Graph-02 Solutions

Solution 1:

Time Complexity : o(v+e)
Space Complexity: o(v+e)

```
import java.util.*;

class Solution{

static void addEdge(int u, int v,
 ArrayList<ArrayList<Integer>> adj){
   adj.get(u).add(v);
}

static void DFSUtil(ArrayList<ArrayList<Integer>> g,
                    int v, boolean[] visited){
   visited[v] = true;

   for(int x : g.get(v)){
      if (!visited[x]){
         DFSUtil(g, x, visited);
      }
   }
}

static int motherVertex(ArrayList<ArrayList<Integer>>g,
                        int V){

   boolean[] visited = new boolean[V];
   int v = -1;

   for(int i = 0; i < V; i++){
      if (!visited[i]){
         DFSUtil(g, i, visited);
         v = i;
      }
   }

   boolean[] check = new boolean[V];
   DFSUtil(g, v, check);
```

```
    for(boolean val : check){
        if (!val){
            return -1;
        }
    }
    return v;
}

 public static void main(String[] args){
    int V = 7;
    int E = 8;

    ArrayList<
    ArrayList<Integer>> adj = new ArrayList<
                            ArrayList<Integer>>();
    for(int i = 0; i < V; i++){
        adj.add(new ArrayList<Integer>());
    }
    addEdge(0, 1,adj);
    addEdge(0, 2,adj);
    addEdge(1, 3,adj);
    addEdge(4, 1,adj);
    addEdge(6, 4,adj);
    addEdge(5, 6,adj);
    addEdge(5, 2,adj);
    addEdge(6, 0,adj);

    System.out.println("The mother vertex is " +
                    motherVertex(adj, V));
}
}
```


Solution 2:

Time Complexity : o(v+e)
Space Complexity: o(1)

```
import java.io.*;
import java.lang.*;
import java.util.*;

class Solution {
        int V, E;
```

```
Edge edge[];

class Edge {
        int src, dest;
};

Solution(int v, int e){
        V = v;
        E = e;
        edge = new Edge[E];
        for (int i = 0; i < e; ++i)
                edge[i] = new Edge();
}

int find(int parent[], int i){
        if (parent[i] == i)
                return i;
        return find(parent, parent[i]);
}

void Union(int parent[], int x, int y){
        parent[x] = y;
}

int isCycle(Solution graph){
        int parent[] = new int[graph.V];

        for (int i = 0; i < graph.V; ++i)
                parent[i] = i;

        for (int i = 0; i < graph.E; ++i) {
                int x = graph.find(parent, graph.edge[i].src);
                int y = graph.find(parent, graph.edge[i].dest);

                if (x == y)
                        return 1;

                graph.Union(parent, x, y);
        }
        return 0;
```

```
	}

	public static void main(String[] args){
		int V = 3, E = 3;
		Solution graph = new Solution(V, E);
		graph.edge[0].src = 0;
		graph.edge[0].dest = 1;
		graph.edge[1].src = 1;
		graph.edge[1].dest = 2;
		graph.edge[2].src = 0;
		graph.edge[2].dest = 2;

		if (graph.isCycle(graph) == 1)
			System.out.println("Graph contains cycle");
		else
			System.out.println(
				"Graph doesn't contain cycle");
	}
}
```


Solution 3 :

Time Complexity : o(n)
Space Complexity: o(n)

```
import java.util.*;

public class Solution {

	static class pair {
		int first, second;

		pair(int first, int second){
			this.first = first;
			this.second = second;
		}
	}

	static ArrayList<ArrayList<Integer> >
```

```
make_graph(int numTasks, Vector<pair> prerequisites){
        ArrayList<ArrayList<Integer> > graph
                = new ArrayList<ArrayList<Integer> >(numTasks);

        for (int i = 0; i < numTasks; i++) {
                graph.add(new ArrayList<Integer>());
        }

        for (pair pre : prerequisites)
                graph.get(pre.second).add(pre.first);

        return graph;
}

static int[] compute_indegree(
        ArrayList<ArrayList<Integer> > graph)
{
        int degrees[] = new int[graph.size()];

        for (ArrayList<Integer> neighbors : graph)
                for (int neigh : neighbors)
                        degrees[neigh]++;

        return degrees;
}

static boolean canFinish(int numTasks,
                                                Vector<pair> prerequisites){
        ArrayList<ArrayList<Integer> > graph
                = make_graph(numTasks, prerequisites);
        int degrees[] = compute_indegree(graph);

        for (int i = 0; i < numTasks; i++) {
                int j = 0;
                for (; j < numTasks; j++)
                        if (degrees[j] == 0)
                                break;

                if (j == numTasks)
                        return false;
```

```
                degrees[j] = -1;
                for (int neigh : graph.get(j))
                    degrees[neigh]--;
            }

            return true;
        }

        public static void main(String args[]){
            int numTasks = 4;
            Vector<pair> prerequisites = new Vector<pair>();

            prerequisites.add(new pair(1, 0));
            prerequisites.add(new pair(2, 1));
            prerequisites.add(new pair(3, 2));

            if (canFinish(numTasks, prerequisites)) {
                System.out.println(
                    "Possible to finish all tasks");
            }
            else {
                System.out.println(
                    "Impossible to finish all tasks");
            }
        }
}
```


