Homework No. 5 Answers
CSCI 4470/6470 Algorithms, CS@UGA, Spring 2017
Due April 6, 2017

120 points in total (including 20 bonus points for undergraduates).

1. (20 points) Let $G = (V, E)$ be a weighted graph (with edge weight function $w$). Assume that edges in $G$ have unique weights. Let $(u, v)$ and $(x, y)$ be the two edges with the two least weights, and $w(u, v) < w(x, y)$.

   (a) Prove that every minimum spanning tree has to contain edge $(u, v)$.
   (b) Does every minimum spanning tree have to contain edge $(x, y)$ as well? Explain.

   Note that this question has nothing to do with how the tree $T$ was generated.

   **Answers:**

   Both (a) and (b) are to test the knowledge of MST and the usage of proof-by-contradiction.

   (a) (10 points) A proof should have the following basic logic.
   (1) Let $(u, v)$ be the edge of the least weight. Let $T$ be any MST.
   (2) Assume that $T$ does not contain edge $(u, v)$.
   (3) Adding edge $(u, v)$ to $T$ forms a cycle.
   (4) Removing from $T$ a different edge $(x, y)$ on the cycle, resulting in another spanning tree $T'$.
   (5) Because edges in $G$ have unique weights and edge $(u, v)$ is of the least weight, $w(u, v) < w(x, y)$.
   (6) So $T'$ is a spanning tree of weight less than that of $T$. This contradicts that $T$ is an MST.
(7) So the assumption in (2) is incorrect.

(8) We conclude that edge \((u, v)\) is in \(T\).

(b) (10 points) Similar to that for (a).
2. (15 points) Let $G = (V, E)$ be a weighted graph (with edge weight function $w$) and $T \subseteq E$ be a minimum spanning tree of $G$. Prove that for every edge $(u, v) \in T$, there is a cut in graph $G$ such that $(u, v)$ is a light edge crossing the cut. **Note that this question has nothing to do with how the tree $T$ was generated.**

**Answers:**

This question is to test if the student can use the learned concepts “cut, cross edge”, etc and the notion of MST. The answer should not rely on an algorithm for MST or the idea of how an MST is constructed, as noted. A “standard” argument/proof may go as follows.

(1) Let $T$ be any MST and $(u, v)$ be an edge in $T$.
(2) Vertices on $T$ are naturally partitioned into two subsets: $U$ and $V - U,$ where every vertex in $U$ reaches $u$ before reaching $v$ on $T$. It is clear that $u \in U$ and $v \in V - U$
(3) By the definition of cut, $(U, V - U)$ is a cut for the graph $G$.
(4) **Assume** $(u, v)$ is not a light edge crossing the cut $(U, V - U)$, there must be a different edge $(x, y)$ crossing the cut such that $w(x, y) < w(u, v)$
(5) Adding $(x, y)$ to $T$ forms a cycle.
(6) Let $T' = T - \{(u, v)\} \cup \{(x, y)\}$ is a spanning of weight less than that of $T$. This contradicts that $T$ is an MST.
(7) So the assumption in (4) is incorrect.
(8) We conclude that there is a cut $(U, V - U)$ and edge $(u, v)$ is a light edge cross the cut.
3. **(15 points)** (1) How does algorithm MST-Kruskal identify a cut and pick a light edge crossing the cut to add to the set $A$?

(2) How does algorithm MST-Prim store $A$? How does the algorithm ensure that the edge newly added to set $A$ does not form a cycle with those edges already in $A$?

**Answers:**

(1) **(5 points)** MST-Kruskal finds the lightest edge $(u, v)$ among the remaining edges (that have not be included in set $A$). There may be more than one cut as a result of this. But all of them has edge $(u, v)$ being the lightest crossing edge.

(2) **(5 points)** MST-Prim stores edges in $T$ with a predecessor point $\pi$, such that $v.\pi = u$ if and only if edge $(u, v)$ is included in $A$.

**5 points** Once a vertex $v$ is considered and the corresponding edge $(u, v)$ is added to the set $A$, where $v.\pi = u$, $v$ is removed from the priority queue $Q$ and will not be considered again. This ensures that no cycle would occur in $A$. 

4. (20 points) (1) Run the Bellman-Ford algorithm on the directed graph of Figure 1, using vertex 4 as the source. 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.

Figure 1: Graph for Question 4.

Answers:

(1) (10 points) Each of the 4 passes by Bellman-Ford needs to be shown.

(2) (3 points) creating a negative cycle.

(7 points) Show how Bellman-Ford checks the negative cycle.
5. **(15 points)** Bellman-Ford algorithm was designed to be able to handle graphs that may contain negative cycles. When the input graph is a DAG (directed acyclic graph), would the algorithm use the same amount of time as algorithm DAG-Shortest Paths? Explain.

**Answers:**

Bellman-Ford differs from DAG-Shortest Paths in at least the following aspects:

(a) the relaxation is not following any topological order of vertices;
(b) it does not know the graph is a DAG, so it may still go through $|V| - 1$ rounds of relaxation. So it takes longer running time;
6. **(15 points)** Give a simple example of a directed graph with negative weight edges for which Dijkstra’s algorithm produces incorrect answers. Also argue that, if all negative weights occur only on the edges leaving the source \( s \), Dijkstra’s algorithm remains correct.

**Answers:**

**(10 points)** The student needs to give such a counter example to Dijkstra’s algorithm and show it works incorrectly.

A typical example is, once an edge \((u, v)\) has been relaxed, but \(d.u\) may get updated later due to some newly discovered negative edge \((w, u)\). This means that edge \((u, v)\) needs to be relaxed again to update \(d.v\). But Dijkstra’s algorithm does not relax an edge more than once.

**(5 points)** This is, relating to the aforementioned example, \(w = s\) so \(d.u\) will not be updated after it is set a non-\(\infty\) value.
7. (20 points, Graduate students only, bonus for undergraduates) How do we use FLOYD-WARSHALL algorithm to detect if the input graph contains a negative weight cycle?

**Answers:**

If all edges are non-negative, the computed matrix will be non-negative. Now if there is a negative cycle, it means that for vertices on that cycle, say $v$, has the distance $d[v, v] < 0$ to itself. So it suffices to examine the diagonal of the computed matrix to see if there is a negative number on it.