

Problem Set 6

This problem set is due **December 6 at 11:59PM**.

Solutions should be turned in through the course website in PS or PDF form using \LaTeX or scanned handwritten solutions, or they may be handwritten and turned in to a member of the 6.006 course staff on or before the due date. Hand-drawn diagrams may also be referenced in your \LaTeX writeup and turned in at the next day's recitation.

A template for writing up solutions in \LaTeX is available on the course website.

Exercises are for extra practice and should not be turned in.

Exercises:

- Exercise 24.1-1 from CLRS.
 - Exercise 24.3-2 from CLRS.
 - Exercise 24.3-4 from CLRS.
 - Exercise 24.5-8 from CLRS.
 - Exercise 24.3-6 from CLRS.
-

1. (18 points) Shortest Paths

Decide whether these statements are **True** or **False**. You must briefly justify all your answers to receive full credit.

- (a) (5 points) If some edge weights are negative, the shortest paths from s can be obtained by adding a constant C to every edge weight, large enough to make all edge weights nonnegative, and running Dijkstra's algorithm.
- (b) (5 points) Given a graph G with nonnegative edge weights and two nodes s and t , there exists a polynomial-time algorithm to either find the longest path from s to t or detect that a cycle on the path is reachable, by negating all the edge weights.
- (c) (4 points) Let $\delta(x, y)$ be the shortest path distance from a vertex x to another vertex y . If $\delta(s, t) = \delta(s, u) + \delta(u, t)$, then u is on a shortest path from s to t .
- (d) (4 points) Let P be a shortest path from some vertex s to some other vertex t . If the weight of each edge in the graph is squared, P remains a shortest path from s to t .

2. **(12 points)** Critical Edges in Shortest Paths

Let graph $G = (V, E)$. Let $w : E \rightarrow \mathfrak{R}$ be a cost function. Assume $w(e) \geq 0$ for all $e \in E$. Let s be a source node. We say that an edge e is *upwards critical* if by increasing $w(e)$ by any $\epsilon > 0$ we increase the distance from s to some vertex $v \in V$. We say that an edge e is *downwards critical* if by decreasing $w(e)$ by any $\epsilon > 0$ (but still having $w(e) \geq 0$, i.e., by definition if $w(e) = 0$ then e is not downwards critical) we decrease the distance from s to some vertex $v \in V$. Give an $O(|E| \log |V|)$ -time algorithm to compute the upwards and downwards critical edges of G .

3. **(30 points)** U.S. Highways

The Howe & Ser Moving Company is transporting the Caltech Cannon from Caltech's campus to MIT's and wants to do so most efficiently. Fortunately, you have at your disposal the National Highway Planning Network (NHPN), a 1:100,000 scale network database that contains line features representing just over 450,000 miles of current and planned highways in the US. The NHPN consists of interstates, principal arterials, and rural minor arterials. (Source: <http://www.fhwa.dot.gov/planning/nhpn/>)

This problem set includes node and link databases from the NHPN, available as files on the 6.006 website. Open `nhpn.nod` and `nhpn.lnk` in a text editor to get a sense of how the data is stored (`datadict.txt` has a more precise description of the data fields and their meanings). To save you the trouble of parsing these structures from a file, we have provided you with a Python module `nhpn.py` containing code to load the database into memory. Read the comments there to make sure you understand how to use the `nhpn.Loader` interface.

Additionally, we have provided some tools to help you visualize the output from your algorithms. You can use the `Visualizer` class to produce a KML (Google Earth) file. To view such a file on Google Maps, place it in a web-accessible location, such as your Athena `Public` directory, and then search for its URL on Google Maps.

For this problem, you only need to turn in code. Use the code template we have provided, which loads the NHPN module with `import nhpn`, to submit your answers.

- (a) **(3 points)** Write a short procedure `node_by_name(nodes, city, state)` to find a city's location in the set of nodes, given its name and state. Return a `Node` object corresponding to the city. Check that it works for a few examples. Note that some nodes have a description which isn't solely the city name, e.g. `CAMBRIDGE NW` or `NORTH CAMBRIDGE`. Given a choice of more than one latitude and longitude, choose the first that appear in the data.

Note that this procedure might be useful in debugging your code later on.

- (b) **(3 points)** Since we are working with a geographical map, where edge weights represent street lengths, we can use a shortest path algorithm that assumes non-negative edge weights. Throughout this problem set, ignore the curvature of the Earth, i.e., assume that the distance between points (x, y) and (x', y') is given by $\sqrt{(x - x')^2 + (y - y')^2}$.

Write a function `lat_long_length(node1, node2)` to return the length of an edge between two NHPN nodes.

Hint: You may find the `math.hypot` function useful.

- (c) **(24 points)** Implement Dijkstra's algorithm for the shortest path between two vertices in a graph with non-negative edge weights, using the `Node` and `Link` data types. Your procedure `dijkstra_search(nodes, edges, w, s, t)` should take as arguments a graph G (represented as nodes and links), a function $w : E(G) \rightarrow \mathbb{R}_+$ mapping edges (represented as vertex pairs) to their weights, and a source vertex s and a destination vertex t represented as `Node` objects.

Your procedure should return a sequence (list) of `Nodes` representing a path from the source to the destination.

Hint: What kind of graph representation (e.g. adjacency list, adjacency matrix) seems most appropriate for this graph? Construct an appropriate representation from the nodes and edges. Don't forget that edges are undirected.