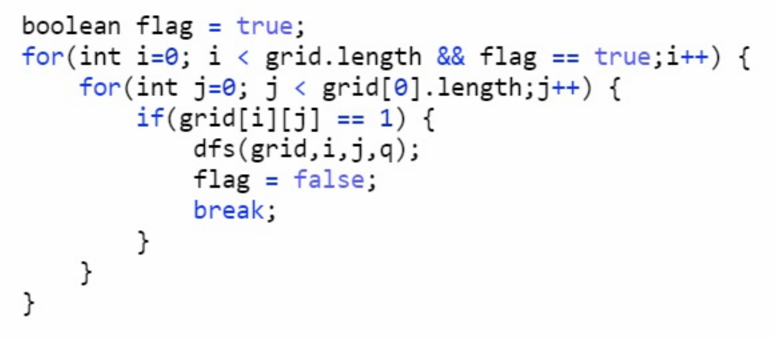
(i) Exactly same with 01 matrix

(ii) change : 01 matrix : find distance of each cell from nearest zero :

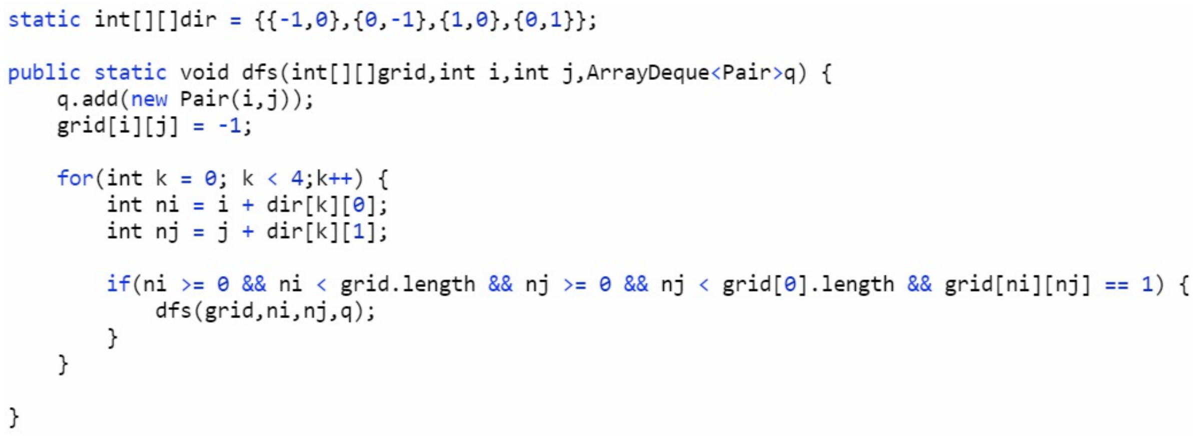
this ques : find distance of each water cell (0) from nearest land cell (1) and return the max-dist.

934. Shortest Bridge

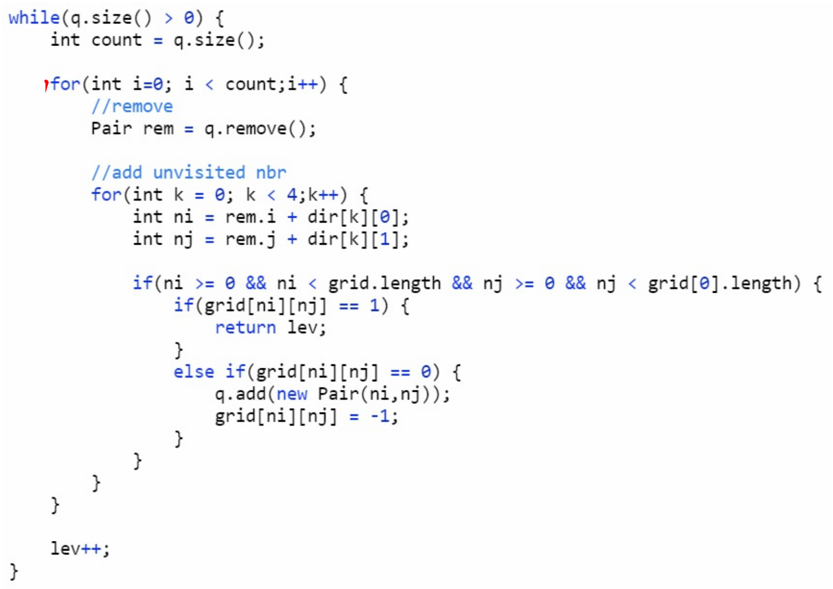
	0	1	2	3	4
0	0	-1	0	0	0
1	-1	-1	0	0	1
2	-1	0	0	1	1
3	0	0	0	0	1



