DIT725 V16, Assignment 3: Speeding up Heaps and Graphs

This assignment is divided into two tasks.The first task is relatively small compared to the second. See the course homepage for deadlines and general instructions on group assignments. You can also find the required Java files there.

1 Binary Heap

In heap.java, there is a working but inefficient implementation of a heap. Your assignment is to modify it so both $deleteMin$ and add are $O(logn)$, and $findMin$ is $O(1)$.

Submit: Your modified Heap.java. Also write a HeapReport.pdf that analyses the complexity of the methods you have written (explain how you know they are $O(logN)$).

2 Dijkstras tram finder

In TramFinder.java and TramNetwork.java, there is a working implementation of a simple travel planner for tram networks. The problem is that it is dreadfully slow ($O(2^N)$). Your assignment is to make a faster version using Dijkstras shortest path algorithm. For the system to work, it needs a correct SortedListImpl.java from Assignment 2. It is used to quickly find the next tram departure as well as finding stations from incomplete names.

Gothenburg.java uses the travel planner to find tram routes in a small part of the Gothenburg tram network. RandomNetwork.java can be used to build much larger random tram networks.

First study the supplied files until you know a bit about how they work. Especially look at TramNetwork.java and how it is used in TramFinder.java. Your assignment is to implement the method $fastFindRoute$ in TramFinder.java. It should work as $findRoute$, but with better complexity. Note, you do not need to find identical routes as in the slow version, but if another route is found it must still arrive at the same time. Use this to test the correctness of your implementation.

Submit: Any Java files that you have modified/added. Report.pdf containing:

- An explanation of your solution.
- A complexity analysis of your solution in terms of the number of stations N, the number of tramlines E and the highest number of daily departure times for any tram T (and any other parameters you find necessary).

Hint: Use the Heap from task 1 to implement the algorithm.

Hint: You may want to extend the TramArrival with some additional fields.

Hint: Your solution may not look much like the slow version. For instance you may want to avoid using recursion which means you probably don't need the Finder-class.

Challenge: How big tram networks can your solution handle? RandomNetwork can be set to generate some really large tram network. Can your solution reliably deal with thousands of stops?

Challenge: The route finder is a bit silly sometimes, for instance it switches tram line and then switches back. Can this be avoided or mitigated (still giving as fast routes)?