Problem Set 6

This problem set is due **at 11:59pm** on **Thursday, April 2, 2015.**

Each submitted solution should start with your name, the course number, the problem number, your recitation section, the date, and the names of any students with whom you collaborated.

Exercise 6-1. Read CLRS, Chapter 25.

Exercise 6-2. Exercise 25.3-2.

Exercise 6-3. Exercise 25.3-5.

Exercise 6-4. Read CLRS, Chapter 23.

Exercise 6-5. Exercise 23.2-4.

Exercise 6-6. Exercise 23.2-5.

Problem 6-1. Dynamic and Bounded-Hop All-Pairs Shortest Paths [25 points]

This problem explores some extensions of the All-Pairs Shortest Paths (APSP) algorithms covered in lecture and in CLRS. All parts of this problem assume nonnegative real weights. We assume that the vertices are numbered $1, 2, \ldots, n$.

Weighted directed graphs may be used to model communication networks, and shortest distances (shortest-path weights) between nodes may be used to suggest routes for messages. However, most communication networks are dynamic, i.e., the weights of edges may change over time. So it is useful to have a way of modifying distance estimates to reflect these changes.

(a) [6 points] Give an efficient algorithm that, given a weighted directed graph $G = (V, E, W)$, a correct distance matrix D for G, a corresponding predecessor matrix Π, and a triple (i, j, r), where $i, j \in V$ and r is a nonnegative real, modifies D and Π to reflect the effects of changing $w_{i,j}$ to r.

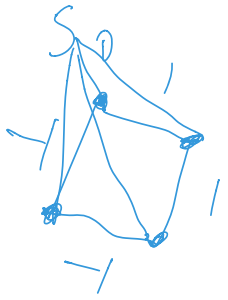
Your algorithm will need to handle three cases: $r = w_{i,j}$, $r < w_{i,j}$, and $r > w_{i,j}$. For each of the cases, analyze your algorithm. (Note: Your worst case running time for one of these cases may not be better than $O(V^3)$.)

(b) [4 points] Give an example of a weighted directed graph $G = (V, E, W)$, a distance matrix D, and a triple (i, j, r) such that any algorithm that modifies D to reflect the effects of changing $w_{i,j}$ to r must take $\Omega(V^2)$ time.

25.3-2
What is the purpose of adding the new vertex s to V, yielding V'?

we want run Dijstra's alg, need no negtive weight, so we add s, to V, compute $h(v) = \delta(s,v)$. then the new weigh. of $w_h(u,u) = w(u,u) + h(u) - h(v) > 0$, we run the new weighted G'. and ensure no negtive cycle, because no edge to s.

25.3-5
Suppose that we run Johnson's algorithm on a directed graph G with weight function w. Show that if G contains a 0-weight cycle c, then $\hat{w}(u,v) = 0$ for every edge (u,v) in c.



cycle c : $v_0, v_1, \cdots v_k, v_0$.

for any $i, j \in (0,k]$ in cycle, $i \neq j$.

$\hat{w}(i,j) = w(i,j) + h(i) - h(j) > 0$.

$h(j) - h(i) < w(i,j)$, hj 可以 relax.

与 hji 是 δ 矛盾. 所以 ≥ 0, 不 > 0. $= 0$.

6-1. (a).

1). $r = w_{ij}$. nothing change.

2) $r < w_{ij}$ $d_{x,y} = \min(d_{x,i} + r + d_{j,y}, d_{x,y})$.

$\Theta(V^2)$, all pair of i, j.

3 $r > w_{ij}$ must to recompute whole D and Π using Floyd-Warshall Alg. in $\Theta(V^3)$

---

c). $d_{ij}^{k,h} = \min\left(d_{ij}^{k-1,h}, \min\left(d_{ij}^{k-1,h'}, d_{ij}^{k-1,h-h'}\right)\right)$ $k \in [0,h]$

$\Theta(V^3 \cdot h)$.

