Introduction to Algorithms: 6.006
Massachusetts Institute of Technology
Erik Demaine, Ronald L. Rivest, and Nir Shavit

October 22, 2013
Handout # 1

Problem Set 4, Problem 1

Kevin Peng
Collaborators: Genghis Chau

This problem set is due **Tuesday, November 5** at **11:59PM**.

This solution template should be turned in through our submission site. [1]

For written questions, full credit will be given only to correct solutions that are described clearly *and concisely*.

---

[1] Register an account, if you haven't done so. Then go to Homework, Problem Set 4, and upload your files.

Problem 4-1. [25 points] **Use Your Tires**

Ben Bitdiddle is trying to travel from Cambridge to Pasadena, CA to attend a cousin's wedding. He is given an undirected graph representing a road map to guide his way. Each edge in this graph represents a road segment, and each vertex represents an intersection. For this problem, all road edges have lengths 1, 2, or 3. He wants to minimize the total distance traveled.

But there's a catch. Not all roads are the best to travel on. There are ***bad*** roads that are so rough that Ben is guaranteed to lose exactly one of his tires whenever he travels on a bad road. He can't fix tires en route so once a tire is gone, it is gone. Ben can keep traveling as long as he has at least one good tire (his car is one he designed and built himself as part of his Mech. Engr. double major).

How does Ben find the minimum-length path from Cambridge to Pasadena that loses at most 3 tires? Be sure to describe your algorithm as well as analyze its runtime. Your algorithm should return the path that Ben would take.

We can take all road edges with lengths 2 and 3 and split them into smaller roads of length 1. If these roads are "bad" roads, we can label the new roads with badness of $\frac{1}{n}$ where n is the length of the original road edges. For example, if nodes A and B are connected by a road of length 3, we would remove the edge connecting A and B and create new edges from A to N_1, N_1 to N_2, and N_2 to B, where each of these edges have a length of 1 and a badness index of $\frac{1}{3}$. This will yield at most $3E$ edges, where E is the number of edges in the original graph and $V + 2E$ vertices. This effectively removes the weights from the problem since all edges now have a length of 1, so finding the path with the minimum length is equivalent to finding the path with the least number of edges. Something to note here is that our modification to the graph will not produce an incorrect path. If we look at our example, any path that originally would have gone from A to B will still have to go from A to N_1 to N_2 to B.

Then, we will run BFS from the starting node until we reach the goal. Here, we will use a queue-based implementation, in which the elements are the paths from the start node to some node in the graph. Every time we consider adding a new node, we will keep track of the length of the path from the start node to this current node (which we can do in constant time since this is the same as keeping track of the level in a regular BFS) and the total number of bad roads that the path contains (this can also be done in constant time per iteration). If the total number of bad roads is greater than 3 (it could be a fraction greater than 3 due to our modification), we discard that path instead of adding it to our queue. Otherwise, if the node is the goal node, we can immediately return the path to that node as the minimum-length path and revert the any edge modifications that we made in our initial setup.

For our time analysis, we note that we can remove an edge and add an edge or vertex in constant time. Thus, our initial modification of the graph will take $O(E)$ time. We know that the running time of BFS is $O(\text{number of vertices } + \text{number of edges})$. For the modified graph, this will take $O(V+5E)$ time. Lastly, undoing the changes we made on the minimum-length path will take $O(E)$ time as well. Thus, the total running time of this algorithm is $O(V + E)$.