Homework No. 5 Answers

CSCI 4470/6470 Algorithms, CS@UGA, Spring 2018

Due Monday April 16, 2018

Answers are for Questions 1-3 and 4 (requirements 1 and 2), 100 points. Program demo has the other 50 points.

1. (15 points) Let $G = (V, E)$ be a non-directed, edge-weighted graph in which edge weights are all distinct. Assume that edge $(u, v)$ has the second smallest weight. Use the “swapping” technique to prove that every minimum spanning tree contains edge $(u, v)$.

Answer:

The proof should use the “swapping” or “exchange” technique that we have used to prove optimal greedy strategy. The argument may go as follows.

- Let $T$ be any minimum spanning tree for $G$;
- If edge $(u, v) \in T$, then we claim is true and proof is done;
- otherwise, we add edge $(u, v)$ to tree $T$, it results in a cycle;
- because a cycle has to have 3 edges, $\exists (x, y)$ on the cycle that is heavier than $(u, v)$;
- remove $(x, y)$ results in a spanning tree

$$T' = T - \{(x, y)\} \cup \{(u, v)\}$$

with smaller weight, contradicting to that $T$ is a minimum spanning tree.
2. (20 points) Consider the following problem to update a found minimum spanning tree. Let \( G = (V, E) \) be a given non-directed, edge-weighted graph and \( T \) be a given minimum spanning tree \( T \) for \( G \). Assume that an edge \((u, v)\) in \( G \) has its weight changed from \( w(u, v) \) to \( w'(u, v) \). Edge \((u, v)\) may or may not be in \( T \).

Describe algorithm \( \text{UpdateMST}(T, G, w', u, v) \) that produces a minimum spanning tree from \( T \) (i.e., modify \( T \) if you need to) based on the weight change of edge \((u, v)\).

Note:
(a) You cannot directly use an MST algorithm to update for a spanning tree. Your algorithm should try to utilize as much information as possible from \( T \).
(b) Your algorithm \( \text{UpdateMST}(T, G, w', u, v) \) needs to be written in English + mathematical languages. You will not get credit if it is written in any programming language.

Answer:

- First, the algorithm has to be in pseudo code. 10 points off if not.
- 10 points off if the algorithm just simply uses an existing MST algorithm.
- Your algorithm \( \text{UpdateMST} \) may examine two cases (each 10 points)
  (1) If edge \((u, v)\) is not on \( T \),
     (a) do nothing if the weight increases for edge: \( w'(u, v) > w(u, v) \);
     (b) otherwise \( (w'(u, v) < w(u, v)) \), add edge edge \((u, v)\) to tree \( T \),
         • follow the parent point \( \pi \) from \( u \) to visits \( u \)'s ancestors;
         • at the same time, follow the parent point \( \pi \) from \( v \) to visits \( v \)'s ancestors;
         until a common ancestor of \( u \)'s and \( v \)'s is found; a cycle is thus detected;
         • find a lightest edge \((x, y)\) on the cycle;
         • let \( T' = T - \{(x, y)\} \cup \{(u, v)\} \), return \( T' \). (note: \((x, y) \ could \ just \ (u, v)\))

Though not required, but it needs to be aware that when designing \( \text{UpdateMST} \), the complexity to update \( T \) for the above situation (1) is \( O(|V|) \), better than the time required by any MST algorithm.
(2) If edge \((u, v)\) is on \(T\),

(a) do nothing if the weight decreases for edge: \(w'(u, v) < w(u, v)\);
(b) otherwise \((w(u, v) < w'(u, v))\), delete \((u, v)\) from tree \(T\);

• assume that \(\pi(v) = u\).
• search \(T\) to find all vertices that has ancestor \(v\), let the set be \(V_1\);
  let \(V_2 = V - V_1\);
  thus removing \((u, v)\) breaks \(T\) into two smaller subtrees \(T_1\) and \(T_2\);
• find a lightest edge \((x, y)\) cross the cut \((V_1, V_2)\);
• let \(T' = T - \{(u, v)\} \cup \{(x, y)\}\); (note: \((x, y)\) could just \((u, v)\))

Though not required, it needs to be aware that, when designing \(\text{UpdateMST}\),
the complexity to update \(T\) for the above situation (2) is \(O(|V| + |E|)\),
taking into consideration the time to find \(V_1\) and time to search through edges,
better than the time required by any MST algorithm.

Make sure to accept any solution from students different from the above ideas,
as long as it does not completely rely on an existing MST algorithm. (Partially relying on an
MST would be okay).
3. **(15 points)** Consider algorithm Bellman-Ford on the directed graph in Figure 1, using vertex 4 as the source.

(1) Show $d$ and $\pi$ values after each of the $|V| - 1$ passes.

(2) Modify the graph so that it contains at least one negative cycle. Run the Bellman-Ford algorithm again to show how the negative cycle can be detected by the algorithm.

**Answer:**

(1) **(10 points)** Correctness; giving partial credits

(2) **(5 points)** correctness
4. (100 points) This question asks you to design and implement a Google Map-like program using the learned shortest path algorithms.

**The map:** the map is a non-directed graph $G = (V, E)$. Each vertex $v \in V$ represents a traffic intersection and an edge $(u, v) \in E$ represents the two-way street (or road) from interactions $u$ to $v$. The weight on every edge $(u, v)$, denoted with $\tau(u, v)$, is the time to travel between $u$ and $v$. You may think of the map looks like Figure 2 (and perhaps much larger).

![Figure 2: Illustration map for Question 4.](image)

**The problem:** Without doubt, shortest path algorithms can be used to find a “static” shortest path from starting position $s$ to destination $t$. But your Google Map-like algorithm should be able to take real-time traffic information into consideration and to dynamically choose a shortest path from the current position $p$ to the destination $t$. That is, at the current position $p$, the algorithm checks if a travel time change on the street $(u, v)$ and makes an adjustment to the current path.

**Requirement 1 (25 points):** Describe (in English + mathematical languages) your algorithm that can make an adjustment to the shortest path $p \rightarrow t$ when the travel time on street $(u, v)$ increases from $\tau(u, v)$ to $\tau'(u, v)$, i.e., $\tau(u, v) < \tau'(u, v)$. Here $(u, v)$ is assumed to be on the path $p \rightarrow t$. You need to assess the time complexity for the adjustment step.

**Requirement 2 (25 points):** Extended from Requirement 1, now $(u, v)$ may not be on the current path $p \rightarrow t$ and travel time $(u, v)$ may also decrease, i.e., it is also possible $\tau(u, v) > \tau'(u, v)$. Your algorithm should consider this situation in adjusting the shortest path $p \rightarrow t$. You need to assess the time complexity for the adjustment step.

**You should write algorithms for requirements 1 and 2 separately.**
Answer:

Neither requirement 1 or 2 has an obvious theoretical better solution than simply running a shortest path again. So these two questions ask students to open mind and propose anything they can come up with.

Requirement 1 (25points)

- an idea that is different from an existing shortest path algorithm;
- OR propose to use an existing shortest path algorithm
  BUT have to explaining why his/her own conceived ideas were not better.

Requirement 2 (25points)

same as Requirement 1