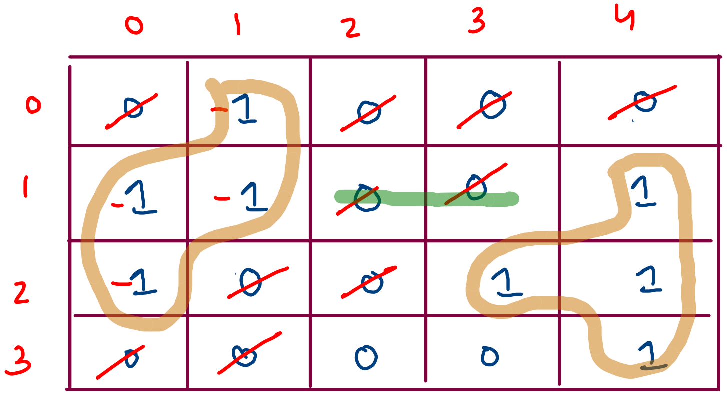
```
boolean flag = true;
for(int i=0; i < grid.length && flag == true;i++) {
    for(int j=0; j < grid[0].length;j++) {
        if(grid[i][j] == 1) {
            dfs(grid,i,j,q);
            flag = false;
            break;
        }
    }
}
```



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

public static void dfs(int[][]grid,int i,int j,ArrayDeque<Pair>q) {
    q.add(new Pair(i,j));
    grid[i][j] = -1;

    for(int k = 0; k < 4;k++) {
        int ni = i + dir[k][0];
        int nj = j + dir[k][1];

        if(ni >= 0 && ni < grid.length && nj >= 0 && nj < grid[0].length && grid[ni][nj] == 1) {
            dfs(grid,ni,nj,q);
        }
    }

}
```



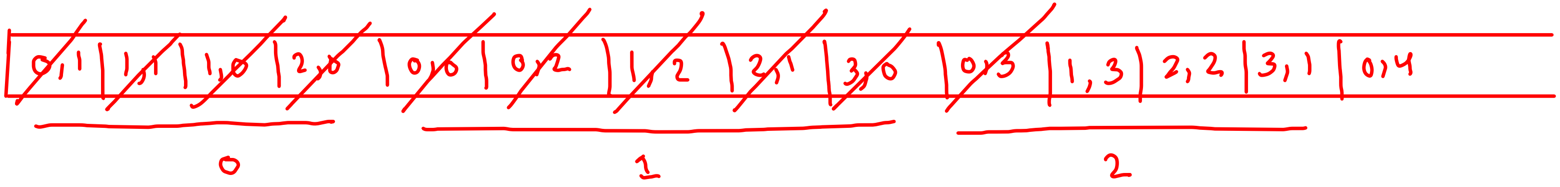
```
while(q.size() > 0) {
    int count = q.size();

    for(int i=0; i < count;i++) {
        //remove
        Pair rem = q.remove();

