LAB 4 - Implementation of Shortest Path Algorithms

Submission Deadline: March 24, 11:59 pm
Assessment: 5% of the total course mark.

Description:

In this assignment you are required to write a `Java` implementation of weighted shortest path algorithms in directed graphs. For this, you will have to modify the `Java` class `Graph` that you wrote for Lab 3 to represent directed graphs. In addition, you have to remove from it the method `minSTPrim`, and add to it some new instance methods. You will also need to use the Java class `MinBinHeap` that you wrote for Lab 3, in your implementation of Dijkstra's algorithm. In addition, you have to write a new Java class `Queue`, that represents FIFO queues, to be used in the implementation of topological sort. Recall that the class `Graph` represents directed graphs using adjacency lists. Use the same classes `Vertex` and `Edge` that you used for Lab 3. All classes have to be declared in the same package.

You are not allowed to use any predefined `Java` methods other than for input and output.

Specification: The constructor for the class `Graph` has the same specification as in Lab 3, with the exception that now the graph is directed. This means that every time a new edge `(end1, end2)` is read from the input string, only one edge object is created and it is included in the adjacency list of vertex `end1`. The method `minST` has to be removed from the class since this method is specific to undirected graphs. The new methods to be included in the class `Graph` are listed below, with their specific requirements. You may declare additional private methods in the class.

◇ `public String dijkstraSP(int i, int j)` - computes the weighted shortest path from vertex v_i to vertex v_j (i.e., from vertex `v[i]` to vertex `v[j]`), if the graph does not contain negative-weight edges. The method returns a string specifying a shortest path and its weight or returns `null` if there is no path in the graph from v_i to v_j. You may assume that v_i and v_j are valid vertices in the graph. The format of the returned string is illustrated in the following example.
Assume that $i = 5$ and $j = 3$ and that the computed shortest path is v_5, v_2, v_1, v_3 with the weight 30. Then the returned string should be:
path: v5, v2, v1, v3, weight: 30

For this method you have to use Dijkstra's algorithm with the source $s = v_i$. Note that the algorithm computes the shortest path from s to any vertex. You may opt to stop early, once the desired path is computed, but this is not a requirement. Note that if no path between v_i and v_j exists, then the field `prev` of v_j will be `null`.
In this algorithm, for each vertex v, you need to keep track of the current path by keeping track of its weight and of the previous node on the path. You should use the variable `v.prev` to store the previous node and the variable `v.key` to store the

weight of the path. For the priority queue needed in the algorithm, you can use an object of the `MinBinHeap` class, in the same way it is used in Prim's algorithm for minimum spanning trees. The only difference between the implementations of Prim's algorithm and Dijkstra's algorithm is the meaning of `v.prev` and `v.key` and therefore the way they are updated. Of course, another difference is in how you compute the output string. For Dijkstra, you need first to recover the shortest path based on the values `v.prev`.

◇ `public String bellmanFordSP(int i, int j)` - computes the weighted shortest path from vertex v_i to vertex v_j (i.e., from vertex `v[i]` to vertex `v[j]`) if the graph does not have negative-weight cycles. The method returns a string specifying a shortest path and its weight if such a path exists, returns `null` if there is no path in the graph from v_i to v_j or returns the message "negative-weight cycle!" if the graph contains a negative-weight cycle. The format of the returned string when the shortest path exists is the same as for `dijkstraSP`.

◇ `public String dagSP(int i, int j)` - computes the weighted shortest path from vertex v_i to vertex v_j (i.e., from vertex `v[i]` to vertex `v[j]`) in a weighted directed acyclic graph in $O(|V|+|E|)$ running time. Two such algorithms were presented in Topic 4.6. You decide which one to use. The method returns a string specifying a shortest path and its weight if such a path exists or returns `null` if there is no path in the graph from v_i to v_j. The format of the returned string when the shortest path exists is the same as for `dijkstraSP`.

Note that this algorithm first sorts the vertices in topological order. For this you have to write the method specified next.

◇ `Vertex[] topologicalSort()` - sorts the vertices in topological order. It returns `null` if such an order does not exist, otherwise it returns an array storing the vertices in topological order. Note that this method should not modify the positions of the vertices in array `v` of the `Graph` object. In the implementation of this method you need to keep track of the indegree of each vertex v. You may use the field `v.key` to store the indegree, or you may allocate a local array to store the indegrees of all vertices. You will also need to use a queue. You have to write a Java class `Queue` for this purpose. The queue may store the `Vertex` objects or it can just store the integers representing the labels of the vertices.

◇ `public String shortestPath(int i, int j)` - computes the weighted shortest path from vertex v_i to vertex v_j (i.e., from vertex `v[i]` to vertex `v[j]`) by applying the most efficient algorithm for the given graph. Therefore, you have to first determine based on the type of the graph, which of the three algorithms is the most efficient, then apply it. The method returns a string obtained by concatenating two substrings. The first substring specifies the algorithm used. More specifically, it has to be one of: 1) "dijkstraSP, "; 2) "bellmanFordSP, "; 3) "dagSP, ". The second substring has the same specification as the string returned by the method `bellmanFordSP`.

Submission Instructions: Submit the source code for each of the `Java` classes `Graph`, `MinBinHeap` and `Queue`in a separate text file. Include the name of the class, your name and student number in the name of the file. For instance, if your name is "Ellen Davis" and the student number is 12345 then the two files should be named “GraphEllen-Davis12345.txt”, “MinBinHeapEllenDavis12345.txt” and “QueueEllenDavis12345.txt”