Graph Data Structure
By Apna College

Note - There are 2 sections in this document
- **Section 1 - Graph Codes** (Page 1 - 24)
- **Section 2 - Graph Assignments** (page 25 - 30)

Graph Codes

Part1

BFS

```
import java.util.*;

public class BFS {
    static class Edge {
        int src;
        int dest;
        int wt;
        public Edge(int s, int d, int w) {
            this.src = s;
            this.dest = d;
            this.wt = w;
        }
    }

    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }

        graph[0].add(new Edge(0, 1, 1));
        graph[0].add(new Edge(0, 2, 1));

        graph[1].add(new Edge(1, 0, 1));
        graph[1].add(new Edge(1, 3, 1));

        graph[2].add(new Edge(2, 0, 1));
```

```java
        graph[2].add(new Edge(2, 4, 1));

        graph[3].add(new Edge(3, 1, 1));
        graph[3].add(new Edge(3, 4, 1));
        graph[3].add(new Edge(3, 5, 1));

        graph[4].add(new Edge(4, 2, 1));
        graph[4].add(new Edge(4, 3, 1));
        graph[4].add(new Edge(4, 5, 1));

        graph[5].add(new Edge(5, 3, 1));
        graph[5].add(new Edge(5, 4, 1));
        graph[5].add(new Edge(5, 6, 1));

        graph[5].add(new Edge(6, 5, 1));
    }

    public static void bfs(ArrayList<Edge> graph[], int V) {
        boolean visited[] = new boolean[V];
        Queue<Integer> q = new LinkedList<>();
        q.add(0); //Source = 0

        while(!q.isEmpty()) {
            int curr = q.remove();
            if(!visited[curr]) {
                System.out.print(curr+" ");
                visited[curr] = true;

                for(int i=0; i<graph[curr].size(); i++) {
                    Edge e = graph[curr].get(i);
                    q.add(e.dest);
                }
            }
        }

        System.out.println();
    }
    public static void main(String args[]) {
        /*
              1 --- 3
             /      | \
            0       |   5 -- 6
```

```
          \        | /
           2 ---- 4

        */

       int V = 7;
       ArrayList<Edge> graph[] = new ArrayList[V];
       createGraph(graph);

       bfs(graph, V);
   }
}
```

DFS

```
import java.util.*;

public class DFS {
   static class Edge {
       int src;
       int dest;
       int wt;
       public Edge(int s, int d, int w) {
           this.src = s;
           this.dest = d;
           this.wt = w;
       }
   }

   static void createGraph(ArrayList<Edge> graph[]) {
       for(int i=0; i<graph.length; i++) {
           graph[i] = new ArrayList<>();
       }

       graph[0].add(new Edge(0, 1, 1));
       graph[0].add(new Edge(0, 2, 1));

       graph[1].add(new Edge(1, 0, 1));
       graph[1].add(new Edge(1, 3, 1));
```

```java
        graph[2].add(new Edge(2, 0, 1));
        graph[2].add(new Edge(2, 4, 1));

        graph[3].add(new Edge(3, 1, 1));
        graph[3].add(new Edge(3, 4, 1));
        graph[3].add(new Edge(3, 5, 1));

        graph[4].add(new Edge(4, 2, 1));
        graph[4].add(new Edge(4, 3, 1));
        graph[4].add(new Edge(4, 5, 1));

        graph[5].add(new Edge(5, 3, 1));
        graph[5].add(new Edge(5, 4, 1));
        graph[5].add(new Edge(5, 6, 1));

        graph[5].add(new Edge(6, 5, 1));
    }

    public static void dfs(ArrayList<Edge> graph[], int curr, boolean visited[]) {
        if(visited[curr]) {
            return;
        }

        System.out.print(curr+" ");
        visited[curr] = true;
        for(int i=0; i<graph[curr].size(); i++) {
            Edge e = graph[curr].get(i);
            dfs(graph, e.dest, visited);
        }
    }
    public static void main(String args[]) {
        /*
              1 --- 3
             /      | \
            0       |   5 -- 6
            \       | /
             2 ---- 4

         */

        int V = 7;
        ArrayList<Edge> graph[] = new ArrayList[V];
```

```
        createGraph(graph);

        dfs(graph, 0, new boolean[V]);
    }
}
```

