CS 170 Efficient Algorithms and Intractable Problems
Spring 2017 Prasad Raghavendra and Sanjam Garg

Homework 3

Instructions: You are welcome to form small groups (up to 4 people total) to work through the homework, but you **must** write up all solutions by yourself. List your study partners for homework on the first page, or "none" if you had no partners.

If using LaTeX (which we recommend), you may use the homework template linked on this Piazza post to get started.

Begin each problem on a new page. Clearly label where each problem and subproblem begin. The problems must be submitted in order (all of P1 must be before P2, etc). For questions asking you to give an algorithm, respond in what we will refer to as the *four-part format* for algorithms: main idea, pseudocode, proof of correctness, and running time analysis.

Read the Homework FAQ Piazza post on Piazza before doing the homework for more explanation on the four-part format and other clarifications for our homework expectations.

No late homeworks will be accepted. No exceptions. This is not out of a desire to be harsh, but rather out of fairness to all students in this large course. Out of a total of approximately 12 homework assignments, the lowest two scores will be dropped.

Special Questions:

- *Redemption questions*: It is important that you carefully read the posted solutions, even for problems you got right. To encourage this, you have the option of submitting a redemption file, a few paragraphs in which you explain, for each problem you choose to cover, what you did wrong and what the right idea was in your own words (not cutting and pasting from the solution!), and appending it to your homework. For example, suppose that as you review your solutions to HW1, you realize you had misunderstood question 3 and answered it incorrectly. You would write down what you just learned, and then submit it in your HW2 assignment the following week. Because these are mainly for your benefit, feel free to format them however is most useful for you.

- *Extra credit questions*: We might have some extra credit questions in the homework for people who really enjoy the material. However, please note that you should do the extra credit problems only if you really enjoy working on these problems and want an extra challenge. It is likely not the most efficient manner in which to maximize your grade.

Due Tuesday, February 14, at 11:59am

1. (★★ level) Dijkstra's Durability

For the following claims, answer yes or no and provide justification. Consider an arbitrary graph G with positive edge weights. Shortest path means least cost path.

(a) Suppose we want to know the distance from a node s to a node t_1, so we run Dijkstra's algorithm. Now suppose we also want to know the distance from s to another node t_2. Do we need to run the algorithm again?

(b) Suppose we know the shortest path from a node s to another node t. Is the shortest path (the sequence of nodes, not the total cost) always the same if we add k to all edge weights, where k is an arbitrary positive number?

(c) Now answer the same question, except we multiply by k instead of adding k.

2. (★★★★★ level) Biconnected Components

Let $G = (V, E)$ be a connected undirected graph. For any two edges $e, e' \in E$, we say that $e \sim e'$ if either $e = e'$ or there is a (simple) cycle containing both e and e'. We say a set of edges C is a *biconnected component* if there is some edge $e \in E$ such that $C = \{e' \in E | e \sim e'\}$.

(a) Show that two distinct biconnected components cannot have any edges in common. (Note: That this is equivalent to showing that if $e' \sim e_1$ and $e' \sim e_2$, then $e_1 \sim e_2$.)

(b) Associate with each biconnected component all the vertices that are endpoints of its edges. Show that the vertex sets corresponding to two different biconnected components are either disjoint or intersect in a single vertex. Such a vertex is a *separating vertex.*

Note that a separating vertex, if removed, would disconnect the graph. We will now walk you through how you can use DFS to identify the biconnected components and separating vertices of a graph in linear time. Consider a DFS tree of G.

(c) Show that the root of the DFS tree is a separating vertex if and only if it has more than one child in the tree.

(d) Show that a non-root vertex v of the DFS tree is a separating vertex if and only if it has a child v' none of whose descendants (including itself) has a backedge to a proper ancestor of v.

For each vertex u define $\texttt{pre}(u)$ to be the pre-visit time of u. For two vertices u and w, w is a **backcestor** of u iff the following two conditions both hold:

1. w is an ancestor of u
2. $\exists$ (back) edge (u, w) OR $\exists$ (back) edge (v, w), where v is some descendant of u in the DFS tree.

