AlgoExpert | Quad Layout | Swift | 12px | Sublime | Monokai | 00:00:00

Prompt | Scratchpad | Our Solution(s) | Video Explanation | Run Code

Solution 1 | Solution 2

```swift

        func addNode(job: Int) {
            let jobNode = JobNode(job: job)

            nodes.append(jobNode)
            graph[job] = jobNode
        }

        func addPrerequisiteToJob(job: Int, prerequisite: Int) {
            let jobNode = getNode(job: job)
            let prerequisiteNode = getNode(job: prerequisite)
            jobNode.prerequisites.append(prerequisiteNode)
        }

        func getNode(job: Int) -> JobNode {
            if let node = graph[job] {
                return node
            } else {
                graph[job] = JobNode(job: job)
                return graph[job]!
            }
        }
    }

    // O(j + d) time | O(j + d) space
    func topologicalSort(jobs: [Int], dependencies: [[Int]]) -> [Int] {
        let jobGraph = createJobGraph(jobs: jobs, dependencies: dependencies)
        return getOrderedJobs(jobGraph: jobGraph)
    }

    func createJobGraph(jobs: [Int], dependencies: [[Int]]) -> JobGraph {
        let jobGraph = JobGraph(jobs: jobs)

        for dependency in dependencies {
            let job = dependency[1]
            let prerequisite = dependency[0]
            jobGraph.addPrerequisiteToJob(job: job, prerequisite: prerequisite)
        }

        return jobGraph
    }

    func getOrderedJobs(jobGraph: JobGraph) -> [Int] {
        var orderedJobs = [Int]()
        var jobNodes = jobGraph.nodes

        while jobNodes.count > 0 {
            if let jobNode = jobNodes.popLast() {
                let containsCycle = depthFirstTraverse(jobNode: jobNode, orderedJobs: &orderedJobs)
                if containsCycle {
                    return []
                }
            }
        }

        return orderedJobs
    }

    func depthFirstTraverse(jobNode: JobNode, orderedJobs: inout [Int]) -> Bool {
        if jobNode.visited {
            return false
        }

        if jobNode.visiting {
            return true
        }

        jobNode.visiting = true

        for prerequisite in jobNode.prerequisites {
            let containsCycle = depthFirstTraverse(jobNode: prerequisite, orderedJobs: &orderedJobs)

            if containsCycle {
                return true
            }
        }

        jobNode.visited = true
        jobNode.visiting = false

        orderedJobs.append(jobNode.job)

        return false
    }
}

```