All Paths

```
import java.util.*;
//For Youtube Lecture
public class PrintAllPaths {
    static class Edge {
        int src;
        int dest;
        public Edge(int s, int d) {
            this.src = s;
            this.dest = d;
        }
    }

    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }

        graph[0].add(new Edge(0, 1));
        graph[0].add(new Edge(0, 2));

        graph[1].add(new Edge(1, 0));
        graph[1].add(new Edge(1, 3));

        graph[2].add(new Edge(2, 0));
        graph[2].add(new Edge(2, 4));

        graph[3].add(new Edge(3, 1));
        graph[3].add(new Edge(3, 4));
        graph[3].add(new Edge(3, 5));

        graph[4].add(new Edge(4, 2));
        graph[4].add(new Edge(4, 3));
        graph[4].add(new Edge(4, 5));
```

```java
        graph[5].add(new Edge(5, 3));
        graph[5].add(new Edge(5, 4));
        graph[5].add(new Edge(5, 6));

        graph[6].add(new Edge(6, 5));
    }

    public static void printAllPaths(ArrayList<Edge> graph[], int src, int tar, String
path, boolean vis[]) {

        if(src == tar) {
            System.out.println(path);
            return;
        }

        for(int i=0; i<graph[src].size(); i++) {
            Edge e = graph[src].get(i);
            if(!vis[e.dest]) {
                vis[e.dest] = true;
                printAllPaths(graph, e.dest, tar, path+"->"+e.dest, vis);
                vis[e.dest] = false;
            }
        }

    }
    public static void main(String args[]) {
        int V = 7;
        ArrayList<Edge> graph[] = new ArrayList[V];
        createGraph(graph);
        int src = 0;
        int tar = 5;
        boolean vis[] = new boolean[V];
        vis[src] = true;
        printAllPaths(graph, src, tar, ""+src, vis);
    }
}
```

Part2

Cycle Detection (Undirected Graph)

```
import java.util.*;

public class CycleUndirected {
   static class Edge {
       int src;
       int dest;
       public Edge(int s, int d) {
           this.src = s;
           this.dest = d;
       }
   }

   static void createGraph(ArrayList<Edge> graph[]) {
       for(int i=0; i<graph.length; i++) {
           graph[i] = new ArrayList<>();
       }

       graph[0].add(new Edge(0, 1));
       graph[0].add(new Edge(0, 2));
       graph[0].add(new Edge(0, 3));

       graph[1].add(new Edge(1, 0));
       graph[1].add(new Edge(1, 2));

       graph[2].add(new Edge(2, 0));
       graph[2].add(new Edge(2, 1));

       graph[3].add(new Edge(3, 0));
       graph[3].add(new Edge(3, 4));

       graph[4].add(new Edge(4, 3));
   }

   public static boolean isCyclicUtil(ArrayList<Edge>[] graph, boolean vis[], int
curr, int par) {
       vis[curr] = true;

       for(int i=0; i<graph[curr].size(); i++) {
           Edge e = graph[curr].get(i);
           //case1
           if(vis[e.dest] && e.dest != par) {
```

```java
                boolean isCycle = isCyclicUtil(graph, vis, e.dest, curr);
                if(isCycle)
                    return true;
            } else if(e.dest == par) {
                //case 2
                continue;
            } else {
                //case 3
                return true;
            }
        }

        return false;
    }

    //O(V+E)
    public static boolean isCyclic(ArrayList<Edge>[] graph, boolean vis[]) {
        for(int i=0; i<graph.length; i++) {
            if(isCyclicUtil(graph, vis, i, -1)) {
                return true;
            }
        }
        return false;
    }

    public static void main(String args[]) {
        /*
                0 ------- 3
               /|         |
             /  |         |
            1   |         4
             \  |
              \ |
                2

         */

        int V = 5;
        ArrayList<Edge> graph[] = new ArrayList[V];
        createGraph(graph);

        System.out.println(isCyclic(graph, new boolean[V]));
```

```
    }
}
```

