Sublime

Monokai

00:00:00

105

106 107 } 108

 $deps = \{\};$ numOfPrereqs = 0;

```
Prompt
             Scratchpad
                              Our Solution(s)
                                                  Video Explanation
                Solution 2
  Solution 1
       void addDep(int job, int dep);
 24
      void addNode(int job);
       JobNode *getNode(int job);
26 };
 27
    JobGraph *createJobGraph(vector<int> jobs, vector<vector<int>> deps);
 29
    vector<int> getOrderedJobs(JobGraph *graph);
 30 void removeDeps(JobNode *node, vector<JobNode *> *nodesWithNoPrereqs);
 31
    // O(j + d) time | O(j + d) space
 32
 33
    vector<int> topologicalSort(vector<int> jobs, vector<vector<int>> deps) {
      JobGraph *jobGraph = createJobGraph(jobs, deps);
 34
 35
       return getOrderedJobs(jobGraph);
 36
 37
 38
     {\tt JobGraph} ~ *create {\tt JobGraph} (vector < int > jobs, vector < vector < int >> deps) ~ \{
 39
       JobGraph *graph = new JobGraph(jobs);
 40
       for (vector<int> dep : deps) {
 41
        graph->addDep(dep[0], dep[1]);
 42
 43
       return graph;
 44
45
 46
    vector<int> getOrderedJobs(JobGraph *graph) {
 47
       vector<int> orderedJobs = {};
 48
       vector<JobNode *> nodesWithNoPrereqs(graph->nodes.size());
 49
       auto it = copy_if(graph->nodes.begin(), graph->nodes.end(),
                        nodesWithNoPrereqs.begin(),
 50
 51
                        [](JobNode *node) { return node->numOfPrereqs == 0; });
       nodesWithNoPrereqs.resize(distance(nodesWithNoPrereqs.begin(), it));
       while (nodesWithNoPrereqs.size()) {
 53
 54
         JobNode *node = nodesWithNoPrereqs.back();
 55
         nodesWithNoPrereqs.pop_back();
 56
         orderedJobs.push_back(node->job);
 57
         removeDeps(node, &nodesWithNoPrereqs);
 58
 59
       bool graphHasEdges = false;
       for (JobNode *node : graph->nodes) {
61
        if (node->numOfPrereqs) {
 62
           graphHasEdges = true;
 63
 64
 65
       return graphHasEdges ? vector<int>{} : orderedJobs;
66
67
    void removeDeps(JobNode *node, vector<JobNode *> *nodesWithNoPrereqs) {
 68
 69
       while (node->deps.size()) {
         JobNode *dep = node->deps.back();
 70
 71
         node->deps.pop_back();
 72
         dep->numOfPrereqs--;
 73
         if (!dep->numOfPrereqs)
 74
           nodesWithNoPrereqs->push_back(dep);
 75
 76
 77
 78
    JobGraph::JobGraph(vector<int> jobs) {
 79
80
       for (int job : jobs) {
 81
         addNode(job);
 82
83
 84
 85 void JobGraph::addDep(int job, int dep) {
 86
       JobNode *jobNode = getNode(job);
       JobNode *depNode = getNode(dep);
 88
       jobNode->deps.push_back(depNode);
 89
       depNode->numOfPrereqs++;
 90
91
 92
    void JobGraph::addNode(int job) {
      graph[job] = new JobNode(job);
93
94
       nodes.push_back(graph[job]);
 95
96
97 JobNode *JobGraph::getNode(int job) {
       if (graph.find(job) == graph.end())
 98
        addNode(job);
99
      return graph[job];
100
101 }
102
    JobNode::JobNode(int job) {
103
104 this->job = job;
```

Run Code