¡Felicitaciones! ¡Aprobaste!

Calificación recibida 100 % **Para Aprobar** 80 % o más

Ir al siguiente elemento

Problem Set #2

Calificación de la entrega más reciente: 100 %

1. Suppose we are given a *directed graph* $G = (V, E)$ in which every edge has a distinct positive edge weight. A directed graph is *acyclic* if it has no directed cycle. Suppose that we want to compute the maximum-weight acyclic subgraph of G (where the weight of a subgraph is the sum of its edges' weights). Assume that G is weakly connected, meaning that there is no cut with no edges crossing it in either direction.

1 / 1 punto

Here is an analog of Prim's algorithm for directed graphs. Start from an arbitrary vertex s, initialize $S = \{s\}$ and $F = \emptyset$. While $S \neq V$, find the maximum-weight edge (u, v) with one endpoint in S and one endpoint in $V - S$. Add this edge to F, and add the appropriate endpoint to S.

Here is an analog of Kruskal's algorithm. Sort the edges from highest to lowest weight. Initialize $F = \emptyset$. Scan through the edges; at each iteration, add the current edge i to F if and only if it does not create a directed cycle.

Which of the following is true?

- ○ Both algorithms always compute a maximum-weight acyclic subgraph.
- ◉ Both algorithms might fail to compute a maximum-weight acyclic subgraph.
- ○ Only the modification of Prim's algorithm always computes a maximum-weight acyclic subgraph.
- ○ Only the modification of Kruskal's algorithm always computes a maximum-weight acyclic subgraph.

Correcto
Indeed. Any ideas for a correct algorithm?

2. Consider a connected undirected graph G with edge costs that are *not necessarily distinct*. Suppose we replace each edge cost c_e by $-c_e$; call this new graph G'. Consider running either Kruskal's or Prim's minimum spanning tree algorithm on G', with ties between edge costs broken arbitrarily, and possibly differently, in each algorithm. Which of the following is true?

1 / 1 punto

- ○ Kruskal's algorithm computes a maximum-cost spanning tree of G but Prim's algorithm might not.
- ○ Both algorithms compute the same maximum-cost spanning tree of G.
- ○ Prim's algorithm computes a maximum-cost spanning tree of G but Kruskal's algorithm might not.
- ◉ Both algorithms compute a maximum-cost spanning tree of G, but they might compute different ones.

Correcto
Different tie-breaking rules generally yield different spanning trees.

3. Consider the following algorithm that attempts to compute a minimum spanning tree of a connected undirected graph G with distinct edge costs. First, sort the edges in decreasing cost order (i.e., the opposite of Kruskal's algorithm). Initialize T to be all edges of G. Scan through the edges (in the sorted order), and remove the current edge from T if and only if it lies on a cycle of T.

1 / 1 punto

Which of the following statements is true?

- ○ The output of the algorithm will never have a cycle, but it might not be connected.
- ◉ The algorithm always outputs a minimum spanning tree.
- ○ The output of the algorithm will always be connected, but it might have cycles.
- ○ The algorithm always outputs a spanning tree, but it might not be a minimum cost spanning tree.

Correcto
During the iteration in which an edge is removed, it was on a cycle C of T. By the sorted ordering, it must be the maximum-cost edge of C. By an exchange argument, it cannot be a member of any minimum spanning tree. Since every edge deleted by the algorithm belongs to no MST, and its output is a spanning tree (no cycles by construction, connected by the Lonely Cut Corollary), its output must be the (unique) MST.

4. Consider an undirected graph $G = (V, E)$ where edge $e \in E$ has cost c_e. A *minimum bottleneck spanning tree* T is a spanning tree that minimizes the maximum edge cost $\max_{e \in T} c_e$. Which of the following statements is true? Assume that the edge costs are distinct.

1 / 1 punto

- ○ A minimum bottleneck spanning tree is always a minimum spanning tree and a minimum spanning tree is always a minimum bottleneck spanning tree.
- ○ A minimum bottleneck spanning tree is always a minimum spanning tree but a minimum spanning tree is not always a minimum bottleneck spanning tree.
- ◉ A minimum bottleneck spanning tree is not always a minimum spanning tree, but a minimum spanning tree is always a minimum bottleneck spanning tree.
- ○ A minimum bottleneck spanning tree is not always a minimum spanning tree and a minimum spanning tree is not always a minimum bottleneck spanning tree.

Correcto
For the positive statement, recall the following (from correctness of Prim's algorithm): for every edge e of the MST, there is a cut (A, B) for which e is the cheapest one crossing it. This implies that every other spanning tree has maximum edge cost at least as large. For the negative statement, use a triangle with one extra high-cost edge attached.

5. You are given a connected undirected graph G with distinct edge costs, in adjacency list representation. You are also given the edges of a minimum spanning tree T of G. This question asks how quickly you can recompute the MST if we change the cost of a single edge. Which of the following are true? [RECALL: It is not known how to deterministically compute an MST from scratch in $O(m)$ time, where m is the number of edges of G.] [Check all that apply.]

1 / 1 punto

- [x] Suppose $e \notin T$ and we increase the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

Correcto
The MST does not change (by the Cycle Property of the previous problem), so no re-computation is needed.

- [x] Suppose $e \in T$ and we increase the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

Correcto
Let A, B be the two connected components of $T - \{e\}$. Edge e no longer belongs to the new MST if and only if it is no longer the cheapest edge crossing the cut (A, B) (this can be checked in $O(m)$ time). If f is the new cheapest edge crossing (A, B), then the new MST is $T - \{e\} \cup \{f\}$.

- [x] Suppose $e \in T$ and we decrease the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

Correcto
The MST does not change (by the Cut Property), so no re-computation is needed.

- [x] Suppose $e \notin T$ and we decrease the cost of e. Then, the new MST can be recomputed in $O(m)$ deterministic time.

Correcto
Let C be the cycle of $T \cup \{e\}$. Edge e belongs to the new MST if and only if it is no longer the most expensive edge of C (this can be checked in $O(n)$ time). If f is the new most expensive edge of C, then the new MST is $T \cup \{e\} - \{f\}$.