10 Applications

Practice Problems: The Java Collections Framework

Practice problems help you get started, if some of the lab and lecture material is not clear. You are not required to do these problems, but make sure you understand how you would solve them. Solving them on paper is a great preparation for the exams.

Read through the documentation for Java Collections Framework library. Find how you can run use the stacks and queues defined there. Write a simple program that will test these algorithms and measure their timing in a manner similar to the previous lab.

10.1 Eliza

Finish all the Eliza part Lab 10 and hand in the completed work with your partner.

10.2 Graph Algorithms: BFS, DFS, Shortest Path

This problem is a continuation and refinement of the **Stacks and Queues** problem from Lab 10.

Graph

Your program needs to represent a graph with nodes that represent capitals of the 48 US states. Each node has a name — the name of the state. For each node, record the information about the capital of that state. Each edge represents a bi-directional connection between two adjacent states. You may consider the *four corner states: Colorado Utah, Arizona and New Mexico* as connected to each other. Each edge has a value that represents the distance between the capitals of the two states. The distances between two cities are based on the geographic distance. (See a separate announcement for a shortcut you can use to compute this distance.)

Algorithms

Your model should implement three graph traversal algorithms:

- Depth-First Search: uses a *Stack* to record the *ToDo* information
- Breadth-First Search: uses a *Queue* to record the *ToDo* information
- Shortest Path Search: uses a *Priority Queue* to record the *ToDo* information

To implement the shortest path you need to represent a priority queue.

The detailed description of the algorithm appears in a separate document. You will encounter a significant penalty for repeating the code - one algorithm implementation should run all three variants, distinguishing between them by selecting the appropriate implementation of a common interface for dealing with the *ToDo* information.

Using Libraries

Furthermore, throughout the project you are encouraged to leverage as much as possible from the existing Java libraries (both the Java Collections Framework, and the JPT libraries). The designer should focus on the design of interfaces between tasks, between components, wrapper and adopter class that allow you to use an existing library class in a customized setting.

User interactions

The file **GraphAlgoView.java** provides the code that creates a GUI allowing the user to select one of the three algorithms, the origin and the destination for the path.

Read the code, or at least the documentation and find the three places where you need to add the code that will invoke your implementation of the three algorithms.

You need to add code to your program that will show the user the path you have computed.

Before the user selects the algorithm to use and the origin and the destination for the path, she must be able to view a representation of the graph for which the computation is to be done.

This can be a graphical display, a text that lists the nodes and the edges (with their weights), or a graphical display of the text that lists the nodes and the edges.

Once the path has been computed, the user should be able to see the resulting path.

This may be a graphical display, or just a text listing the nodes along the path.

Here is a list of possible enhancements:

- Highlight the path is a different color in the graphics display.
- Display the steps in the search by highlighting in a different color the *visited* nodes, the *fringe* nodes (those currently in the queue or the stack), the *origin*, the *target*, and the *unseen* nodes. Animate the process using either the timer, or a user advance triggered by a key press.
- Animate the reconstruction of the path by traversing from the found target back to the previous node, all the way up to the origin.