CS 170 Fall 2014 — Algorithms — David Wagner

HW 6

Due Oct. 10, 6:00pm

Instructions. This homework is due Friday, October 10th, at 6:00pm electronically. Same rules as for prior homeworks. See `http://www-inst.cs.berkeley.edu/~cs170/fa14/hws/instruct.pdf` for the required format for writing up algorithms.

1. (20 pts.) Fundamental concepts

Decide whether each of the following claims is correct or not. If it is correct, write "True" and then provide a short proof (one or two sentences should be enough). If it is incorrect, write "False" and provide a counterexample (please make your counterexample as small as possible, for the readers' sake).

(a) Let G be a dag and suppose the vertices $v_1, \ldots, v_n$ are in topologically sorted order. If there is an edge (v_i, v_j) in G, then we are guaranteed that $i < j$.

(b) Let G be a dag and suppose the vertices $v_1, \ldots, v_n$ are in topologically sorted order. If there is a path from v_i to v_j in G, then we are guaranteed that $i < j$.

(c) Let $G = (V, E)$ be a directed graph. If there is an edge (u, v) in G, then u and v must be in the same strongly connected component.

(d) Let $G = (V, E)$ be a directed graph. If $s \in V$ is a source and $t \in V$ is a sink, then there is a path from s to t.

(e) Let $G = (V, E)$ be a directed graph where every vertex has at most three outgoing edges. Then every vertex has at most three incoming edges.

(f) Let $G = (V, E)$ be a dag. Then there are at most $|V|^2$ possible ways to order the vertices so they are in topologically sorted order.

(g) Let G be a connected undirected graph with positive lengths on all the edges. Let s be a fixed vertex. Let $d(s, v)$ denote the distance from vertex s to vertex v, i.e., the length of the shortest path from s to v. If we choose the vertex v that makes $d(s, v)$ as small as possible, subject to the requirement that $v \neq s$, then every edge on the path from s to v must be part of every minimum spanning tree of G.

Revised 10/8: G is connected.

(h) Let G be a connected undirected graph with positive lengths on all the edges, where no two edges have the same length. Let s be a fixed vertex. Let $d(s, v)$ denote the distance from vertex s to vertex v, i.e., the length of the shortest path from s to v. If we choose the vertex v that makes $d(s, v)$ as small as possible, subject to the requirement that $v \neq s$, then every edge on the path from s to v must be part of every minimum spanning tree of G.

Revised 10/8: G is connected.

2. (20 pts.) BFS?

We previously used DFS and `post` times for topological sorting. What if we use BFS? In particular, consider the following minor modification to BFS:

```
BFS($G$, $s$):
1.  For each $u \in V$:
2.      Set dist($u$) $:= \infty$.
3.  Set dist($s$) $:= 0$.
4.  Set clock $:= 0$.
5.  Initialize $Q$ to a queue containing just $s$.
6.  While $Q$ is not empty:
7.      $u :=$ eject($Q$) (pop the front element off $Q$, and call it $u$)
8.      For each edge $(u,v) \in E$:
9.          If dist($v$) $= \infty$:
10.             inject($Q,v$) (append $v$ to the end of $Q$)
11.             Set dist($v$) $:=$ dist($u$) $+ 1$.
12.     Set post($u$) $:=$ clock, and clock $:=$ clock $+ 1$.
```

Let G be a directed acyclic graph. Suppose there is a source $s \in V$ such that every vertex in G is reachable from s. Suppose we run BFS(G, s), and then sort the vertices according to ~~decreasing~~ increasing `post`-value. Is this guaranteed to produce a valid topological sorting of the vertices? Either prove that the resulting algorithm is correct (i.e., that it is guaranteed to output a valid linearization of G), or show a small counterexample (a dag where the algorithm fails).

Revised 10/5: use *increasing* `post` values, not decreasing.

3. (~~20~~ 25 pts.) Travel planning

You are given a set of n cities and an $n \times n$ matrix M, where $M(i,j)$ is the cost of the cheapest direct flight from city i to city j. All such costs are non-negative. You live in city A and want to find the cheapest and most convenient route to city B, but the direct flight might not be the best option. As far as you're concerned, the best route must have the following properties: the sum of the costs of the flights in the route should be as small as possible, but if there are several possible routings with the same total cost, then you want the one that minimizes the total number of flights. Design an efficient algorithm to solve this problem. Your algorithm should run in $O(n^2 \lg n)$ time.

(If you want, you can assume that the best route will require fewer than, say, 1000 flights, and that the cost of each flight is an integer number of dollars.)

Revised 10/8: The flight costs are all non-negative.

4. (~~20~~ 25 pts.) Road network design

There is a network of roads $G = (V,E)$ connecting a set of cities V. Each road $e \in E$ has an associated (non-negative) length $\ell(e)$. We can build *one* new road, and there are a bunch of candidates for where this road might go. As a designer for the public works department, you are asked to determine which of these candidates, if added to the existing network G, would result in the maximum decrease in the driving distance between two fixed cities s and t. Design an $O((|V|+|E|)\log|V|)$ time algorithm for this problem.

In particular, the problem is:

Input: an undirected graph G with non-negative edge lengths $\ell : E \to \mathbb{R}_{\geq 0}$, and two vertices s,t, and a list $(a_1,b_1),\ldots,(a_n,b_n)$ of pairs of vertices

Output: the index i such that adding the edge (a_i,b_i) reduces the distance from s to t as much as possible

5. (10 pts.) MSTs for directed graphs

Kruskal's algorithm takes as input an *undirected* graph that's connected, and finds a connected subgraph of minimum weight. But suppose we have a *directed* graph that's strongly connected, and we want to find a strongly connected subgraph of minimum weight. Will Kruskal's algorithm work for this task?

In particular, consider the following algorithm:

```
ModifiedKruskal($G = (V,E)$, $w$):
1.  Set $X := \{\}$ (the empty set).
2.  Sort the edges by weight.
3.  For each edge $(u,v) \in E$, in increasing order of weight:
4.      If there is a path from $u$ to $v$ using only edges in $X$, do nothing, else add $(u,v)$ to $X$.
5.  Return $X$.
```

Suppose we are given a *directed* graph $G = (V,E)$ that is strongly connected, with weights $w : E \to \mathbb{R}$ on the edges. Suppose we run ModifiedKruskal(G, w) and get the result X. Define the graph G^* by $G^* = (V,X)$; in other words, G^* has the edges returned by the above algorithm. Are we guaranteed that G^* has the minimal possible total weight, out of all possible graphs $G' = (V,E')$ such that $E' \subseteq E$ and G' is strongly connected? Either prove that the answer is yes, or give a small counterexample to show that the answer is no.

6. (0 pts.) Crazy hard optional problem: Two vertex-disjoint paths

(This is an *optional* bonus challenge problem. Only solve it if you want an extra challenge.)

Find a polynomial-time algorithm to solve the following problem:

Input: A dag G, and vertices s_1, s_2, t_1, t_2.

Question: Does there exist a path from s_1 to t_1 and a path from s_2 to t_2, such that no vertex appears in both paths?

Your algorithm should have running time $O(|V|^c)$ for some constant c (e.g., $c = 4$ or something like that).

Don't submit a solution. We won't grade it. Health warning: This problem may be brain-meltingly hard.