Homework 4: Dijkstra's Algorithm

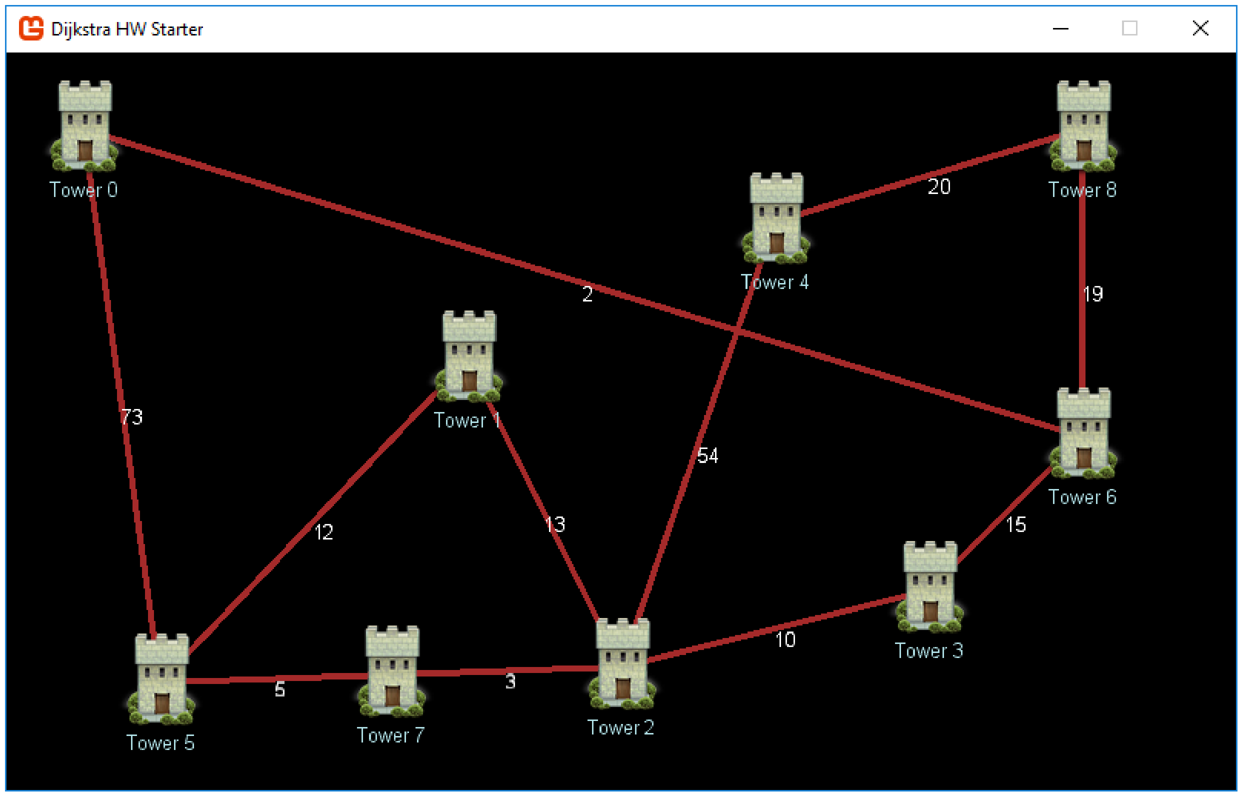
Due Date: Sunday, April 26th by 11:59PM

Goal

Gain experience working with graphs by implementing Dijkstra's algorithm for shortest paths.

Overview

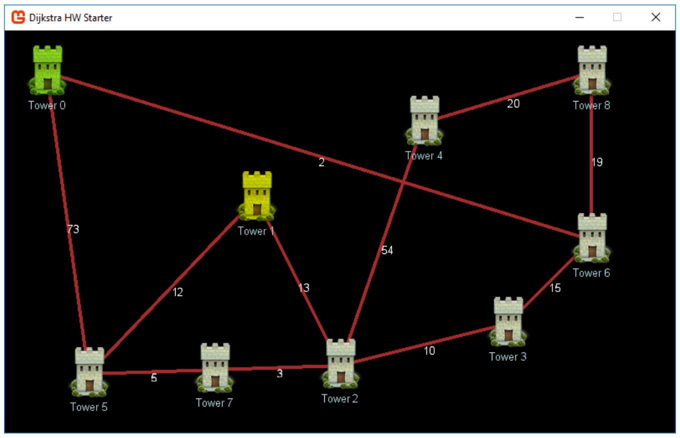
Download the included starter project .zip file from MyCourses, which contains a project called "Dijkstra HW Starter." Running the solution shows a window that looks like the screenshot below.



This program contains a vertex class and a graph class. The graph class can already represent a weighted graph: a connected set of vertices where each edge has an arbitrary cost associated with it.

In the example screenshot, you can see that each vertex is represented by a small tower image. Each edge in the graph automatically has a line drawn to represent the connection, as well as a white number representing the weight (or *cost*) of that edge. (Note that these numbers were *arbitrarily chosen* and do not reflect physical distances between towers. It may help to think about them as toll road costs.)

