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

// O(j + d) time | O(j + d) space
function topologicalSort(jobs, deps) {
  const jobGraph = createJobGraph(jobs, deps);
  return getOrderedJobs(jobGraph);
}

function createJobGraph(jobs, deps) {
  const graph = new JobGraph(jobs);
  for (const [prereq, job] of deps) {
    graph.addPrereq(job, prereq);
  }
  return graph;
}

function getOrderedJobs(graph) {
  const orderedJobs = [];
  const {nodes} = graph;
  while (nodes.length) {
    const node = nodes.pop();
    const containsCycle = depthFirstTraverse(node, orderedJobs);
    if (containsCycle) return [];
  }
  return orderedJobs;
}

function depthFirstTraverse(node, orderedJobs) {
  if (node.visited) return false;
  if (node.visiting) return true;
  node.visiting = true;
  for (const prereqNode of node.prereqs) {
    const containsCycle = depthFirstTraverse(prereqNode, orderedJobs);
    if (containsCycle) return true;
  }
  node.visited = true;
  node.visiting = false;
  orderedJobs.push(node.job);
  return false;
}

class JobGraph {
  constructor(jobs) {
    this.nodes = [];
    this.graph = {};
    for (const job of jobs) {
      this.addNode(job);
    }
  }

  addPrereq(job, prereq) {
    const jobNode = this.getNode(job);
    const prereqNode = this.getNode(prereq);
    jobNode.prereqs.push(prereqNode);
  }

  addNode(job) {
    this.graph[job] = new JobNode(job);
    this.nodes.push(this.graph[job]);
  }

  getNode(job) {
    if (!(job in this.graph)) this.addNode(job);
    return this.graph[job];
  }
}

class JobNode {
  constructor(job) {
    this.job = job;
    this.prereqs = [];
    this.visited = false;
    this.visiting = false;
  }
}

exports.topologicalSort = topologicalSort;

```