CS 323 Homework 4

Due by Thursday, February 23, 2017 8:45 PM

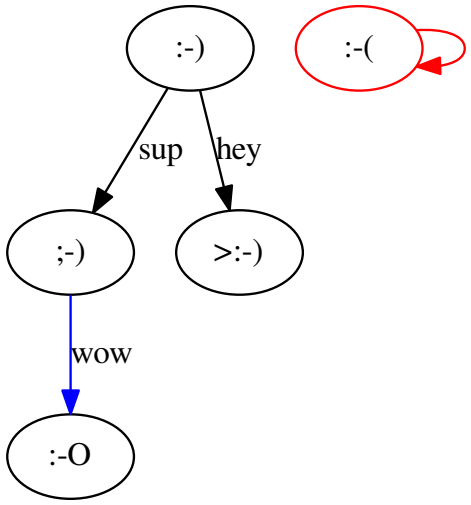
Submission instructions

Submit your assignment through the QTest system, using course ID: **CS323** and exam ID: **hw04**. Write all your code into the code boxes provided in QTest, and make sure that it works correctly by pressing the "Execute" button. If your program is composed of multiple classes, make the first one public and all the other ones not public (just omit the public visibility modifier for all classes except the first one). No email submissions are accepted. No late submissions are accepted. Include a collaboration statement in which you acknowledge any collaboration, help, or resource you used or consulted to complete this assignment. This section must be written even if you worked on the assignment alone.

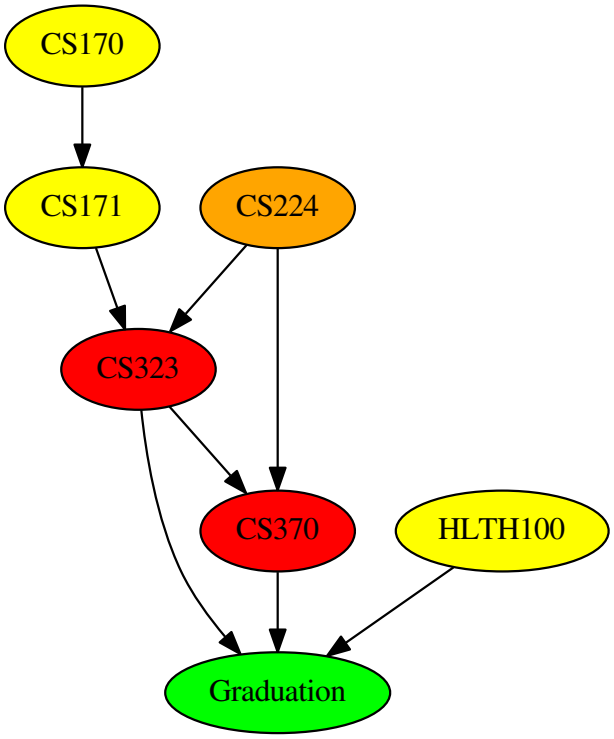
1 Graphviz

Graphviz (`http://www.graphviz.org`) is a powerful software for graph visualization. It is composed of a set of command line tools that take a plain text file with the description of a graph, and output the result in a variety of formats. Download the software from its website and familiarize with it. Then generate the following pictures, each of them in a separate PDF file. Include the three PDF files and the corresponding `.dot` source files in a ZIP archive named `graph-pics.zip`.

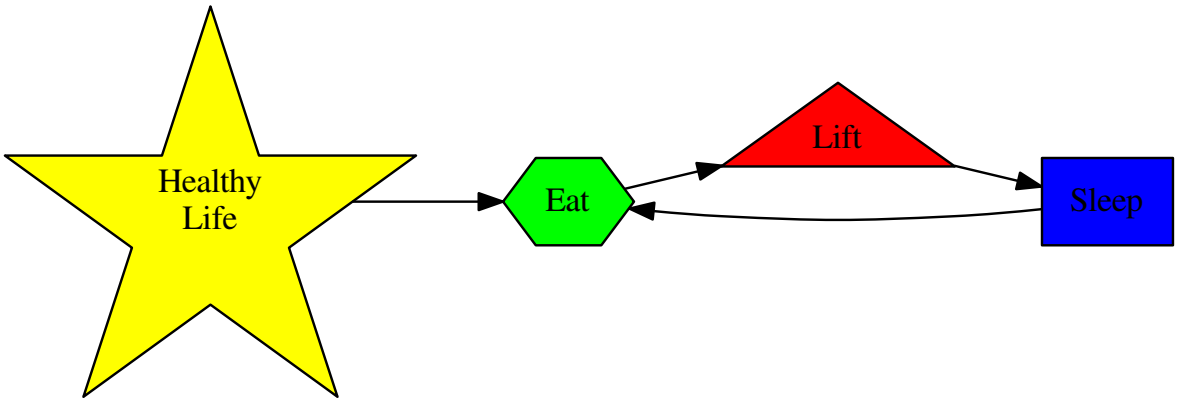
1.1 graph1.pdf (10 points)



1.2 graph2.pdf (10 points)



1.3 graph3.pdf (10 points)



2 Graph Algorithms

The included file `Graph.java` contains an implementation of a directed weighted graph datatype using an adjacency list approach.