        //add unvisited nbr
        for(int k = 0; k < 4;k++) {
            int ni = rem.i + dir[k][0];
            int nj = rem.j + dir[k][1];

            if(ni >= 0 && ni < grid.length && nj >= 0 && nj < grid[0].length) {
                if(grid[ni][nj] == 1) {
                    return lev;
                }
                else if(grid[ni][nj] == 0) {
                    q.add(new Pair(ni,nj));
                    grid[ni][nj] = -1;
                }
            }
        }
    }

    lev++;
}
```



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

c = 4

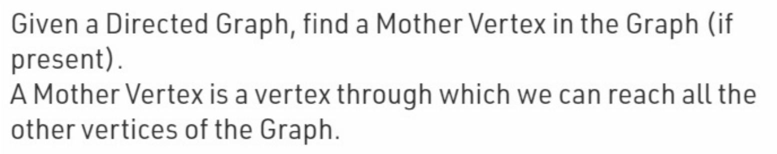
lev = ~~0~~
~~1~~ 2



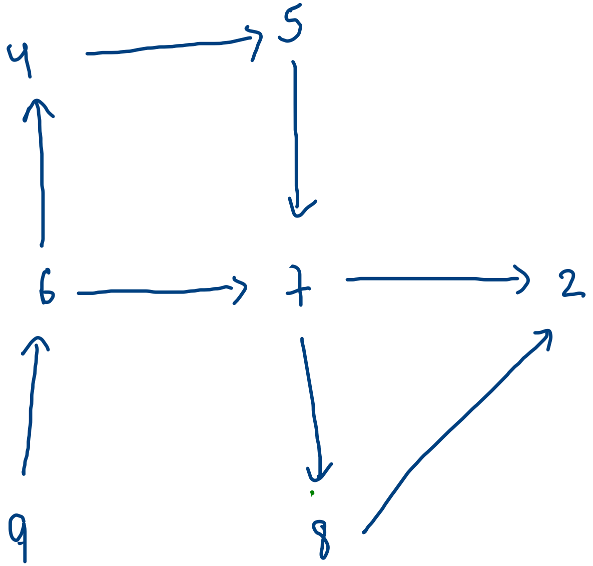
~~0,1~~	~~1,1~~	~~1,0~~	~~2,0~~	~~0,0~~	~~0,2~~	~~1,2~~	~~2,1~~	~~3,0~~	~~0,3~~	1,3	2,2	3,1	0,4
0				1					2				

2

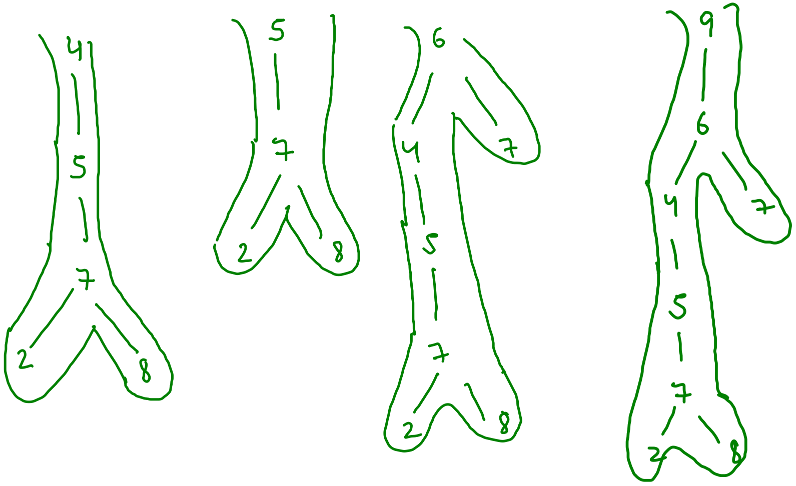
Mother Vertex



Given a Directed Graph, find a Mother Vertex in the Graph (if present).
