PROGRAM-5: adding features to a graph class.

DUE: Sunday, May 5 by 11:59PM

NO LATE SUBMISSIONS WILL BE ACCEPTED FOR THIS ASSIGNMENT.

DELIVERABLE: just modified source file Graph.h

---------------------------------------------------------------------------------------------------------------------

BACKGROUND: You are given a graph class (in the file Graph.h).

Many graph operations in the graph class express their

results via "labels" assigned to the vertices. There is a vertex\_label

struct specified in Graph.h which includes the various labels that can

be assigned to a vertex (e.g., distance, predecessors, etc.)

A particular algorithm like bfs may populate a vector of such labels. The

vector is indexed by vertex ID. We sometimes call such a vector a

"report." From such reports we can do things like extract paths, etc.

All of the operations except bfs (and dfs which you do not need to touch)

operate on DAGs.

Some Vocabulary:

In a DAG G:

inputs: subset of vertices with INDEGREE ZERO

outputs: subset of vertices with OUTDEGREE ZERO

input-path: a path in G STARTING AT AN INPUT VERTEX

(and ending at any vertex).

output-path: a path in G starting at any vertex and

ENDING AT AN OUTPUT VERTEX.

input-output-path (or io-path): a path STARTING AT AN INPUT

VERTEX \_AND\_ ENDING AT AN OUTPUT VERTEX.

---------------------------------------------------------------------------------------------------------------------

INSTRUCTIONS

In the source file src/Graph.h you will find a graph class with a bunch of

functions already implemented (building the graph, dfs,

bfs, topo-sort...) and also a number of "stubs" for functions which you

are to complete.

GETTING FAMILILAR WITH THE CODE: Browse the Graph.h file to gain an

understanding of how a graph instance is organized. Start with the

comments starting around line 40. Draw some diagrams so you understand

how a graph instance represents a particular graph.

WARMUP "LAB": follow the instructions in src/WARMUP-LAB.txt to help get

more familiar with some fundamental algorithms and interpreting their

results.

YOUR JOB FOR THIS ASSIGNMENT: Now that you are warmed up, your task for

this assignment is to implement each new feature/function marked with "TODO"

in Graph.h

Each TODO item has a banner comment explaining the requirements.

Below is a short summary of the TODO items (see Graph.h for

details).

TODO 1 (10 points): Modifying BFS so that the NUMBER OF SHORTEST PATHS to each

vertex is recorded. Recall that in BFS path length is simply

measured by number of edges.

TODO 2 (20 points): Path extraction (function: extract\_path).

When an algorithm like BFS or DFS (or the critical paths function

below) is run, the actual paths explored/discovered are encoded

using "predecessor" information. Given the predecessor info,

you will reconstruct paths (see comments for details).

Advice: you might attempt extract\_path first.

TODO 3 (30 points): implement function dag\_critical\_paths. This

function takes a DAG and labels each vertex with the length of the

LONGEST input path ("critical-paths" measured by sum of edge

weights) ending at that vertex. It also encodes the paths

themselves using the predecessor values (and the paths can be

extracted using TODO 2 above.)

TODO 4 (30 points): Implement function dag\_num\_paths. This function

labels each vertex with the number of input-to-output paths

which include that vertex. This function does not encode

any particular paths. It simply records the number of such paths.

See source code for details.

TODO 5 (30 points): Implement function valid\_topo\_order. This

function takes a vertex ordering and determines if it is

indeed a valid topological ordering of the given graph.

(If graph is not even a DAG, the function will return false).

See source code for details.

TODO 6 (30 points): Implement function enum\_paths. This function

takes a vertex and constructs ALL input-paths ending at that

vertex. The paths are represented as strings; a vector of

strings is populated with the paths.

The above items total 150 points. In addition, submissions receive up to

50 points for "honest effort" (i.e., there was clear effort put into the

assignment even if the result does not work completely).

Note: "honest effort" and "turned in \_something\_" are not the same thing!

Total Points Possible: 200

General suggestion: use programs bfs.cpp, dfs.cpp, topo.cpp and epaths.cpp as

models for additional programs to test your additional features.