```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

type Dep struct {
  Prereq int
  Job    int
}

// O(j + d) time | O(j + d) space
func TopologicalSort(jobs []int, deps []Dep) []int {
  jobGraph := createJobGraph(jobs, deps)
  return getOrderedJobs(jobGraph)
}

func createJobGraph(jobs []int, deps []Dep) *JobGraph {
  graph := NewJobGraph(jobs)
  for _, dep := range deps {
    graph.AddPrereq(dep.Job, dep.Prereq)
  }
  return graph
}

func getOrderedJobs(graph *JobGraph) []int {
  orderedJobs := []int{}
  nodes := graph.Nodes
  for len(nodes) != 0 {
    node := nodes[len(nodes)-1]
    nodes = nodes[:len(nodes)-1]
    containsCycle := depthFirstTraverse(node, &orderedJobs)
    if containsCycle {
      return []int{}
    }
  }
  return orderedJobs
}

func depthFirstTraverse(node *JobNode, orderedJobs *[]int) bool {
  if node.Visited {
    return false
  } else if node.Visiting {
    return true
  }
  node.Visiting = true
  for _, prereqNode := range node.Prereqs {
    containsCycle := depthFirstTraverse(prereqNode, orderedJobs)
    if containsCycle {
      return true
    }
  }
  node.Visited = true
  node.Visiting = false
  *orderedJobs = append(*orderedJobs, node.Job)
  return false
}

type JobGraph struct {
  Nodes []*JobNode
  Graph map[int]*JobNode
}

func NewJobGraph(jobs []int) *JobGraph {
  g := &JobGraph{
    Graph: map[int]*JobNode{},
  }
  for _, job := range jobs {
    g.AddNode(job)
  }
  return g
}

func (g *JobGraph) AddPrereq(job, prereq int) {
  jobNode := g.GetNode(job)
  prereqNode := g.GetNode(prereq)
  jobNode.Prereqs = append(jobNode.Prereqs, prereqNode)
}

func (g *JobGraph) AddNode(job int) {
  g.Graph[job] = &JobNode{Job: job}
  g.Nodes = append(g.Nodes, g.Graph[job])
}

func (g *JobGraph) GetNode(job int) *JobNode {
  if _, found := g.Graph[job]; !found {
    g.AddNode(job)
  }
```