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-design-and-analysis-of-algorithms-spring-2015/assignments/MIT6_046JS15_pset6sols.pdf

Now suppose that we add a new constraint to the problem — an upper bound of h on the number of hops (edges) in the paths we want to consider. More formally, given a weighted digraph $G = (V, E, W)$ and a positive integer h, $0 \leq h \leq n-1$, we would like to produce a distance matrix D giving the shortest distances for at-most-h-hop paths.

(c) [5 points] Adapt the Floyd-Warshall algorithm to solve the bounded-hop APSP problem. Analyze its complexity in terms of graph parameters and h.

(d) [5 points] Adapt the matrix multiplication strategy from Section 25.1 to solve the bounded-hop APSP problem. Analyze its complexity in terms of graph parameters and h. Try to get logarithmic dependence on h.

(e) [5 points] Finally, consider a dynamic version of the bounded-hop APSP problem. Design an algorithm that is given the following as input:

1. a weighted directed graph $G = (V, E, W)$;
2. a hop count h, $0 \leq h \leq n-1$;
3. a correct distance matrix D yielding shortest at-most-h-hop distances, and possibly additional distance information that is useful for solving this problem; and
4. a triple (i, j, r), where $i, j \in V$ and r is a nonnegative real.

Your algorithm should modify D to reflect the effects of changing $w_{i,j}$ to r, and should also update any additional distance information that you have added. As for Part (a), your algorithm will need to handle three cases: $r = w_{i,j}$, $r < w_{i,j}$, and $r > w_{i,j}$. For each of the cases, analyze your algorithm's complexity, in terms of graph parameters and h.

Problem 6-2. Minimum Spanning Trees with Unique Edge Weights [25 points]

Consider an undirected graph $G = (V, E)$ with a weight function w providing nonnegative real-valued weights, such that the weights of all the edges are different.

(a) [5 points] Prove that, under the given uniqueness assumption, G has a unique Minimum Spanning Tree. use kruskal (or Prim alg), every time have only one option, and the order is determined,

Each of the next three parts outlines an MST algorithm for graphs with unique edge weights. In each case, say whether this is a correct MST algorithm or not. If so, give a proof, a more detailed description of an efficient algorithm, and an analysis. If not, give a specific counterexample. (We are omitting the point values for these parts because we will assign more points to algorithms and fewer to counterexamples.)

(b) [Batched Edge-Addition MST]

The algorithm maintains a set A of edges that are known to be in the MST. Initially, A is empty. The algorithm operates in phases; in each, it adds a batch of one or more edges to A. Phases continue until we have a spanning tree.

right.

Specifically, in each phase, the algorithm does the following: For each component tree C in the forest formed by A, identify the lightest weight edge e_C crossing the cut between C and the rest of the components. After determining these edges for all component trees, add all of the edges e_C to A, in one batch.

(c) [Divide-and-Conquer MST]

The algorithm uses a simple *Divide-and-Conquer* strategy: Divide the set V of vertices arbitrarily into disjoint sets V_1 and V_2, each of size roughly $V/2$. Define graph $G_1 = (V_1, E_1)$, where E_1 is the subset of E for which both endpoints are in V_1. Define $G_2 = (V_2, E_2)$ analogously.

Recursively find (unique) MSTs for both G_1 and G_2; call them T_1 and T_2. Then find the (unique) lightest edge that crosses the cut between the two sets of vertices V_1 and V_2, and add that to form the final spanning tree T.

(d) [Cycle-Breaking MST]

The algorithm operates in phases. In each phase, the algorithm first finds some nonempty subset of the simple cycles in the graph. Then it identifies the heaviest edge on each cycle, and removes all these heavy edges. Phases continue until we have a spanning tree.

23.2-4

Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Kruskal's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?

23.2-5

Suppose that all edge weights in a graph are integers in the range from 1 to $|V|$. How fast can you make Prim's algorithm run? What if the edge weights are integers in the range from 1 to W for some constant W?