

## Graph-1 Solutions

Solution 1:

Time Complexity :  $O(V+E)$

Space Complexity:  $O(V+E)$

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.Queue;
```

```
class Solution{
```

```
    public static void main(String arg[]){
```

```
        int V = 4;
        @SuppressWarnings("unchecked")
        ArrayList<Integer> adj[] = new ArrayList[V];
        for(int i = 0; i < 4; i++){
            adj[i] = new ArrayList<Integer>();
```

```
            addEdge(adj, 0, 1);
            addEdge(adj, 1, 2);
            addEdge(adj, 2, 0);
            addEdge(adj, 2, 3);
```

```
            if (isCyclicDisconntected(adj, V))
                System.out.println("Yes");
            else
                System.out.println("No");
```

```
        }
```

```
        static void addEdge(ArrayList<Integer> adj[], int u, int v){
```

```
            adj[u].add(v);
            adj[v].add(u);
```

```
        }
```

```
        static boolean isCyclicConntected(
```

```
            ArrayList<Integer> adj[], int s,
            int V, boolean visited[]){
```

```
            int parent[] = new int[V];
```

```

Arrays.fill(parent, -1);

Queue<Integer> q = new LinkedList<>();
visited[s] = true;
q.add(s);

while (!q.isEmpty()){
    int u = q.poll();
    for (int i = 0; i < adj[u].size(); i++){
        int v = adj[u].get(i);
        if (!visited[v]){
            visited[v] = true;
            q.add(v);
            parent[v] = u;
        }
        else if (parent[u] != v)
            return true;
    }
}
return false;
}

static boolean isCyclicDisconnected(
    ArrayList<Integer> adj[], int V){
    boolean visited[] = new boolean[V];
    Arrays.fill(visited, false);

    for (int i = 0; i < V; i++)
        if (!visited[i] &&
            isCyclicConntected(adj, i, V, visited))
            return true;
    return false;
}
}

```

**Solution 2:**

Time Complexity :  $O(n)$   
Space Complexity:  $O(1)$

```
import java.util.*;
```

```
class Solution{

static class Node{
    int data;
    Node left, right;
}

static class qltem{
    Node node;
    int depth;

    public qltem(Node node, int depth){
        this.node = node;
        this.depth = depth;
    }
}

static int minDepth(Node root){
    if (root == null)
        return 0;
    Queue<qltem> q = new LinkedList<>();
    qltem qi = new qltem(root, 1);
    q.add(qi);

    while (q.isEmpty() == false){
        qi = q.peek();
        q.remove();
        Node node = qi.node;
        int depth = qi.depth;
        if (node.left == null && node.right == null)
            return depth;
        if (node.left != null){
            qi.node = node.left;
            qi.depth = depth + 1;
            q.add(qi);
        }

        if (node.right != null){
            qi.node = node.right;
            qi.depth = depth + 1;
        }
    }
}
```

```
        q.add(qi);
    }
}
return 0;
}

static Node newNode(int data){
    Node temp = new Node();
    temp.data = data;
    temp.left = temp.right = null;
    return temp;
}

public static void main(String[] args){
    Node root = newNode(1);
    root.left = newNode(2);
    root.right = newNode(3);
    root.left.left = newNode(4);
    root.left.right = newNode(5);

    System.out.println(minDepth(root));
}
}
```

### Solution 3 :

Time Complexity :  $O(r \times c)$   
Space Complexity:  $O(r \times c)$

```
import java.util.LinkedList;
import java.util.Queue;
```

```
public class Solution{
    public final static int R = 3;
    public final static int C = 5;

    static class Ele{
        int x = 0;
```

```

        int y = 0;
        Ele(int x,int y){
            this.x = x;
            this.y = y;
        }
    }

    static boolean isValid(int i, int j){
        return (i >= 0 && j >= 0 && i < R && j < C);
    }

    static boolean isDelim(Ele temp){
        return (temp.x == -1 && temp.y == -1);
    }

    static boolean checkAll(int arr[][]){
        for (int i=0; i<R; i++)
            for (int j=0; j<C; j++)
                if (arr[i][j] == 1)
                    return true;
        return false;
    }

    static int Solution(int arr[][]){
        Queue<Ele> Q=new LinkedList<>();
        Ele temp;
        int ans = 0;
        for (int i=0; i < R; i++)
            for (int j=0; j < C; j++)
                if (arr[i][j] == 2)
                    Q.add(new Ele(i,j));

        Q.add(new Ele(-1,-1));

        while(!Q.isEmpty()){
            boolean flag = false;
            while(!isDelim(Q.peek())){
                temp = Q.peek();
                if(isValid(temp.x+1, temp.y) && arr[temp.x+1][temp.y] == 1){
                    if(!flag){

```