Cycle Detection (Directed Graph)

```
import java.util.*;

public class CycleDirected {
    static class Edge {
        int src;
        int dest;
        public Edge(int s, int d) {
            this.src = s;
            this.dest = d;
        }
    }

    //graph1 - true
    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }

        graph[0].add(new Edge(0, 2));

        graph[1].add(new Edge(1, 0));

        graph[2].add(new Edge(2, 3));

        graph[3].add(new Edge(3, 0));
    }

    //graph2 - false
    // static void createGraph(ArrayList<Edge> graph[]) {
    //     for(int i=0; i<graph.length; i++) {
    //         graph[i] = new ArrayList<>();
    //     }

    //     graph[0].add(new Edge(0, 1));
    //     graph[0].add(new Edge(0, 2));
```

```
//     graph[1].add(new Edge(1, 3));

//     graph[2].add(new Edge(2, 3));
// }

public static boolean isCyclicUtil(ArrayList<Edge>[] graph, int curr, boolean
vis[], boolean stack[]) {
    vis[curr] = true;
    stack[curr] = true;

    for(int i=0; i<graph[curr].size(); i++) {
        Edge e = graph[curr].get(i);
        if(stack[e.dest]) { //cycle exists
            return true;
        } else if(!vis[e.dest] && isCyclicUtil(graph, e.dest, vis, stack)) {
            return true;
        }
    }
    stack[curr] = false;
    return false;
}

//O(V + E)
public static boolean isCyclic(ArrayList<Edge>[] graph) {
    boolean vis[] = new boolean[graph.length];

    for(int i=0; i<graph.length; i++) {
        if(vis[i] == false) {
            boolean cycle = isCyclicUtil(graph, i, vis, new boolean[vis.length]);
            if(cycle) {
                return true;
            }
        }
    }
    return false;
}
public static void main(String args[]) {
    int V = 4;
    ArrayList<Edge> graph[] = new ArrayList[V];
    createGraph(graph);
```

```
        System.out.println(isCyclic(graph));
    }
}
```

Topological Sorting

```
import java.util.*;

public class TopologicalSort {
    static class Edge {
        int src;
        int dest;
        public Edge(int s, int d) {
            this.src = s;
            this.dest = d;
        }
    }

    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }

        graph[2].add(new Edge(2, 3));

        graph[3].add(new Edge(3, 1));

        graph[4].add(new Edge(4, 0));
        graph[4].add(new Edge(4, 1));

        graph[5].add(new Edge(5, 0));
        graph[5].add(new Edge(5, 2));
    }

    public static void topoSortUtil(ArrayList<Edge> graph[], int curr, boolean vis[],
Stack<Integer> s) {
        vis[curr] = true;

        for(int i=0; i<graph[curr].size(); i++) {
            Edge e = graph[curr].get(i);
            if(!vis[e.dest]) {
                topoSortUtil(graph, e.dest, vis, s);
```

```
            }
        }

        s.push(curr);
    }

    //O(V+E)
    public static void topoSort(ArrayList<Edge> graph[]) {
        boolean vis[] = new boolean[graph.length];
        Stack<Integer> s = new Stack<>();

        for(int i=0; i<graph.length; i++) {
            if(!vis[i]) {
                topoSortUtil(graph, i, vis, s);
            }
        }

        while(!s.isEmpty()) {
            System.out.print(s.pop()+" ");
        }
    }

    public static void main(String args[]) {
        int V = 6;
        ArrayList<Edge> graph[] = new ArrayList[V];
        createGraph(graph);

        topoSort(graph);
    }
}
```