Interactivity



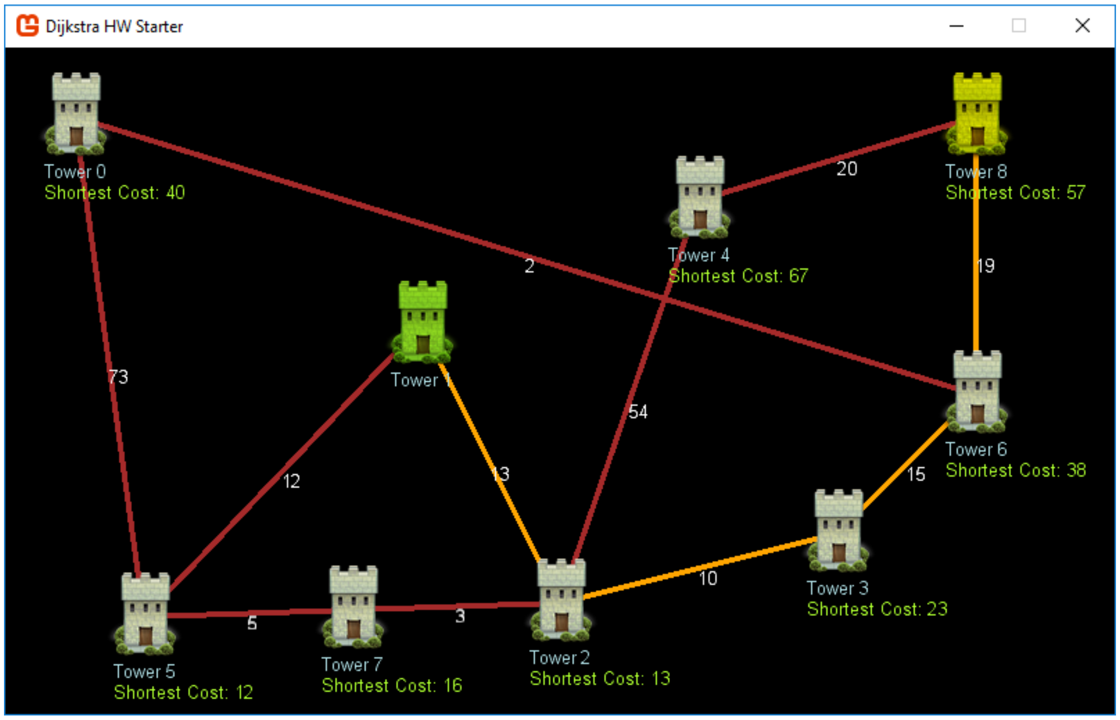
The given graph class also responds to mouse input. Hovering over a vertex (one of the towers) highlights it yellow temporarily. Clicking on a vertex sets it as the *starting tower* for the algorithm, highlighting it green. See the screenshot to the right for an example.

This won't have a noticeable effect quite yet, but once you've completed the homework, all towers will display their distance from the starting tower. In addition, when you're done, selecting a tower then hovering over a different tower will highlight the shortest path between the two.

Note that all of this interactivity and highlighting **is done for you**. You simply need to correctly implement the algorithm, updating vertices appropriately, and the highlighting & distances will appear.

Completed Example

If you look in the bin/Windows/x86/Debug folder, there's a version called "Dijkstra COMPLETE.exe". This program has the full algorithm implemented, such that selecting a tower will display all of the shortest distances (costs) to all other towers. Hovering over a second tower will also highlight the shortest path between them.



Dijkstra's Algorithm

Since the graph class is already complete, your job is to implement Dijkstra's shortest path algorithm. When the user clicks on one of the towers, the shortest path **to all the other locations** must be calculated.

You'll do this by completing in the `FindShortestPaths()` method in the Student Methods section of Graph.cs. This method is automatically called whenever a tower is clicked. This is the **only** method you need to complete for this assignment, although feel free to add private helper methods to make your life easier. When the user clicks on a tower, the vertex that tower represents is automatically passed to this method. Use that vertex as the source vertex for the algorithm.

Refer to the associated lecture for the steps of the algorithm itself. Here are some pointers for how to use the code provided:

- The provided Vertex class already contains all of the fields & properties you'll need.
- Recall that for Dijkstra's algorithm, you will need to reset *each* vertex before running the algorithm. For instance: to start, all "source distances" should be int.MaxValue. Luckily the Vertex class has a Reset() method you can use for this.
    - However, the starting vertex's distance should be set to zero!
- The Graph class contains both a *dictionary* and a *list* that you may find useful. The list contains all of the vertices. The dictionary maps strings to ints, so that given a name, you can get an index, which you can plug into the list to get the vertex object itself.
    - The dictionary is called "vertNameToIndex" and the list is simply called "vertices".
    - An example of this expression would be: `vertices[vertNameToIndex["Tower 0"]]`
- The Graph class also contains an adjacency matrix called "adjMatrix", which is a 2D array of ints.
- The starter code handles selecting different towers. Once your homework is complete, selecting different towers will re-run the algorithm, updating the shortest path information stored in each vertex. It should also then display the shortest cost under each tower.
- Once your homework is complete, hovering over towers after selecting one should automatically highlight the shortest path between the two.

You may add extra towers (vertices) and paths (edges) if you'd like, or even adjust the ones that are there, though it is not necessary. If you do, make sure you have at least as many as the starter code provides.