Solution 4 :

Time Complexity : o(v+e)
Space Complexity: o(v+e)

```
class Solution {
    public String alienOrder(String[] words) {
        Map<Character, Set<Character>> map = new HashMap<>();
        Map<Character, Integer> degree = new HashMap<>();
        String result = "";
        if (words == null || words.length == 0) { return result; }
```

```
for (String s: words) {
    for (char c: s.toCharArray()) {
        degree.put(c, 0);
    }
}

for (int i = 0; i < words.length - 1; i++) {
    String curr = words[i];
    String next = words[i + 1];
    int min = Math.min(curr.length(), next.length());
    for (int j = 0; j < min; j++) {
        char c1 = curr.charAt(j);
        char c2 = next.charAt(j);
        if (c1 != c2) {
            Set<Character> set = map.getOrDefault(c1, new HashSet<>());
            if (!set.contains(c2)) {
                set.add(c2);
                map.put(c1, set);
                degree.put(c2, degree.get(c2) + 1); // update c2, c1 < c2
            }
            break;
        }
    }
}

LinkedList<Character> q = new LinkedList<>();
for (char c: degree.keySet()) {
    if (degree.get(c) == 0) {
        q.add(c);
    }
}

while (!q.isEmpty()) {
    char c = q.poll();
    result += c;
    if (map.containsKey(c)) {
        for (char next: map.get(c)) {
            degree.put(next, degree.get(next) - 1);
            if (degree.get(next) == 0) {
                q.offer(next);
```

```
                }
            }
        }
    }

    return result.length() == degree.size() ? result : "";
  }
}
```

Solution 5 :

Time Complexity : o(n*m)
Space Complexity: o(n*m)

```
import java.util.*;

class Solution{

static void dfs(int[][] matrix, boolean[][] visited,
                int x, int y, int n, int m,
                boolean hasCornerCell){

  if (x < 0 || y < 0 || x >= n || y >= m ||
    visited[x][y] == true || matrix[x][y] == 0)
    return;

  if (x == 0 || y == 0 ||
    x == n - 1 || y == m - 1){
    if (matrix[x][y] == 1)
      hasCornerCell = true;
  }

  visited[x][y] = true;

  dfs(matrix, visited, x + 1, y, n, m,
    hasCornerCell);
  dfs(matrix, visited, x, y + 1, n, m,
```

```
        hasCornerCell);
    dfs(matrix, visited, x - 1, y, n, m,
        hasCornerCell);
    dfs(matrix, visited, x, y - 1, n, m,
        hasCornerCell);
}

static int countClosedIsland(int[][] matrix, int n,
                             int m){

    boolean[][] visited = new boolean[n][m];
    int result = 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) &&
                 matrix[i][j] == 1 &&
                 visited[i][j] == false) {

                boolean hasCornerCell = false;
                dfs(matrix, visited, i, j, n, m,
                    hasCornerCell);
                if (!hasCornerCell)
                    result = result + 1;
            }
        }
    }
    return result;
}

public static void main(String[] args){
    int N = 5, M = 8;

    int[][] matrix = { { 0, 0, 0, 0, 0, 0, 0, 1 },
                       { 0, 1, 1, 1, 1, 0, 0, 1 },
                       { 0, 1, 0, 1, 0, 0, 0, 1 },
                       { 0, 1, 1, 1, 1, 0, 1, 0 },
                       { 0, 0, 0, 0, 0, 0, 0, 1 } };
    System.out.print(countClosedIsland(matrix, N, M));
```

```
    }

}
```