Part3

Dijkstra's Algorithm (Shortest Distance)

```
import java.security.Permissions;
import java.util.*;

public class Dijkstras {
    static class Edge {
```

```
    int src;
    int dest;
    int wt;
    public Edge(int s, int d, int w) {
        this.src = s;
        this.dest = d;
        this.wt = w;
    }
}
static void createGraph(ArrayList<Edge> graph[]) {
    for(int i=0; i<graph.length; i++) {
        graph[i] = new ArrayList<>();
    }

    graph[0].add(new Edge(0, 1, 2));
    graph[0].add(new Edge(0, 2, 4));

    graph[1].add(new Edge(1, 3, 7));
    graph[1].add(new Edge(1, 2, 1));

    graph[2].add(new Edge(2, 4, 3));

    graph[3].add(new Edge(3, 5, 1));

    graph[4].add(new Edge(4, 3, 2));
    graph[4].add(new Edge(4, 5, 5));
}

static class Pair implements Comparable<Pair> {
    int n;
    int path;

    public Pair(int n, int path) {
        this.n = n;
        this.path = path;
    }

    @Override
    public int compareTo(Pair p2) {
        return this.path - p2.path;
    }
}
```

```java
public static int[] dijkstra(ArrayList<Edge> graph[], int src) {
    PriorityQueue<Pair> pq = new PriorityQueue<>();
    int dist[] = new int[graph.length];
    boolean vis[] = new boolean[graph.length];
    for(int i=0; i<dist.length; i++) {
        if(i != src) {
            dist[i] = Integer.MAX_VALUE;
        }
    }
    pq.add(new Pair(src, 0));
    while(!pq.isEmpty()) {
        Pair curr = pq.remove();
        if(!vis[curr.n]) {
            vis[curr.n] = true;

            for(int i=0; i<graph[curr.n].size(); i++) {
                Edge e = graph[curr.n].get(i);
                int u = e.src;
                int v = e.dest;
                if(!vis[v] && dist[u]+e.wt < dist[v]) {
                    dist[v] = dist[u] + e.wt;
                    pq.add(new Pair(v, dist[v]));
                }

            }
        }
    }

    return dist;
}
public static void main(String args[]) {
    int V = 6;
    ArrayList<Edge> graph[] = new ArrayList[V];
    createGraph(graph);
    int src = 0;

    int dist[] = dijkstra(graph, src);
    for(int i=0; i<dist.length; i++) {
        System.out.println(dist[i]+" ");
    }
```

```
    }
}
```

Bellman Ford Algorithm (Shortest Distance)

```java
import java.util.*;

public class BellmanFord {
    static class Edge {
        int src;
        int dest;
        int wt;
        public Edge(int s, int d, int w) {
            this.src = s;
            this.dest = d;
            this.wt = w;
        }
    }
    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }

        graph[0].add(new Edge(0, 1, 2));
        graph[0].add(new Edge(0, 2, 4));

        graph[1].add(new Edge(1, 2, -4));

        graph[2].add(new Edge(2, 3, 2));

        graph[3].add(new Edge(3, 4, 4));

        graph[4].add(new Edge(4, 1, -1));
    }

    public static void bellmanFord(ArrayList<Edge> graph[], int src) {
        int dist[] = new int[graph.length];
        for(int i=0; i<dist.length; i++) {
            if(i != src)
                dist[i] = Integer.MAX_VALUE;
```

```java
    }

    //O(V)
    for(int i=0; i<graph.length-1; i++) {
        //edges - O(E)
        for(int j=0; j<graph.length; j++) {
            for(int k=0; k<graph[j].size(); k++) {
                Edge e = graph[j].get(k);
                int u = e.src;
                int v = e.dest;
                int wt = e.wt;
                if(dist[u] != Integer.MAX_VALUE && dist[u]+wt < dist[v]) {
                    dist[v] = dist[u] + wt;
                }
            }
        }
    }

    //Detecting Negative Weight Cycle
    for(int j=0; j<graph.length; j++) {
        for(int k=0; k<graph[j].size(); k++) {
            Edge e = graph[j].get(k);
            int u = e.src;
            int v = e.dest;
            int wt = e.wt;
            if(dist[u] != Integer.MAX_VALUE && dist[u]+wt < dist[v]) {
                System.out.println("negative weight cycle exists");
                break;
            }
        }
    }

    for(int i=0; i<dist.length; i++) {
        System.out.print(dist[i]+" ");
    }
    System.out.println();
}

public static void main(String args[]) {
    int V = 5;
    ArrayList<Edge> graph[] = new ArrayList[V];
    createGraph(graph);
```

```
        int src = 0;
        bellmanFord(graph, src);
    }
}
```