A Mother Vertex is a vertex through which we can reach all the other vertices of the Graph.



Brute force



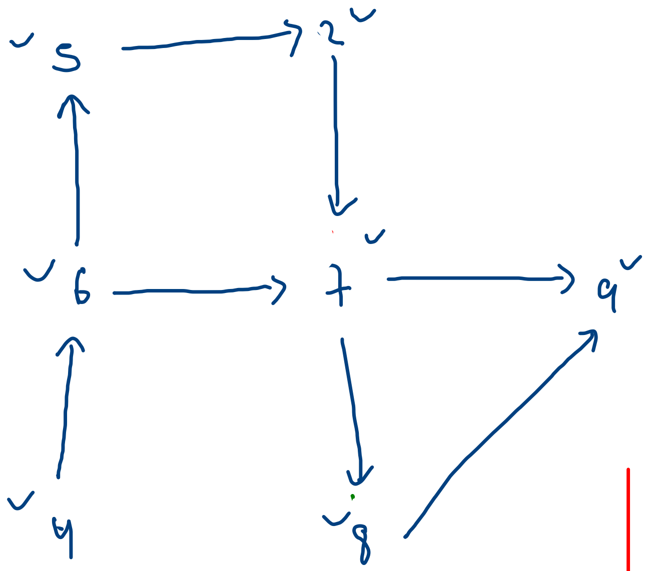
1 DFS or 1 BFS

$\boxed{O(V+E)}$

$V * O(DFS)$

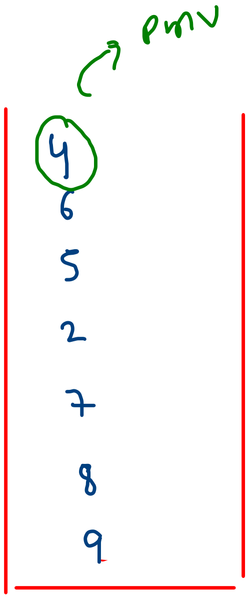
mother vertex is 9.

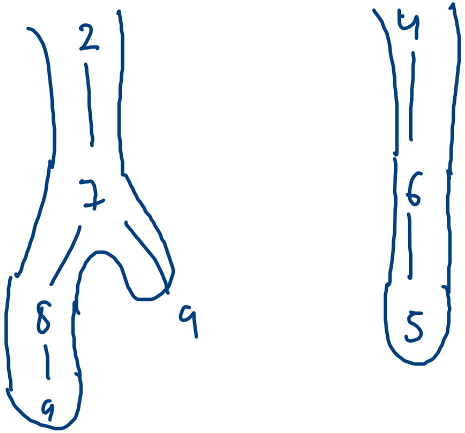
what ?

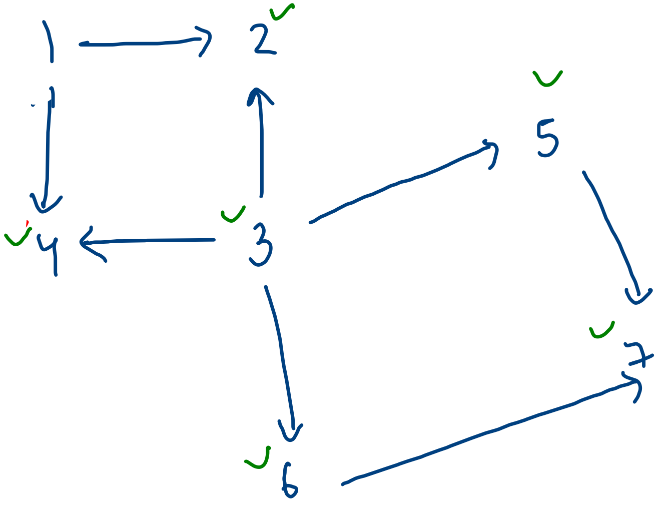


(i) perform dfs and add each node in post area in stack.

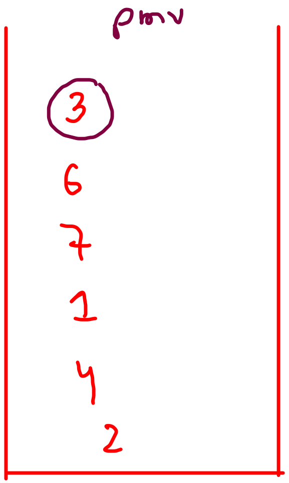
