Graph Algorithms (Assignment 7)

Due: Friday, 6th of June

Summary

We add Dijkstra's Algorithm to calculate the shortest path between cities. Since this is our last assignment and we are leading up to the end of the quarter, we'll do a short assignment which requires us to code only ONE function in the Graph class (you shouldn't have to fiddle with the visualizer, but even if you have to, it'd be super minor)!

This is what you have to do:

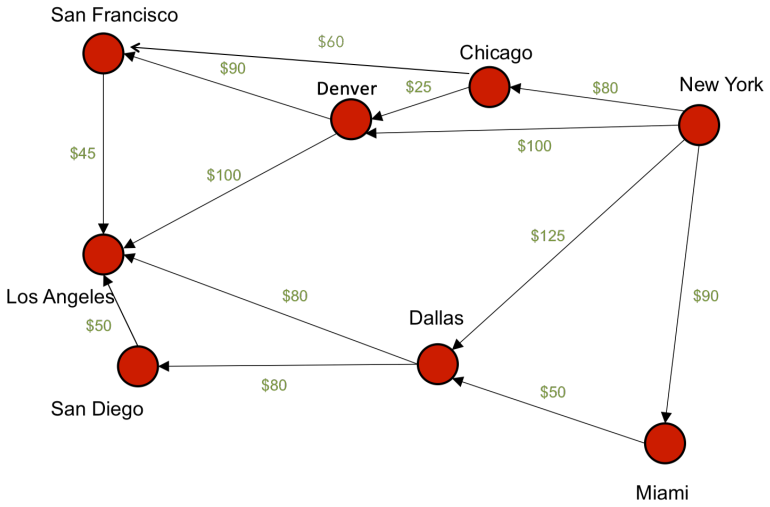
a. Write Dijkstra's Algorithm and hook it up to the front-end by replacing the earlier BFS that we used for path finding.

Here we'll see how using a priority queue (I've given you the priority queue class with all the necessary functions) enables Dijkstra's to solve the cheapest path finding issue with plain BFS.

By the end of this assignment, you would have written all the major kinds of graph algorithms (BFS and DFS from assignment 6 and route planning using Dijkstra's from this assignment). Bravo!

Details (PLEASE FOLLOW THIS CAREFULLY)

Step 1. Input graph



The input graph remains the same as last time. So use the same hardcoded values as assignment 6.

Step 2. Note that your project now has a priority queue class

I have given you the implementation of a minimum heap priority queue (partially sourced from here) and added the ExtractMin() and DecreaseKey() functions. The functions that you would need to use from this class are:

a. Insert
b. ExtractMin
c. DecreaseKey

All of these have been tested and are working, so this should make your life easier. If you are curious, feel free to dig under the surface in this priorityqueue class to see how it works.

Step 3. Change the GraphNode class a little bit

In your ***GraphNode*** class, add the following changes:

a. A field called cost (we saw this being used in the EL stop example in class). For example (I have added the field and a property):

```
private int cost;

public int Cost
{
    get { return cost; }
    set { cost = value; }
}
```

b. Change the GraphNode class definition to

```
public class GraphNode:IComparable<GraphNode>
```

IComparable is a C# interface that we use to compare the values/sizes of any two object. If you look up C# documentation, you'll see that this interface asks you to implement a CompareTo() function. So add that to the GraphNode as well (shown on next page).

c. Add this function to the GraphNode class as well (note, we use the cost field here, so ensure that it defined)

```
public int CompareTo(GraphNode other)
     {
         if (this.cost < other.cost) return -1;
         else if (this.cost > other.cost) return 1;
         else return 0;
            }
```

NOTE: in the DIjkstra Algorithm slides (slide#10), I mentioned that adding a variable called index to the GraphNode is a good idea, but I have already fixed that for you in the priority queue class. You **DON'T** have to include this index field in the GraphNode class.

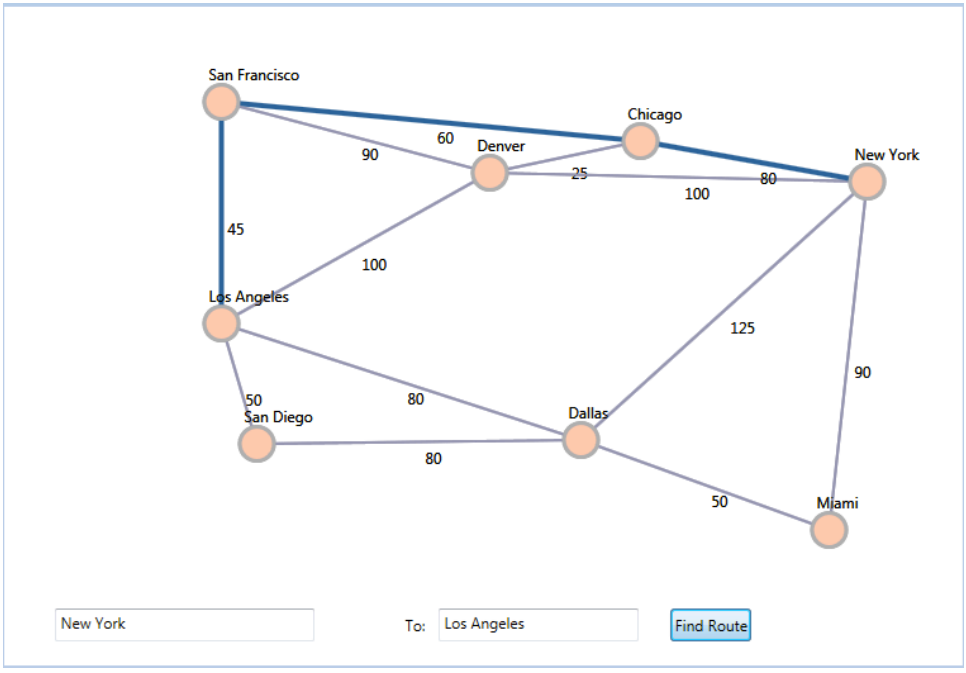
Step 4. Write Dijkstra's Algorithm ☺

Follow the class slides, it'll be of help. The algorithm should look like

```
public List<GraphNode> DijkstraShortestPath(GraphNode start, GraphNode finish){

       List<GraphNode> returnList = new List<GraphNode>();
       PriorityQueue<GraphNode> PQ = new PriorityQueue<GraphNode>();

       //Define rest of the stuff as it seems fit to you

}
```

Step 5. Ensure things still work with the Visualizer

The test case is going from New York to Los Angeles with minimal path length (cost = 185, should go from NY,CHI,SF,LA)



NOTE: If you compare this to the output in assignment 6, you'd notice how this is different (earlier, the route should have been different)

Turning the assignment

c. In Visual Studio, chose "Clean Solution" from the Build menu. This will remove all the binary files from the solution, leaving only the source code. That will make your assignment smaller and easier to upload.

c. Close Visual Studio

c. Right click on the folder icon for your assignment and choose the "Compressed (zipped) folder" selection from the Send To menu. o Note:Many of you are using commercial compression software such as WinRar. You may use these programs to make a zip file, if you prefer. However, you must specifically turn you program in as a ZIP file. Do not turn in other file formats such as RAR, TAR, TGZ, etc.

c. Return to this assignment on blackboard and upload your zip file as your submission.