Homework #6
Algorithms I
600.463
Spring 2017

Due on: Thursday, April 6th, 11:59pm
Late submissions: will NOT be accepted
Format: Please start each problem on a new page.
Where to submit: On Gradescope under HW6
Please type your answers; handwritten assignments will not be accepted.
To get full credit, your answers must be explained clearly,
with enough details and rigorous proofs.

April 17, 2017

Problem 1 (20 points)

Suppose you are given an undirected graph G with weighted edges and a minimum spanning tree T of G.

- Design an algorithm to update the minimum spanning tree when the weight of a single edge e is *increased*.
- Design an algorithm to update the minimum spanning tree when the weight of a single edge e is *decreased*.

In both cases, the input to your algorithm is the edge e and its new weight; your algorithms should modify T so that it is still a minimum spanning tree. Analyze the running time of your algorithms and prove the correctness.
Answer:

- Consider the case that edge e is *increased*: if e is not an edge in T, no need to change anything, since e cannot be a part of the MST by increasing its weight. If add e to T, the new T creates a subgraph H with one cycle. Then

e is the longest edge in that cycle even before we increase its weight. Thus increasing the weifht of e only makes it more useless.
If e is an edge of T, we can first delete e from T, which creates an intermediate spanning forest F with two components. Let e_{min} be the minimum-weight edge with one endpoint in each component of F. So the new MST is $F \cup e_{min}$. (It is possible that e and e_{min} are the same edge.)
Then we need a method to find this e_{min}. We first mark all the vertices in one component of F, by using BFS/DFS in $O(V + E)$ time. Then we go through every edge of G and find the minimum-weight edge with exactly one marked endpoint in $O(E)$ time.

- Consider the case that edge e is *decreased*: if e is an edge in T, no need to change anything, since e still belongs to the MST after we decrease its weight. If deleting e from T, there obtains an intermediate spanning forest F. Then e is a safe edge with respect to F even before we decrease its weight. Thus decreasing the weight of e only makes it safer.
  If e is not an edge of T, the graph $U = T \cup \{e\}$ must contain a cycle, since there is a unique path in T between the endpoints of e. Let e_{max} be the maximum-weight edge in this cycle. The new MST is $T' = U\ \{e_{max}\}$. (e and e_{max} could be the same edge.)
  We can use BFS/DFS to find e_{max} in U in $O(V + E)$ time.

Problem 2 (20 points)

Problem 2.1 (10 points)

Let $G = (V, E)$ be a directed, weighted graph with weight function $w : E \rightarrow \mathbb{R}$. Give an $O(VE)$-time algorithm to find, for each vertex $v \in V$, the value $\delta^*(v) = min_{u \in V}\{\delta(u, v)\}$, where $\delta(u, v)$ is the *shortest-path weight* from u to v defined in the textbook.
Answer:
You can do this by making a new dynamic-programming algorithm that probably looks a bit like Bellman-Ford, with the updates going in the "opposite" direction. Let's consider a solution without a new algorithm at all. Construct a graph $G' = (V', E')$, where $V' = V \cup \{s\}$ and $E' = E \cup \{(s, u)|u \in V\}$, i.e. we have a new vertex s and there is an edge from the new vertex to every vertex in t he graph. Set the weight of these new edges to be $w(s, u) = 0$. Then, just run Bellman-Ford algorithm on G' starting from s.

Claim 0.1. $\delta^*_G(v) = \delta_{G'}(s, v)$

Proof. If we want to prove this , we should show that $\delta^*_G(v) \leq \delta_{G'}(s,v)$ and $\delta^*_G(v) \geq \delta_{G'}(s,v)$. To show the former, suppose $s \to u \leadsto v$ is a shortest path from s to v in G', then $w(s \to u \leadsto v) = w(u \leadsto v)$, and $u \leadsto v$ is a shortest path from u to v in G. Thus, we have $\delta^*_G(v) \leq \delta_G(u,v) = w(s \to u \leadsto v) = \delta_{G'}(s,v)$. To argue the other direction, suppose that $u = argmin_{u \in V}\{\delta_G(u,v)\}$, i.e., that $\delta_G(u,v) = \delta^*_G(v)$. Then a valid path from s to v in G' is $s \to u \leadsto v$ with weight $\delta^*_G(v)$, and we conclude that $\delta_{G'}(s,v) \leq \delta_G(u,v) = \delta^*_G(v)$. $\square$

Problem 2.2 (10 points)

Let $G = (V, E)$ be a directed, weighted graph with nonnegative weight function $w : E \to \{1, 2, \ldots, W\}$ for some nonnegative integer W. Devise an algorithm to compute the shortest path distances from a given source vertex s in $O(WV + E)$ time.

Answer:

Let's consider two types of solutions:

- **Solution 1:** Run Dijkstra's algorithm but with a priority queue that is geared specifically towards this problem. In particular, at any time, only W different distances are in the queue, with at most WV distance in total. You could thus implement the priority queue as follows: create an array of size WV containing all possible distances. Keep a pointer to the current min distance. Each array entry contains (a pointer to) an unordered list of elements having that distance. To $EXTRACT\text{-}MIN$, remove any element from the current list in $O(1)$ time. If that list is empty, advance the pointer until finding a nonempty list. There are $O(WV)$ pointer advances in total (amortized analysis here). To $DECREASE\text{-}KEY$, simply move the element from one list to another in $O(1)$ time, splicing it out of one and inserting it at the front of the other. So the total cost is $O(WV + V + E)$ for V $EXTRACT\text{-}MINs$ and E $DECREASE\text{-}KEYs$. Note that this is not a general-purpose priority queue — the decrease key is only allowed to decrease as low as the current value of the pointer, which is sufficient for Dijkstra's algorithm.

- **Solution 2:** Reduce a graph with weights in $\{1, 2, \ldots, W\}$ to a graph with edges of weights all 1 with the same shortest path distances. BFS finds shortest paths in unit-weighted graphs. So you can apply BFS here at that point. Specifically, the new graph is as follows. For each u, create W copies $u_1, u_2, \ldots, u_W$. That is

$$V' = \{u_i | u \in V, 1 \leq i \leq W\}. \quad (1)$$

As for the edges, string together all u_i in a chain, i.e., $u_1 \to u_2 \to \cdots \to u_W$. Now the idea is t hat we can mimic the original edges by being careful about where we put them. All edges pointed towards v originally should target v_1 in the new graph. An edge (u, v) pointed out of u should original from the $w(u, v)$th copy of u (i.e., $u_{w(u,v)}$). Thus, following the edge (u, v) in the original graph corresponds to the $w(u, v)$-hop path $u_1 \to u_2 \to \cdots \to u_{w(u,v)} \to v_1$ in the augmented graph. We have

$$E' = \{(u_w, v_1)|(u, v) \in E, w = w(u, v)\} \cup \{(u_i, u_{i+1})\} \tag{2}$$

The new graph has $|V'| = W|V|$ vertices and $|E'| = |E| + (W - 1)|V|$ edges. Applying BFS thus runs in $O(WV + E)$ time.

Thinking along the lines of the second solution, you may be tempted to create $w(u, v)$ vertices for each edge (u, v). The problem with that is that you would have $\Theta(WE)$ edges in the worst case, which is worse than $O(WV + E)$. So it is important that we should be careful in the construction to avoid this. Thus, we use a single chain of W vertices per original vertex.