(ii) st.peek is a potential mother check (perform dfs) and find if it is really a mother vtx.

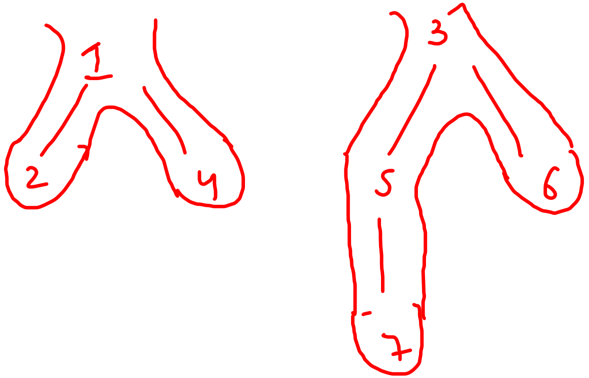




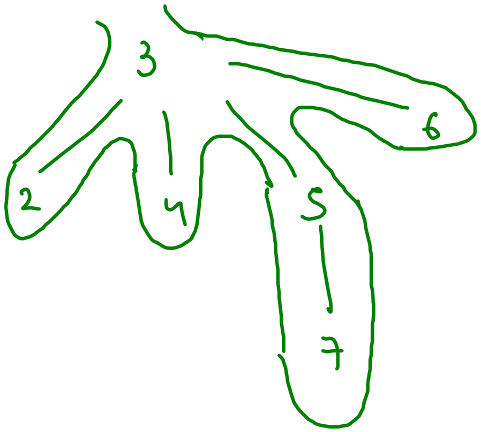


1

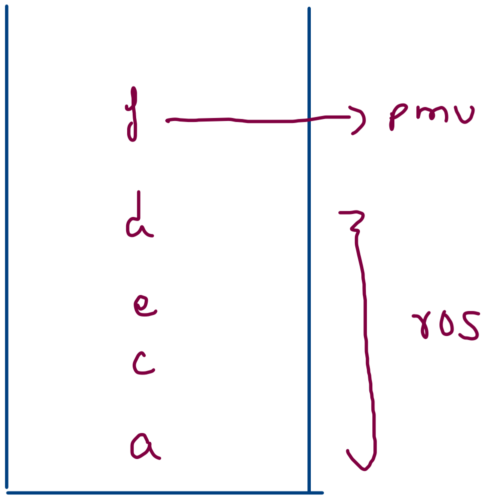




2

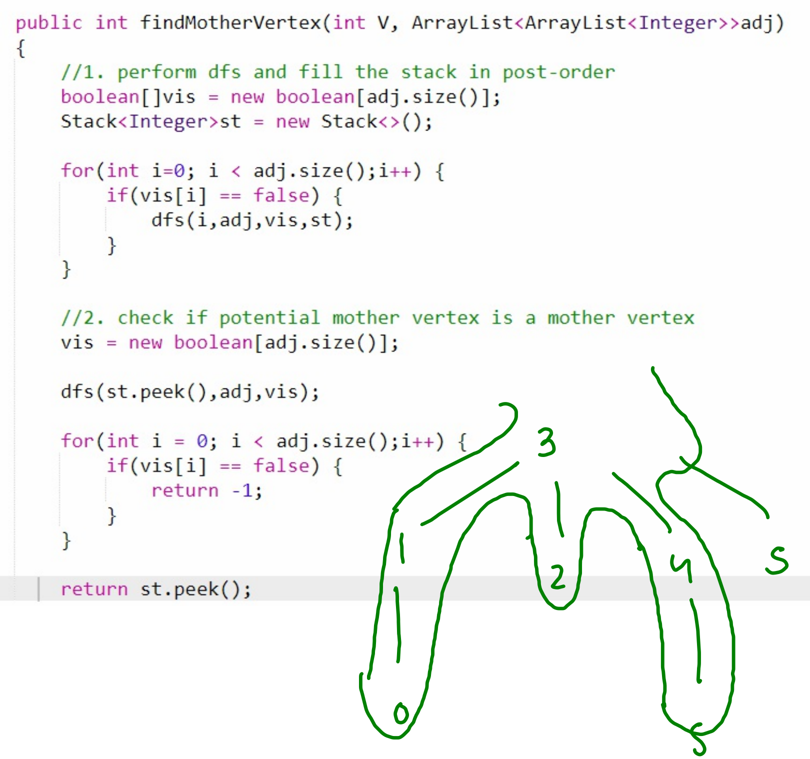


why ?



Cases

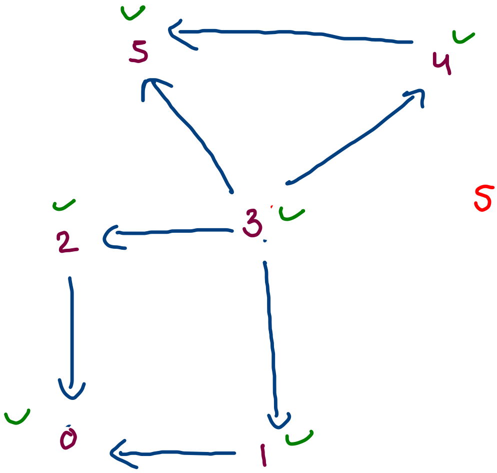
①	Pmv ✓	ros ✗	cover
②	pmv ✓	ros ✓	cover (one of the mv)
③	pmv ✗	ros ✗	cover
④	pmv ✗	ros ✓	impossible



