```
  void addDep(int job, int dep);
  void addNode(int job);
  JobNode *getNode(int job);
};

JobGraph *createJobGraph(vector<int> jobs, vector<vector<int>> deps);
vector<int> getOrderedJobs(JobGraph *graph);
void removeDeps(JobNode *node, vector<JobNode *> *nodesWithNoPrereqs);

// O(j + d) time | O(j + d) space
vector<int> topologicalSort(vector<int> jobs, vector<vector<int>> deps) {
  JobGraph *jobGraph = createJobGraph(jobs, deps);
  return getOrderedJobs(jobGraph);
}

JobGraph *createJobGraph(vector<int> jobs, vector<vector<int>> deps) {
  JobGraph *graph = new JobGraph(jobs);
  for (vector<int> dep : deps) {
    graph->addDep(dep[0], dep[1]);
  }
  return graph;
}

vector<int> getOrderedJobs(JobGraph *graph) {
  vector<int> orderedJobs = {};
  vector<JobNode *> nodesWithNoPrereqs(graph->nodes.size());
  auto it = copy_if(graph->nodes.begin(), graph->nodes.end(),
                    nodesWithNoPrereqs.begin(),
                    [](JobNode *node) { return node->numOfPrereqs == 0; });
  nodesWithNoPrereqs.resize(distance(nodesWithNoPrereqs.begin(), it));
  while (nodesWithNoPrereqs.size()) {
    JobNode *node = nodesWithNoPrereqs.back();
    nodesWithNoPrereqs.pop_back();
    orderedJobs.push_back(node->job);
    removeDeps(node, &nodesWithNoPrereqs);
  }
  bool graphHasEdges = false;
  for (JobNode *node : graph->nodes) {
    if (node->numOfPrereqs) {
      graphHasEdges = true;
    }
  }
  return graphHasEdges ? vector<int>{} : orderedJobs;
}

void removeDeps(JobNode *node, vector<JobNode *> *nodesWithNoPrereqs) {
  while (node->deps.size()) {
    JobNode *dep = node->deps.back();
    node->deps.pop_back();
    dep->numOfPrereqs--;
    if (!dep->numOfPrereqs)
      nodesWithNoPrereqs->push_back(dep);
  }
}

JobGraph::JobGraph(vector<int> jobs) {
  nodes = {};
  for (int job : jobs) {
    addNode(job);
  }
}

void JobGraph::addDep(int job, int dep) {
  JobNode *jobNode = getNode(job);
  JobNode *depNode = getNode(dep);
  jobNode->deps.push_back(depNode);
  depNode->numOfPrereqs++;
}

void JobGraph::addNode(int job) {
  graph[job] = new JobNode(job);
  nodes.push_back(graph[job]);
}

JobNode *JobGraph::getNode(int job) {
  if (graph.find(job) == graph.end())
    addNode(job);
  return graph[job];
}

JobNode::JobNode(int job) {
  this->job = job;
  deps = {};
  numOfPrereqs = 0;
}
```