

PromptScratchpadOur Solution(s)Video Explanation

Run Code

Solution 1Solution 2

```
21
22     JobGraph(vector<int> jobs);
23     void addPrereq(int job, int prereq);
24     void addNode(int job);
25     JobNode *getNode(int job);
26 };
27
28 JobGraph *createJobGraph(vector<int> jobs, vector<vector<int>> deps);
29 vector<int> getOrderedJobs(JobGraph *graph);
30 bool depthFirstTraverse(JobNode *node, vector<int> *orderedJobs);
31
32 // O(j + d) time | O(j + d) space
33 vector<int> topologicalSort(vector<int> jobs, vector<vector<int>> deps) {
34     JobGraph *jobGraph = createJobGraph(jobs, deps);
35     return getOrderedJobs(jobGraph);
36 }
37
38 JobGraph *createJobGraph(vector<int> jobs, vector<vector<int>> deps) {
39     JobGraph *graph = new JobGraph(jobs);
40     for (vector<int> dep : deps) {
41         graph->addPrereq(dep[1], dep[0]);
42     }
43     return graph;
44 }
45
46 vector<int> getOrderedJobs(JobGraph *graph) {
47     vector<int> orderedJobs = {};
48     vector<JobNode *> nodes = graph->nodes;
49     while (nodes.size()) {
50         JobNode *node = nodes.back();
51         nodes.pop_back();
52         bool containsCycle = depthFirstTraverse(node, &orderedJobs);
53         if (containsCycle)
54             return {};
55     }
56     return orderedJobs;
57 }
58
59 bool depthFirstTraverse(JobNode *node, vector<int> *orderedJobs) {
60     if (node->visited)
61         return false;
62     if (node->visiting)
63         return true;
64     node->visiting = true;
65     for (JobNode *prereqNode : node->prereqs) {
66         bool containsCycle = depthFirstTraverse(prereqNode, orderedJobs);
67         if (containsCycle)
68             return true;
69     }
70     node->visited = true;
71     node->visiting = false;
72     orderedJobs->push_back(node->job);
73     return false;
74 }
75
76 JobGraph::JobGraph(vector<int> jobs) {
77     nodes = {};
78     for (int job : jobs) {
79         addNode(job);
80     }
81 }
82
83 void JobGraph::addPrereq(int job, int prereq) {
84     JobNode *jobNode = getNode(job);
85     JobNode *prereqNode = getNode(prereq);
86     jobNode->prereqs.push_back(prereqNode);
87 }
88
89 void JobGraph::addNode(int job) {
90     graph[job] = new JobNode(job);
91     nodes.push_back(graph[job]);
92 }
93
94 JobNode *JobGraph::getNode(int job) {
95     if (graph.find(job) == graph.end())
96         addNode(job);
97     return graph[job];
98 }
99
100 JobNode::JobNode(int job) {
101     this->job = job;
102     prereqs = {};
103     visited = false;
104     visiting = false;
105 }
106
```