```
public int findMotherVertex(int V, ArrayList<ArrayList<Integer>>adj)
{
    //1. perform dfs and fill the stack in post-order
    boolean[]vis = new boolean[adj.size()];
    Stack<Integer>st = new Stack<>();

    for(int i=0; i < adj.size();i++) {
        if(vis[i] == false) {
            dfs(i,adj,vis,st);
        }
    }

    //2. check if potential mother vertex is a mother vertex
    vis = new boolean[adj.size()];

    dfs(st.peek(),adj,vis);

    for(int i = 0; i < adj.size();i++) {
        if(vis[i] == false) {
            return -1;
        }
    }

    return st.peek();
```

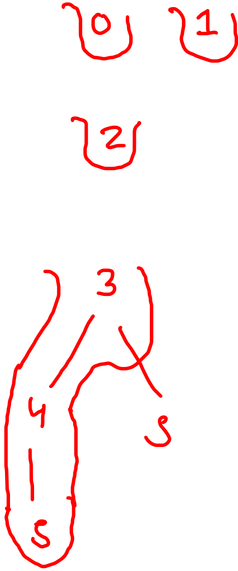
① pmv ✓   ras ✗

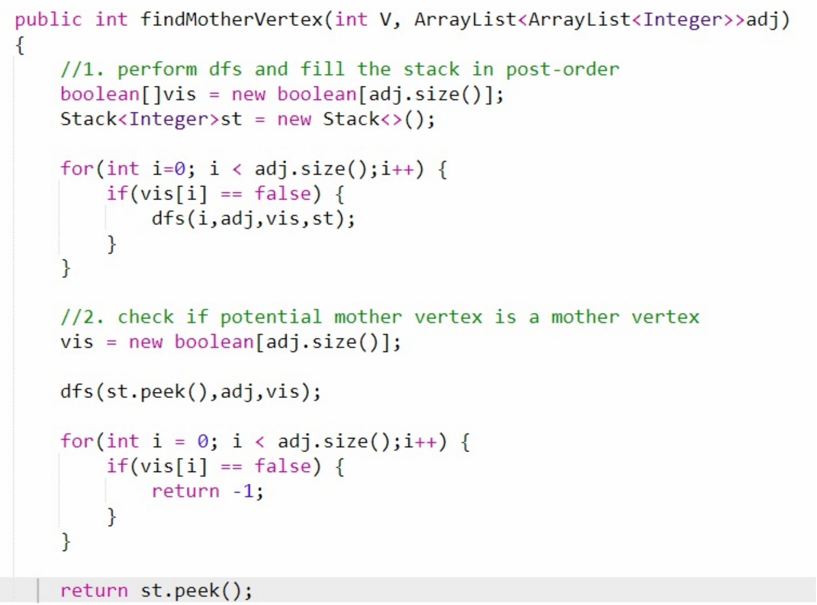
single mother vertex



step1.

3
4
5
2
1
0



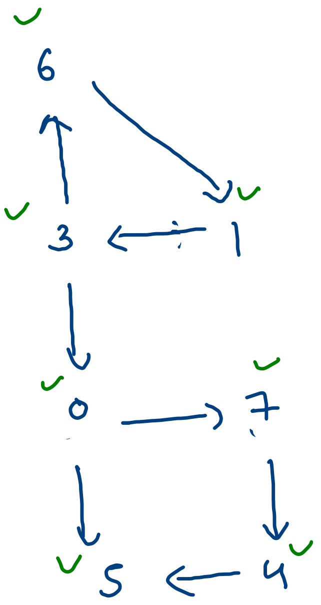


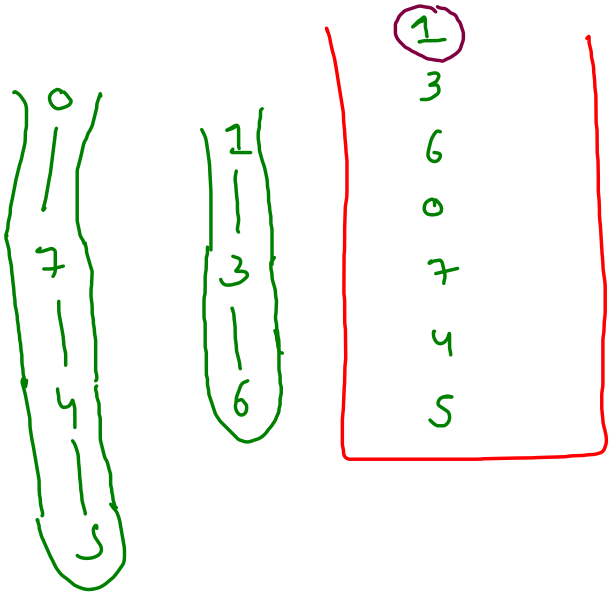
```java
public int findMotherVertex(int V, ArrayList<ArrayList<Integer>>adj)
{
    //1. perform dfs and fill the stack in post-order
    boolean[]vis = new boolean[adj.size()];
    Stack<Integer>st = new Stack<>();

    for(int i=0; i < adj.size();i++) {
        if(vis[i] == false) {
            dfs(i,adj,vis,st);
        }
    }

    //2. check if potential mother vertex is a mother vertex
    vis = new boolean[adj.size()];

    dfs(st.peek(),adj,vis);

    for(int i = 0; i < adj.size();i++) {
        if(vis[i] == false) {
            return -1;
        }
    }

