HW3 Graphs Writeup:

Running Code Instructions:

To run the program, change your terminal directory to data, which contains the files needed in the assignment.

My personal example:

PS C:\Users\rylee\CS315\data>

Then run the code under that directory.

If you run everything except for anything involving the priority queue or djikstra's, bfs will work.

Implement Priority Queue using Binary Heap:

This algorithm took a while for me to figure out, since I did not load the files in and understand how everything fits together before working on it. I created all heap functions needed to implement djikstra's, but just could not get them to work in time. I believe my heap implementation is fairly correct: see code.

Breadth First Search:

To implement this search, I loaded the vertex and edge files required for the assignment into their own lists— Cities, edges-- then I found the length of each dataset. For BFS to function properly, I needed to create a graph from the dataset. First, I wrote a graph class by creating a dictionary, then wrote an add_edge function that attached the vertexes from the data to one another. Next, I wrote the definition for Breadth First Search. This function has three parameters, the graph—A, the source node—s, and the destination—d. The function includes the initialization of a visited array, which then appends the source vertex to the queue. After, this, the source is popped from the queue. It then traverses the graph to search each connected vertex until the destination node has been visited. It then prints the path, which I stored in its own list. Instead of creating a print path function, I just included it at the end of the BFS function.

My driver function for the search initialized a graph variable with the Graph class. It then traversed the list to find the source node value, the destination node value, and the weight—did not use weights for BFS. It then created an edge through add_edge. It creates a new edge with every iteration until it reaches the end of edges. This results in the graph to be traversed by BFS, which I called outside of the loop for each source and destination pair.

Shortest path from Arad to Sibiu: Arad→Sibiu

Shortest path from Arad to Craiova: Arad →Sibiu→ RimnicuVilcea →Craiova

Shortest path from Arad to Bucharest: Arad →Sibiu→ RimnicuVilcea→Pitesti→Bucharest

Dijkstra's Algorithm:

Unfortunately, I was unable to make djikstra's work before the deadline, so I have to turn it in as incomplete. I know it is not working because of my graph class not having the right parameters to match up with those in djikstra's, but when I tried to fix it, it just messed up my breadth first search.

If I was able to, I would have used Djikstra's to look at the source node and destination node, as well as the distance in between them. In searching for the destination, I would have added up the weights in the paths to the destination and compared them with one another. Whatever ended up being the shortest distance would have been the shortest path.

I'm assuming the shortest path between arad and Bucharest would not have been the same as in BFS.

Prim's Algorithm:

Not sure if I should write stuff for this part, but I will if I end up having the time