Lab: Topological Sorting
CSC 207, “Algorithms and Object-Oriented Design”
Department of Computer Science
Grinnell College
May 14, 2021

Goal

The goal of this lab is to do tasks related to the topological sorting. Use the files that you already modified for Word Ladders and Dijkstra’s Algorithm labs. In Graph.java class, the acyclic method computes the shortest weighted paths using the topological ordering.

Tasks

1. The critical (longest) path between two vertices in a weighted directed acyclic graph is the path with the greatest sum of edge weights. In this graph, each vertex has a value equal to the sum of the edge weights of a path from a given start vertex to that vertex.

   We can find the critical path by considering the vertices one at a time in topological order. For each vertex consider all the edges that leave the vertex. For each of these edges, add the weight of the edge to the value of the source vertex on that edge. For each edge, compare the sum with the value of the destination vertex. Make the larger of these values the value of the destination vertex. After all vertices have been visited, the largest value stored in a vertex will be cost of the longest path to that vertex.

   a. Inside Graph.java, define a method, called criticalPath, to compute the longest paths from a start vertex to the other vertices on the acyclic graphs. Use the body of the acyclic method to implement criticalPath.

   b. Modify the processRequest method by asking users to enter a name for the algorithm that you implement for criticalPath.

   c. Test criticalPath by finding the longest paths for the following weighted directed graph. Assume that A is a start vertex. Confirm that the critical path from A to I is 19.

![Graph Diagram]
2. The topological sorting algorithm adds a vertex to a queue when there is not an incoming edge to the vertex. (That is, the vertex indegree is zero.) Now, suppose that the topological sorting adds a vertex to the queue when there is not an outgoing edge from that vertex.

Write a method that displays the topological ordering of vertices in a graph using outdegree instead of indegree of vertices. You may need two queues to implement this method. The first queue stores a vertex when its outdegree becomes zero. The second one stores a vertex when it is visited (removed from the first queue). Scanning the second queue provides the topological ordering of vertices.

Test your method for the graph shown above.