```

        ans++;
        flag = true;
    }
    arr[temp.x+1][temp.y] = 2;

    temp.x++;
    Q.add(new Ele(temp.x,temp.y));
    temp.x--;
}

if (isValid(temp.x-1, temp.y) && arr[temp.x-1][temp.y] == 1){
    if (!flag){
        ans++;
        flag = true;
    }
    arr[temp.x-1][temp.y] = 2;
    temp.x--;
    Q.add(new Ele(temp.x,temp.y)); // push this cell
    temp.x++;
}

if (isValid(temp.x, temp.y+1) && arr[temp.x][temp.y+1] == 1) {
    if (!flag){
        ans++;
        flag = true;
    }
    arr[temp.x][temp.y+1] = 2;
    temp.y++;
    Q.add(new Ele(temp.x,temp.y)); // Push this cell
    temp.y--;
}

if (isValid(temp.x, temp.y-1) && arr[temp.x][temp.y-1] == 1){
    if (!flag){
        ans++;
        flag = true;
    }
    arr[temp.x][temp.y-1] = 2;
    temp.y--;
}

```

to Queue

to Queue

```

        Q.add(new Ele(temp.x,temp.y)); // push this cell
        to Queue
    }
    Q.remove();

    }
    Q.remove();
    if (!Q.isEmpty())
    {
        Q.add(new Ele(-1,-1));
    }
}
return (checkAll(arr))? -1: ans;
}

public static void main(String[] args){
    int arr[][] = { {2, 1, 0, 2, 1},
                    {1, 0, 1, 2, 1},
                    {1, 0, 0, 2, 1}};

    int ans = Solution(arr);
    if(ans == -1)
        System.out.println("All oranges cannot rot");
    else
        System.out.println("Time required for all oranges to rot = " + ans);
}
}

```

#### Solution 4 :

Time Complexity :  $O(r \times c)$

Space Complexity:  $O(r \times c)$

```

import java.io.*;
import java.util.*;

```

```

class Solution {

```

```

static int ROW, COL, count;
static boolean isSafe(int[][] M, int row, int col,
                      boolean[][] visited)
{
    return (
        (row >= 0) && (row < ROW) && (col >= 0)
        && (col < COL)
        && (M[row][col] == 1 && !visited[row][col]));
}

static void DFS(int[][] M, int row, int col,
                boolean[][] visited){
    int[] rowNbr = { -1, -1, -1, 0, 0, 1, 1, 1 };
    int[] colNbr = { -1, 0, 1, -1, 1, -1, 0, 1 };

    visited[row][col] = true;
    for (int k = 0; k < 8; k++) {
        if (isSafe(M, row + rowNbr[k], col + colNbr[k],
                  visited)) {
            count++;
            DFS(M, row + rowNbr[k], col + colNbr[k],
              visited);
        }
    }
}

static int largestRegion(int[][] M)    {

    boolean[][] visited = new boolean[ROW][COL];
    int result = 0;
    for (int i = 0; i < ROW; i++) {
        for (int j = 0; j < COL; j++) {
            if (M[i][j] == 1 && !visited[i][j]) {
                count = 1;
                DFS(M, i, j, visited);
                result = Math.max(result, count);
            }
        }
    }
    return result;
}

```



```

    }

    public static void main(String args[]){
        int M[][] = { { 0, 0, 1, 1, 0 },
                      { 1, 0, 1, 1, 0 },
                      { 0, 1, 0, 0, 0 },
                      { 0, 0, 0, 0, 1 } };

        ROW = 4;
        COL = 5;
        System.out.println(largestRegion(M));
    }
}

```

### Solution 5 :

Time Complexity :  $O(n^2 \cdot m)$

Space Complexity:  $O(n \cdot m)$

```
import java.util.*;
```

```

class Solution {
    static int shortestChainLen(String start,
                                String target,
                                Set<String> D){

```

```

        if(start == target)
            return 0;
        if (!D.contains(target))
            return 0;
        int level = 0, wordlength = start.length();
        Queue<String> Q = new LinkedList<>();
        Q.add(start);
        while (!Q.isEmpty()){
            ++level;
            int sizeofQ = Q.size();
            for (int i = 0; i < sizeofQ; ++i){

```

```

        char []word = Q.peek().toCharArray();
        Q.remove();
        for (int pos = 0; pos < wordlength; ++pos){
            char orig_char = word[pos];
            for (char c = 'a'; c <= 'z'; ++c){
                word[pos] = c;
                if (String.valueOf(word).equals(target))
                    return level + 1;
                if (!D.contains(String.valueOf(word)))
                    continue;
                D.remove(String.valueOf(word));
                Q.add(String.valueOf(word));
            }
            word[pos] = orig_char;
        }
    }
}

return 0;
}

public static void main(String[] args){
    Set<String> D = new HashSet<String>();
    D.add("poon");
    D.add("plee");
    D.add("same");
    D.add("poie");
    D.add("plie");
    D.add("poin");
    D.add("plea");
    String start = "toon";
    String target = "plea";
    System.out.print("Length of shortest chain is: "
        + shortestChainLen(start, target, D));
}
}

```

**APNA**  
**COLLEGE**