CS311 Homework 6

200 points

For this assignment, you will implement six graphical algorithms, three of which are related to topological sorting and three of which are related to maximum flow. The problems have been selected so as to create a significant opportunity for the reuse of code; your grade will in part be based on the degree to which you take advantage of this opportunity. A brief description of each problem follows; more detail can be found in Javadoc comments in the attached interface files.

I **Max-Flow Problems**

(1) **Maximum Flow**

Implement the Ford-Fulkerson algorithm discussed in lecture and the textbook. Your implementation must take $O(Ef)$ time in the worst case, where f is the magnitude of the flow in the computed maximum flow and E is the number of edges in the graph.

(2) **Maximum Flow with Vertex Capacities**

In this variation on the standard maximum flow problem, edges have unlimited capacity for flow but the maximum flow through each vertex is constrained. Your implementation must take $O((V + E)f)$ time in the worst case.

(3) **Maximum Cardinality Pairwise Vertex-Disjoint Paths**

Two paths from some vertex s to some vertex t are said to be *vertex-disjoint* if there exists no vertex (besides s and t themselves) that is in both paths. A set of paths is said to be *pairwise vertex-disjoint* if every pair of paths in the set is vertext-disjoint. Implement an algorithm that computes the size of the largest set of vertex-disjoint paths from a given start vertex s to a given end vertex t. Your implementation must take $O(V + E)$ time in the worst case.

II **Topological Sort Problems**

(1) **Depth First Search**

Implement a depth-first traversal algorithm that operates on arbitrary (directed) graphs. Your implementation should accept functionality in the form of three callback functions (processEdge, processDiscoveredVertex, and processExploredVertex) and should call each callback function with the appropriate parameters at the appropriate times. Your implementation should take $O(V + E)$ time in the worst case, disregarding the time consumed by the callback function calls.

(2) **Topological Sort**

Implement an algorithm that computes a valid topological ordering on the vertices in a given DAG. Your implementation should take $O(V + E)$ time in the worst case.

(3) **Parallel Precedence-Constrained Scheduling**

Implement an algorithm that computes the minimum amount of time required to complete a set of jobs by a computer with an infinite number of finitely quick parallel processors. Each job has a dependency on some (possibly empty) set of jobs whereby the job cannot begin until every job that it depends upon is complete - that is, some pairs of jobs have a *precedence constraint*. Your implementation should take $O(V + E)$ time in the worst case.

Submit Java implementations of the attached interfaces (IGraph, IMaxFlowAlgorithms, and ITopologicalSortAlgorithms) to Blackboard by the posted due date. Document each implementation with a brief description of its function and an analysis of its time complexity. Be sure to consider the time complexity of the various operations in your IGraph implementation.