Part4

Prim's Algorithm (MST)

```
import java.util.*;

public class PrimsAlgorithm {
    static class Edge {
        int src;
        int dest;
        int wt;
        public Edge(int s, int d, int w) {
            this.src = s;
            this.dest = d;
            this.wt = w;
        }
    }
    static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<>();
        }

        graph[0].add(new Edge(0, 1, 10));
        graph[0].add(new Edge(0, 2, 15));
        graph[0].add(new Edge(0, 3, 30));

        graph[1].add(new Edge(1, 0, 10));
        graph[1].add(new Edge(1, 3, 40));

        graph[2].add(new Edge(2, 0, 15));
        graph[2].add(new Edge(2, 3, 50));

        graph[3].add(new Edge(3, 1, 40));
        graph[3].add(new Edge(3, 2, 50));
```

```
}

static class Pair implements Comparable<Pair> {
    int v;
    int wt;

    public Pair(int v, int wt) {
        this.v = v;
        this.wt = wt;
    }
    @Override
    public int compareTo(Pair p2) {
        return this.wt - p2.wt;
    }
}

//O(ElogE)
public static void primAlgo(ArrayList<Edge> graph[]) {
    boolean vis[] = new boolean[graph.length];
    PriorityQueue<Pair> pq = new PriorityQueue<>();
    pq.add(new Pair(0, 0));
    int cost = 0;

    while(!pq.isEmpty()) {
        Pair curr = pq.remove();
        if(!vis[curr.v]) {
            vis[curr.v] = true;
            cost += curr.wt;

            for(int i=0; i<graph[curr.v].size(); i++) {
                Edge e = graph[curr.v].get(i);
                if(!vis[e.dest]) {
                    pq.add(new Pair(e.dest, e.wt));
                }
            }
        }
    }

    System.out.println(cost);
}

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

```
        int V = 4;
        ArrayList<Edge> graph[] = new ArrayList[V];
        createGraph(graph);

        primAlgo(graph);
    }
}
```

Kosaraju's Algorithm (Strongly Connected Components)

```
import java.util.*;

public class Kosaraju {
    static class Edge {
        int src;
        int dest;

        public Edge(int s, int d) {
            this.src = s;
            this.dest = d;
        }
    }

    public static void createGraph(ArrayList<Edge> graph[]) {
        for(int i=0; i<graph.length; i++) {
            graph[i] = new ArrayList<Edge>();
        }

        graph[0].add(new Edge(0, 2));
        graph[0].add(new Edge(0, 3));

        graph[1].add(new Edge(1, 0));

        graph[2].add(new Edge(2, 1));

        graph[3].add(new Edge(3, 4));
    }

    public static void topSort(ArrayList<Edge> graph[], int curr, Stack<Integer> s,
boolean vis[]) {
```

```
    vis[curr] = true;

    for(int i=0; i<graph[curr].size(); i++) {
        Edge e = graph[curr].get(i);
        if(!vis[e.dest]) {
            topSort(graph, e.dest, s, vis);
        }
    }

    s.push(curr);
}

public static void dfs(ArrayList<Edge> graph[], boolean vis[], int curr) {
    vis[curr] = true;
    System.out.print(curr+" ");

    for(int i=0; i<graph[curr].size(); i++) {
        Edge e = graph[curr].get(i);
        if(!vis[e.dest]) {
            dfs(graph, vis, e.dest);
        }
    }
}

public static void kosaraju(ArrayList<Edge> graph[], int V) {
    //Step1
    Stack<Integer> s = new Stack<>();
    boolean vis[] = new boolean[V];
    for(int i=0; i<V; i++) {
        if(!vis[i]) {
            topSort(graph, i, s, vis);
        }
    }

    //Step2
    ArrayList<Edge> transpose[] = new ArrayList[V];
    for(int i=0; i<V; i++) {
        transpose[i] = new ArrayList<Edge>();
    }

    for(int i=0; i<V; i++) {
        vis[i] = false;
```

```
            for(int j=0; j<graph[i].size(); j++) {
                Edge e = graph[i].get(j);
                transpose[e.dest].add(new Edge(e.dest, e.src));
            }
        }

