Graphs

Andrew Rosen

Abstract

In this lab you get setup to work with graphs.

Create a Graph class to store nodes and edges or download a Graph library such as JUNG. Use it to implement Breadth First Search and Depth First Search. Follow the video from class if you need a reference. You can complete this in any language.

1 Using JUNG

Watch my videos on how to setup JUNG and read the provided pdf tutorial. One answer to a common question: there is no `Node` class or `Edge` class unless you write one; you can use any object as a vertex or edge. If you need weights, create an `Edge` class that will contain a weight. Again, see the tutorial.

This assignment is less about solving something tricky and more about being ready to solve the next problem.

1.1 Methods

If you're using JUNG, I recommend making your methods look something like this:

```
public static <V,E> void bfs(Graph<V,E> g, V start)
```

although you can return anything you want instead of a void. This way your algorithm will work on graphs that have any kind of vertices and edges.

2 Extra Credit

Add a method which performs Dijkstra's algorithm on a Graph from a given stating node. It should return a Map containing the shortest distances to each node.

You may need to make your own edge class as described in the tutorial to handle weights.

Here is how your method for the extra credit should look if you are working in Java. If you use this to do BFS or DFS, you'll be in for a bad time.

```
public static Map<V,Double> dijkstra(Graph<V,E> graph, V start){ }
```

Grading

25 points BFS

25 points DFS

10 points Extra Dijkstra's.