HW 2 (due Monday, at noon, February 9, 2015)

OLD CS 473: Fundamental Algorithms, Spring 2015 Version: **1.2**

You also have to do quiz online (on moodle).

Collaboration Policy: For this homework, Problems 1–3 can be worked in groups of up to three students.

1. (40 PTS.) Know your neighbors.

You are given an undirected graph $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ with n vertices and m edges. The graph is already stored in memory using adjacency list representation, with positive weights on the edges, where $\mathsf{V} = \{1, \ldots, n\}$. Let $d_{\mathsf{G}}(u, v)$ denote the length of the shortest path in G between u and v, for any $u, v \in \mathsf{V}$.

Let $C_t(u)$ be the set of t vertices in G closest to u in G according to $d_{\mathsf{G}}(u, \cdot)$, and let $\Gamma(C_t(u))$ be the set of all edges in G that have an endpoint in $C_t(u)$.

(A) (10 PTS.) Describe how to modify Dijkstra's algorithm, such that computing the set of vertices $C_t(u)$ takes $O\Big(t \log n + \Big|\Gamma\big(C_t(u)\big)\Big|\Big)$ time. Note, that this is not trivial, as initializing the standard Dijkstra algorithm takes $\Theta(n)$ time. You are allowed to use hashing, and assume that every basic hashing operation takes $O(1)$ time.

(B) (10 PTS.) Describe a graph, with $O(n)$ edges, and n vertices, such that computing $C_t(u)$, for all $u \in \mathsf{V}(\mathsf{G})$, for some $t = O(1)$, takes $\Omega(n^2)$ time, if using the algorithm from (A). Prove your answer.

(C) (20 PTS.) (Harder.) Describe in detail, and prove correctness and running time, of an algorithm that computes $C_t(u)$, for all the vertices $u \in \mathsf{V}$. The running time of your algorithm has to be $O(t(n \log n + m))$. In particular, your algorithm should run in $O(n \log n)$ time for the instance of the problem specified in (B). (Hint: Think about the naive algorithm, which executes (A) from all vertices, and think how to rearrange the execution of this algorithm, so that it avoids unnecessary work.)

2. (30 PTS.) Heavy neighbors.

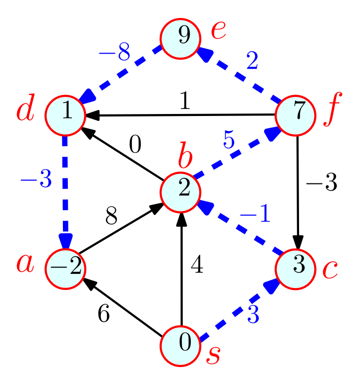
Let G be a directed graph with n vertices and m edges (which might have cycles). Every vertex is assigned a positive weight. Describe an algorithm that computes the vertex in G, such that the total weight of its reachable set (i.e., the vertices in G it can reach) is maximized. Your algorithm should be as fast as possible.

3. (30 PTS.) Anti-diet and its effect on shortest path trees.

Let $\mathsf{G} = (\mathsf{V}, \mathsf{E})$ be a directed graph with edge lengths that can be negative. Let $\ell(e)$ denote the length of edge $e \in \mathsf{E}$ and assume it is an integer. Assume you have a shortest path tree T rooted at a source node s that contains all the nodes in V. You also have the distance values $d_{\mathsf{G}}(s, u)$ for each $u \in \mathsf{V}$ in an array (thus, you can access the distance from s to u in $O(1)$ time). Note that the existence of T implies that G does not have a negative length cycle.

(A) Let $e = (p, q)$ be an edge of G that is *not* in T. Show how to compute in $O(1)$ time the smallest integer amount by which we can decrease $\ell(e)$ before T is not a valid shortest path tree in G.

(B) Let $e = (p, q)$ be an edge in the tree T. Show how to compute in $O(m + n)$ time the smallest integer amount by which we can increase $\ell(e)$ such that T is no longer a valid shortest path tree. Your algorithm should output ∞ if no amount of increase will change the shortest path tree.



The example above may help you. The dotted edges form the shortest path tree T and the distances to the nodes from s are shown inside the circles. For the first part consider an edge such as (b, d) and for the second part consider an edge such as (f, e).