        //Step3
        while(!s.isEmpty()) {
            int curr = s.pop();
            if(!vis[curr]) {
                System.out.print("SCC : ");
                dfs(transpose, vis, curr);
                System.out.println();
            }
        }
    }

    public static void main(String args[]) {
        int V = 5;

        ArrayList<Edge> graph[] = new ArrayList[V];
        createGraph(graph);

        kosaraju(graph, V);
    }
}
```

Part5

Bridge in Graph (Tarjan's Algorithm)

```
public static void dfs(ArrayList<Edge> graph[], int curr, int par, boolean vis[], int
dt[], int low[], int time) {
        vis[curr] = true;
        dt[curr] = low[curr] = ++time;

        for(int i=0; i<graph[curr].size(); i++) {
            Edge e = graph[curr].get(i);
            if(e.dest == par)
                continue;
```

```
        if(vis[e.dest]) {
            low[curr] = Math.min(low[curr], dt[e.dest]);
        } else {
            dfs(graph, e.dest, curr, vis, dt, low, time);
            low[curr] = Math.min(low[curr], low[e.dest]);
            if(dt[curr] < low[e.dest]) {
                System.out.println("BRIDGE : " + curr + "---" + e.dest);
            }
        }
    }
}

public static void getBridge(ArrayList<Edge> graph[], int V) {
    int dt[] = new int[V];
    int low[] = new int[V];
    int time = 0;
    boolean vis[] = new boolean[V];

    for(int i=0; i<V; i++) {
        if(!vis[i]) {
            dfs(graph, i, -1, vis, dt, low, time);
        }
    }
}
```

Articulation Point in Graph (Tarjan's Algorithm)

```
public static void dfs(ArrayList<Edge> graph[], int curr, int par,
                            boolean vis[], int dt[], int low[], int time,
                            boolean isArticulation[]) {
    vis[curr] = true;
    dt[curr] = low[curr] = ++time;
    int child = 0;

    for(int i=0; i<graph[curr].size(); i++) {
        Edge e = graph[curr].get(i);
        if(e.dest == par)
            continue;
        if(vis[e.dest]) {
            low[curr] = Math.min(low[curr], dt[e.dest]);
        } else {
            dfs(graph, e.dest, curr, vis, dt, low, time, isArticulation);
            low[curr] = Math.min(low[curr], low[e.dest]);
```

```java
            if(dt[curr] <= low[e.dest] && par != -1) {
                isArticulation[curr] = true;
            }
            child++;
        }
    }

    if(par == -1 && child > 1) {
        isArticulation[curr] = true;
    }

}

public static void getArticulation(ArrayList<Edge> graph[], int V) {
    int dt[] = new int[V];
    int low[] = new int[V];
    int time = 0;
    boolean vis[] = new boolean[V];
    boolean isArticulation[] = new boolean[V];

    for(int i=0; i<V; i++) {
        if(!vis[i]) {
            dfs(graph, i, -1, vis, dt, low, time, isArticulation);
        }
    }