    return st.peek();
```

② pmv ✓ ros ✓

more than one 1

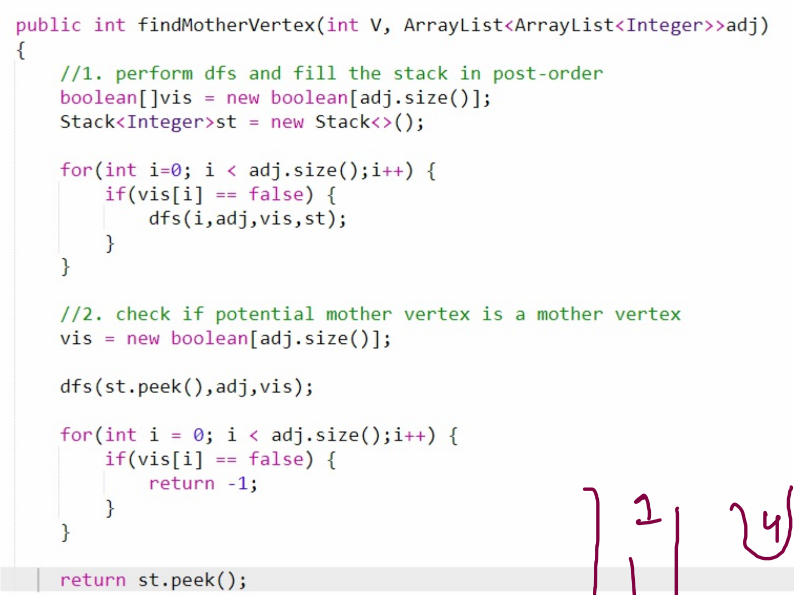
mother vertices .





1 pmv

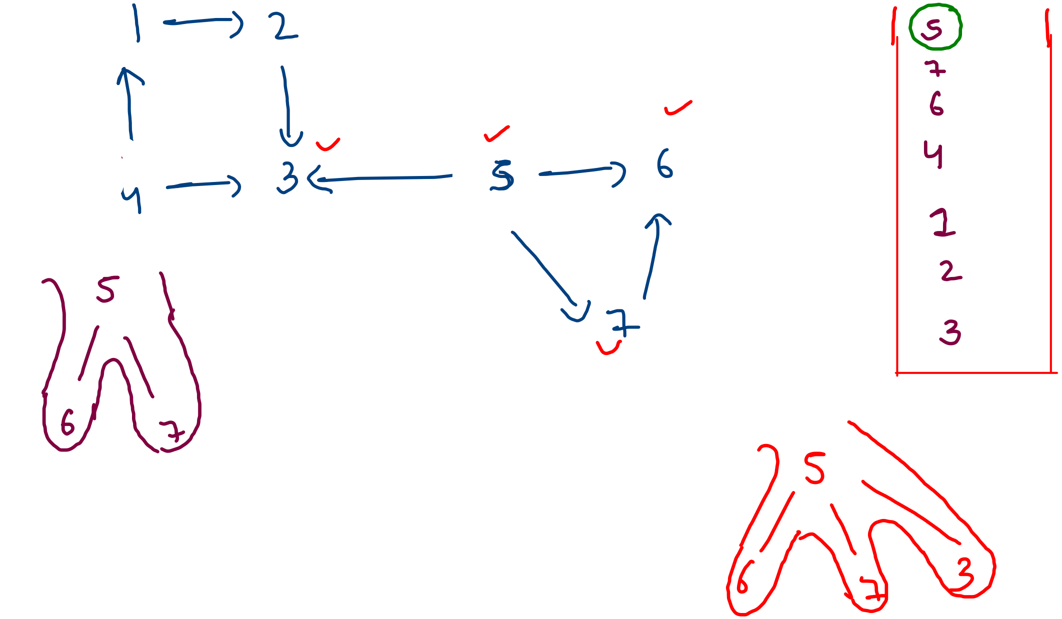
DFS ✓



```java
public int findMotherVertex(int V, ArrayList<ArrayList<Integer>>adj)
{
    //1. perform dfs and fill the stack in post-order
    boolean[]vis = new boolean[adj.size()];
    Stack<Integer>st = new Stack<>();

    for(int i=0; i < adj.size();i++) {
        if(vis[i] == false) {
            dfs(i,adj,vis,st);
        }
    }

    //2. check if potential mother vertex is a mother vertex
    vis = new boolean[adj.size()];

    dfs(st.peek(),adj,vis);

    for(int i = 0; i < adj.size();i++) {
        if(vis[i] == false) {
            return -1;
        }
    }

    return st.peek();
```

③ - pmv ✗ rus ✗

no mother vertex



1
2
3
4
5
6
7
5
7
6
4
1
2
3
5
6
7
3