```
4
5   public class Program {
6     // O(j + d) time | O(j + d) space
7     public static List<int> TopologicalSort(List<int> jobs, List<int[]> deps) {
8       JobGraph jobGraph = createJobGraph(jobs, deps);
9       return getOrderedJobs(jobGraph);
10    }
11
12    public static JobGraph createJobGraph(List<int> jobs, List<int[]> deps) {
13      JobGraph graph = new JobGraph(jobs);
14      foreach (int[] dep in deps) {
15        graph.addPrereq(dep[1], dep[0]);
16      }
17      return graph;
18    }
19
20    public static List<int> getOrderedJobs(JobGraph graph) {
21      List<int> orderedJobs = new List<int>();
22      List<JobNode> nodes = new List<JobNode>(graph.nodes);
23      while (nodes.Count > 0) {
24        JobNode node = nodes[nodes.Count - 1];
25        nodes.RemoveAt(nodes.Count - 1);
26        bool ContainsCycle = depthFirstTraverse(node, orderedJobs);
27        if (ContainsCycle) return new List<int>();
28      }
29      return orderedJobs;
30    }
31
32    public static bool depthFirstTraverse(JobNode node, List<int> orderedJobs) {
33      if (node.visited) return false;
34      if (node.visiting) return true;
35      node.visiting = true;
36      foreach (JobNode prereqNode in node.prereqs) {
37        bool ContainsCycle = depthFirstTraverse(prereqNode, orderedJobs);
38        if (ContainsCycle) return true;
39      }
40      node.visited = true;
41      node.visiting = false;
42      orderedJobs.Add(node.job);
43      return false;
44    }
45
46    public class JobGraph {
47      public List<JobNode> nodes;
48      public Dictionary<int, JobNode> graph;
49
50      public JobGraph(List<int> jobs) {
51        nodes = new List<JobNode>();
52        graph = new Dictionary<int, JobNode>();
53        foreach (int job in jobs) {
54          addNode(job);
55        }
56      }
57
58      public void addPrereq(int job, int prereq) {
59        JobNode jobNode = getNode(job);
60        JobNode prereqNode = getNode(prereq);
61        jobNode.prereqs.Add(prereqNode);
62      }
63
64      public void addNode(int job) {
65        graph.Add(job, new JobNode(job));
66        nodes.Add(graph[job]);
67      }
68
69      public JobNode getNode(int job) {
70        if (!graph.ContainsKey(job)) addNode(job);
71        return graph[job];
72      }
73    }
74
75    public class JobNode {
76      public int job;
77      public List<JobNode> prereqs;
78      public bool visited;
79      public bool visiting;
80
81      public JobNode(int job) {
82        this.job = job;
83        prereqs = new List<JobNode>();
84        visited = false;
85        visiting = false;
86      }
87    }
88  }
89
```