    for(int i=0; i<V; i++) {
        if(isArticulation[i]) {
            System.out.println(i);
        }
    }
}
```

Graph Assignments

To do : after Part1

Qs - Rotten Oranges (Amazon/Adobe/Intuit/Uber)

You are given an m x n grid where each cell can have one of three values:

- 0 representing an empty cell,
- 1 representing a fresh orange, or
- 2 representing a rotten orange.

Every minute, any fresh orange that is 4-directionally adjacent to a rotten orange becomes rotten.

Return the minimum number of minutes that must elapse until no cell has a fresh orange. If this is impossible, return -1.

Example 1
Input: grid = [[2,1,1],[1,1,0],[0,1,1]]
Output: 4

Example 2
Input: grid = [[2,1,1],[0,1,1],[1,0,1]]
Output: -1
Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

Practice online : https://leetcode.com/problems/rotting-oranges/

Qs - Number of Islands (Google/Microsoft/Facebook/Apple)

Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return the number of islands.

An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

Example 1

Input: grid = [
 ["1","1","1","1","0"],
 ["1","1","0","1","0"],
 ["1","1","0","0","0"],
 ["0","0","0","0","0"]
]
Output: 1

Example 2

Input: grid = [
 ["1","1","0","0","0"],
 ["1","1","0","0","0"],
 ["0","0","1","0","0"],
 ["0","0","0","1","1"]
]
Output: 3

Practice online : https://leetcode.com/problems/number-of-islands/

To do : after Part2

Qs - Course Schedule (Facebook/Coinbase/Intuit)

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1. You are given an array prerequisites where prerequisites[i] = [ai, bi] indicates that you must take course bi first if you want to take course ai.

For example, the pair [0, 1], indicates that to take course 0 you have to first take course 1.
Return true if you can finish all courses. Otherwise, return false.

Example 1
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.

Example 2
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you should also have finished course 1. So it is impossible.

Practice online : https://leetcode.com/problems/course-schedule/

Qs - Find Eventual Safe States (Amazon/Adobe)

There is a directed graph of n nodes with each node labeled from 0 to n - 1. The graph is represented by a 0-indexed 2D integer array graph where graph[i] is an integer array of nodes adjacent to node i, meaning there is an edge from node i to each node in graph[i].

A node is a terminal node if there are no outgoing edges. A node is a safe node if every possible path starting from that node leads to a terminal node (or another safe node).

Return an array containing all the safe nodes of the graph. The answer should be sorted in ascending order.

Example 1
Input: graph = [[1,2],[2,3],[5],[0],[5],[],[]]
Output: [2,4,5,6]
Explanation: The given graph is shown above.
Nodes 5 and 6 are terminal nodes as there are no outgoing edges from either of them.
Every path starting at nodes 2, 4, 5, and 6 all lead to either node 5 or 6.

Example 2
Input: graph = [[1,2,3,4],[1,2],[3,4],[0,4],[]]
Output: [4]
Explanation:
Only node 4 is a terminal node, and every path starting at node 4 leads to node 4.

Practice online :
https://leetcode.com/problems/find-eventual-safe-states/description/

To do : after Part3

Qs - Cheapest Flights within K Stops (Amazon/TikTok/Airbnb)

There are n cities connected by some number of flights. You are given an array flights where flights[i] = [fromi, toi, pricei] indicates that there is a flight from city from-i to city to-i with cost price-i.

You are also given three integers src, dst, and k, return the cheapest price from src to dst with at most k stops. If there is no such route, return -1.

Example 1

Input: n = 4, flights = [[0,1,100],[1,2,100],[2,0,100],[1,3,600],[2,3,200]], src = 0, dst = 3, k = 1

Output: 700

Explanation:

The graph is shown above.

The optimal path with at most 1 stop from city 0 to 3 is marked in red and has cost 100 + 600 = 700.

Note that the path through cities [0,1,2,3] is cheaper but is invalid because it uses 2 stops.

Example 2

Input: n = 3, flights = [[0,1,100],[1,2,100],[0,2,500]], src = 0, dst = 2, k = 1

Output: 200

Explanation:

The graph is shown above.

The optimal path with at most 1 stop from city 0 to 2 is marked in red and has cost 100 + 100 = 200.

Practice online : https://leetcode.com/problems/cheapest-flights-within-k-stops/

To do : after Part4

Qs - Remove Max Number of Edges to Keep Graph Fully Traversable
(Microsoft/Google/Uber)

Alice and Bob have an undirected graph of n nodes and three types of edges:

Type 1: Can be traversed by Alice only.
Type 2: Can be traversed by Bob only.
Type 3: Can be traversed by both Alice and Bob.
