Matthew Feng October 5, 2018

6.046 Problem Set 4

Collaborators: *Eric Qian, Tyler Barr, Alex Guo*

Problem 1

(a)

Since we must first prioritize Jane's commute, we first compute the shortest path from Jane's office o to her house h. We can do this $O(E + V \log V)$ time using Dijkstra's algorithm with Fibonacci heaps. After we have found this path, we create a supernode z, and connect z to every node in that path with an edge of weight 0, and then delete the edges in the original graph. We can then run Prim's algorithm starting with node z to find the "minimum spanning tree" that also includes the shortest route from o to h (in reality, we are finding the minimum spanning trees for each of nodes in the shortest path and then connecting them via the edges in the shortest path). Finally, we remove the 0 weight edges and add back the original shortest path edges and return that set of edges. The algorithm will find the minimum cost edges to form the minimum spanning tree because after we remove the shortest path edges from the graph and then run the MST algorithm, we will find a true MST for the graph. By readding the shortest path edges, we ensure that Jane has her shortest path, and the rest of the MST has minimum cost paths given the requirement that Jane's shortest path must be in the graph. Prim's algorithm also runs in $O(E + V \log V)$ time with a Fibonacci heap, so the overall algorithm will run in $O(E + V \log V)$ time.

(b)

Problem 2

(a)

We construct the digraph $G = (V, E)$ with capacity function $c(u, v) : E \to \mathbb{R}$ denoting the maximum flow that can pass through edge (u, v) at any moment in time.

This is a variation on the bipartite matching problem. As such, we have m vertices $r_j \in V, \forall j \in [1, m]$ that will represent riders waiting to be matched, and n vertices $k_i \in V, \forall i \in [1, i]$

that represent cars that can hold riders. We have a source s and sink t as well, which will allow us to think of the flow in this network intuitively as riders getting (matched) into cars.

We then will construct m edges (s, r_j), $\forall j \in [1, m]$, where $c(s, r_j) = 1$, as each rider only needs one car.

We will also construct n edges (k_i, t), $\forall i \in [1, n]$, where $c(k_i, t) = s_i$, the seat capacity of each car. By setting the capacities in this way, we can sure that each car is matched with at most its seat capacity.

Finally, for each rider j, we construct edge (r_j, k), $\forall k \in C_j$, each with infinite capacity.

(b)

(c)

(d)