H3.

- The assignment is due at Gradescope on Saturday, November 5 at 11:59pm. Late assignments will not be accepted. Submit early and often.
- You are permitted to study with friends and discuss the problems; however, *you must write up you own solutions, in your own words*. Do not submit anything you cannot explain. If you do collaborate with any of the other students on any problem, please do list all your collaborators in your submission for each problem.
- Finding solutions to homework problems on the web, or by asking students not enrolled in the class is strictly prohibited.
- We require that all homework submissions are prepared in Latex. If you need to draw any diagrams, however, you may draw them with your hand. Please use *a new page to begin each answer.*

PROBLEM 1 *Programming Prim*

In this problem, you will program Prim's MST algorithm in function `calculateTree` in the file `Prim.java` that accompanies this homework. The source file describes the input and the expected output. You should use the Heap datastructure that we provide for you in the file `Heap.java` and you can consult `HeapTest.java` to see how to use the class.

PROBLEM 2 *Programming Dijkstra*

In this problem, you will program Dijkstra's algorithm in the function `calculatePaths` in the file `Dijkstra.java` that accompanies this assignment.

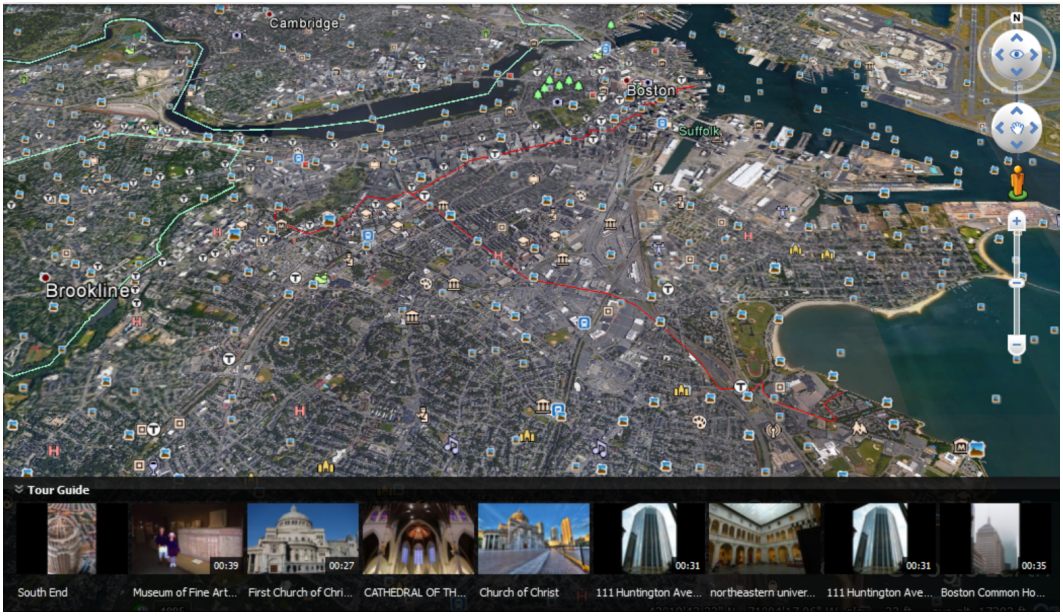
PROBLEM 3 *Steiner tree approximation*

The Steiner tree problem is very closely related to the minimum spanning tree, but it has one extra *flexibility* which makes the problem much harder to solve. In the Steiner tree problem, you are given a graph $G = (V, E)$, and you wish to find a tree of minimum weight that connects the nodes in some subset $R \subset V$ (i.e. not all the nodes in V have to be connected). This problem can be used to place fire stations on intersections in a city so as to minimize the travel time to any particular house, or, in our case, it will help the City of Boston design a new bus system that helps connect all of *your* favorite landmarks in Boston.

In this problem, we are going to use the functions `calculateTree` and `calculatePaths` to find an *approximate* Steiner tree for a graph G. The algorithm works as follows:

1. For each node $x \in R$, compute the shortest paths $\delta_{x,y}$ for all the other nodes $y \in R$ using `calculatePaths`.
2. Create a new graph $G_R = (R, E_R)$ where the edges E_R are the complete set of edges between all nodes. Define new weights $w_R(x, y)$ for the edge between $x, y \in R$ to be $\delta(x, y)$ that was computed in the previous step.
3. Now compute a minimum spanning tree T_R on G_R using `calculateTrees`.
4. Now solve the original problem by extending T_R into a tree for G as follows: for each edge $(x, y) \in T_R$, replace the edge with the shortest path from x to y computed in the first step.

Complete the function `main` in the file `Steiner.java` with the algorithm above. We have included a `graph.txt` file which contains all of the streets in Boston, and methods to read that graph, and to read a file of special locations from `locations.txt`. Our stub program also provides a method to write your tree to a KML file which can be viewed using either Google Earth or Google Maps (use google to discover how).
The image below shows a sample tree rendered in Google Earth based on the locations file.



PROBLEM 4 *Another all pairs shortest path*

Set $\text{BSHORT}_{i,j,k}$ to be the shortest path from i to j that uses at most k edges. Note this is different from the ASHORT variable that we used in class in that we do not restrict the intermediate nodes to be $1 \ldots k$ in this formulation. State a recursive formula for BSHORT. Devise an algorithm that uses this recurrence. What is the running time of this algorithm?

PROBLEM 5 *Negative Cycles*

In lecture, we argued that the Bellman-Ford algorithm can be made to detect a negative cycle by running for one more iteration and seeing if any of the verticies have a shorter path. Prove this by showing that if graph $G = (V, E, w)$ has a negative cycle, then there exists a vertex $v \in V$ for which $\text{SHORT}_{V-1,v} > \text{SHORT}_{V,v}$.