Given an array edges where edges[i] = [typei, ui, vi] represents a bidirectional edge of type typei between nodes ui and vi, find the maximum number of edges you can remove so that after removing the edges, the graph can still be fully traversed by both Alice and Bob. The graph is fully traversed by Alice and Bob if starting from any node, they can reach all other nodes.

Return the maximum number of edges you can remove, or return -1 if Alice and Bob cannot fully traverse the graph.

Example 1
Input: n = 4, edges = [[3,1,2],[3,2,3],[1,1,3],[1,2,4],[1,1,2],[2,3,4]]
Output: 2
Explanation: If we remove the 2 edges [1,1,2] and [1,1,3]. The graph will still be fully traversable by Alice and Bob. Removing any additional edge will not make it so. So the maximum number of edges we can remove is 2.

Example 2
Input: n = 4, edges = [[3,1,2],[3,2,3],[1,1,4],[2,1,4]]
Output: 0
Explanation: Notice that removing any edge will not make the graph fully traversable by Alice and Bob.

Practice online :
https://leetcode.com/problems/remove-max-number-of-edges-to-keep-graph-fully-traversable/description/

To do : after Part5

Qs - Critical Connection in a Network (Facebook/Microsoft/Amazon)

There are n servers numbered from 0 to n - 1 connected by undirected server-to-server connections forming a network where connections[i] = [ai, bi] represents a connection between servers ai and bi. Any server can reach other servers directly or indirectly through the network.

A critical connection is a connection that, if removed, will make some servers unable to reach some other server.

Return all critical connections in the network in any order.

Example 1
Input: n = 4, connections = [[0,1],[1,2],[2,0],[1,3]]
Output: [[1,3]]
Explanation: [[3,1]] is also accepted.

Example 2
Input: n = 2, connections = [[0,1]]
Output: [[0,1]]

Practice online :
https://leetcode.com/problems/critical-connections-in-a-network/description/

X

NPTEL

reviewer4@nptel.iitm.ac.in ▾

Courses » Wild Life Ecology

Announcements Course Ask a Question Progress FAQ

Unit 2 - Week 1 - Introduction

Register for Certification exam

Course outline

How to access the portal

Week 1 - Introduction

- Lecture 01_Introduction to the course
- Lecture 02_A historical overview of Ecology
- Lecture 03_Ecology & Evolution
- Quiz : Assignment 1
- Assignment 1 Solution
- Wild Life Ecology : Feedback For Week 1

Week 2 - Ecological structure

Week 3_Ecological Interactions

Week

Assignment 1

The due date for submitting this assignment has passed.
As per our records you have not submitted this assignment.

Due on 2019-02-13, 23:59 IST.

1) In the Greek word root of Ecology, Oikos refers to *2 points*

- household
- preservation
- environment
- study

No, the answer is incorrect.
Score: 0

Accepted Answers:
household

2) Ecology is the scientific study of ___ that determine the distribution and abundance of organisms. (Fill in the blanks) *2 points*

- statics
- interactions
- dynamics
- habitat

No, the answer is incorrect.
Score: 0

Accepted Answers:
interactions

3) In the Greek word root of Ecology, logos refers to *2 points*

- household
- preservation
- environment

ce De

blanks)

- habitat
- niche
- environment
- population

No, the answer is incorrect.
Score: 0

Accepted Answers:
environment

5) "Enquiry into plants" is a book written by **2 points**

- Theophrastus
- Linnaeus
- Malthus
- Humboldt

No, the answer is incorrect.
Score: 0

Accepted Answers:
Theophrastus

6) Who amongst these is considered the father of Biogeography? **2 points**

- Theophrastus
- Linnaeus
- Malthus
- Humboldt

No, the answer is incorrect.
Score: 0

Accepted Answers:
Humboldt

7) Which of these is not a characteristic of fitness? **2 points**

- Fitness is environment-specific.
- Fitness is species-specific.
- Higher reproductive rate means higher fitness.
- Fitness should be measured across several generations.

No, the answer is incorrect.
Score: 0

Accepted Answers:
Higher reproductive rate means higher fitness.

8) Which of these is not a step in natural selection? **2 points**

- variation
- underpopulation
- struggle for existence
- survival of the fittest

No, the answer is incorrect.
Score: 0

Accepted Answers:
underpopulation

9) Which of these is not a characteristic of fitness? ***2 points***

- Fitness is environment-specific.
- Fitness is species-specific.
- Fitness works on traits such as size and speed.
- Fitness should be measured across several generations.

No, the answer is incorrect.
Score: 0

Accepted Answers:
Fitness works on traits such as size and speed.

10)Which of these is not a kind of selection ***2 points***

- directional
- stochastic
- disruptive
- stabilising

No, the answer is incorrect.
Score: 0

Accepted Answers:
stochastic

Previous Page End