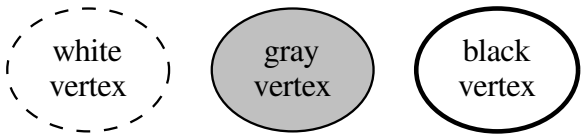
2.1 Comment the given code (10 points)

The given code does not contain any comment. Read it carefully, test it out, and understand all its design choices and how it works. Then write plenty of comments in the code to demonstrate that you thoroughly understood it.

2.2 Method toString (10 points)

Modify the `toString` method in the `Graph` class so that it returns a valid representation of the current state of the graph in the Graphviz DOT format. In this way you will be able to compile your graphs and see them through Graphviz. Represent the different colors of the vertices in the following way:

- White vertices have a dashed outline.
- Gray vertices have a normal outline and are filled in gray.
- Black vertices have a bold outline.



Do not modify the `toString` methods in the `Vertex` and `Edge` classes.

2.3 Method breadthFirstSearch (10 points)

Implement the `breadthFirstSearch` method using the algorithm in Chapter 22 of the textbook, with the following additions:

- For each main step of the algorithm (i.e., after the initialization, after a vertex turns gray, and after a vertex turns black) record a snapshot of your graph in a String (if you implemented the `toString` method correctly, the resulting String should be in DOT format). Store the sequence of snapshots in the attribute `searchSteps` (which is a list that should be emptied at the beginning of any new call of `breadthFirstSearch` or `depthFirstSearch`.
- Fill in also the attributes `discoverStep` and `finishStep` of each vertex, with the same meaning as they are filled by `depthFirstSearch`. Hint: the size of the `searchSteps` list can be used instead of the variable `time` in the textbook.
- The `distance` attribute of each vertex should be updated using the weight of the corresponding edges (i.e., the total distance of a path is the sum of the weights of the composing edges).

2.4 Method depthFirstSearch (10 points)

Implement the `breadthFirstSearch` method using the algorithm in Chapter 22 of the textbook, with the following additions:

- For each main step of the algorithm (i.e., after the initialization, after a vertex turns gray, and after a vertex turns black) record a snapshot of your graph in a String (if you implemented the `toString` method correctly, the resulting String should be in DOT format). Store the sequence of snapshots in the attribute `searchSteps` (which is a list that should be emptied at the beginning of any new call of `breadthFirstSearch` or `depthFirstSearch`.
- To fill in the attributes `discoverStep` and `finishStep` of each vertex, the size of the `searchSteps` list can be used instead of the variable `time` in the textbook.
- The `distance` attribute of each vertex should be updated using the weight of the corresponding edges (i.e., the total distance of a path is the sum of the weights of the composing edges).

2.5 Method path (5 points)

Implement the method `path` which returns a sequence of vertices connecting `startVertex` to `endVertex`, based on the information discovered by the last call of `breadthFirstSearch` or `depthFirstSearch`. This method will not call any search method. If neither `breadthFirstSearch` or `depthFirstSearch` was ever run before calling this method, this method should return an empty list.

2.6 Method pathWeight (5 points)

Implement the method `pathWeight` which returns the total weight of the path connecting `startVertex` to `endVertex`, based on the information discovered by the last call of `breadthFirstSearch` or `depthFirstSearch`. This method will not call any search method. If neither `breadthFirstSearch` or `depthFirstSearch` was ever run before calling this method, this method should return `Double.POSITIVE_INFINITY`.

2.7 Method topologicalSort (10 points)

Implement the method `topologicalSort` which returns a list of this graph's vertices in topologically sorted order.

2.8 Main method (10 points)

In the main method of your program, write enough test cases to thoroughly test all the methods you implemented. Using Graphviz and LaTeX, generate a PDF that shows a full run of (1) `breadthFirstSearch` and (2) `depthFirstSearch` on the graph of Figure 22.6 of the textbook.

Grading criteria

- Programs that do not compile in QTest get zero points.
- -10 points for missing collaboration statement.
- -10 points for missing comments or bad usage of comments.