Define $\texttt{low}(u)$ to be the minimum possible value of $\texttt{pre}(w)$, where w is a backcestor of u.

Another way of stating the result of (d) is that a non-root vertex u is a separating vertex if and only if $\texttt{pre}(u) \leq \texttt{low}(v)$ for any child v of u.

(e) Give an algorithm that computes all separating vertices and biconnected components of a graph in linear time. (Hint: Think of how to compute `low` in linear time with DFS. Use `low` to identify separating vertices and run an additional DFS with an extra stack of edges to remove biconnected components one at a time.)

No need for a full 4-part solution. Just give a clear description of the algorithm, in plain English or psuedocode, and running time analysis.

As an example, part (a) is done for you.

a) Let B and C be two distinct biconnected components defined by $B = \{e' | e_1 \sim e'\}$ and $C = \{e' | e_2 \sim e'\}$, where e_1 and e_2 are distinct edges. Suppose B and C share edge f in common. Then $f \sim e_1$ and $f \sim e_2$. But this implies $e_1 \sim e_2$.

   To see why, note that is clearly true if $f = e_1$ or $f = e_2$. Otherwise, there is a simple cycle containing f and e_1 and a simple cycle containing f and e_2. The union of these two simple cycles contains a simple cycle that has both e_1 and e_2 as edges. (Convince yourself why. Drawing some visuals may help.)

   By a similar argument, if $e' \sim e_1$ then $e' \sim e_2$ because $e_1 \sim e_2$. Likewise, if $e' \sim e_2$ then $e' \sim e_1$. Thus, $B = C$, contradicting the fact that they are distinct sets.

3. (★★★ level) Alternative Factory

You have the unfortunate predicament that you frequently find yourself inexplicably trapped within factories, a different one every time. Each factory consists of many small platforms, connected by a network of conveyor belts. Being an amateur surveyor, you can accurately calculate the time it takes to ride each conveyor belt with a quick glance. Some platforms have a button on them. Whenever you press one of these buttons, ALL conveyor belts in the factory reverse direction. A factory has one exit; you want to get there as quickly as possible. Give an efficient ($O((|V|+|E|)\log|V|)$ time) algorithm to find the fastest way out of the factory.

You only need to give a main idea and running time analysis. Be sure your main idea clearly and fully describes your algorithm.

For convenience, you can treat the factory as a graph, with the platforms being vertexes and the conveyor belts being directed edges. You know before running your algorithm the platform s at which you start, the location of the exit t, and the location of all buttons (think of this as a list of button locations). You can determine the time it takes to traverse the conveyor belt between any two platforms u and v via the function $\ell(u,v)$.

Notes in case you want to pick at the details:
-It takes a negligible amount of time to get from one end of a platform to the other.
-The time it takes to press a button is also negligible.
-You can't run up a conveyor belt the wrong way.
-You may assume $\ell(u,v) = \ell(v,u)$.
-Be warned that you should throw out your sense of spatial reasoning. You have seen all kinds of setups of conveyor belt configurations, sometimes with thousands of conveyor belts connected to a single platform.

(Hint: Try to construct a new graph that encodes the state of the conveyor belts. Perhaps your new graph will have twice as many vertexes as the original.)

4. (★★★ level) Shortest Path Oracle

Given a connected, directed graph $G = (V,E)$ with positive edge weights, and vertices s and t, you want to find the shortest path from s to t in a graph in $O(|E|+|V|)$ time. Normally, this wouldn't be possible; however, you have consulted the Order Oracle, who gave you a list L of the V vertices in increasing order of the shortest-path distance from s.

In other words, $L[1]$ is s, $L[2]$ is the closest other vertex to s, $L[|V|]$ is the farthest vertex from s, etc. Deign an algorithm $\textsc{ShortestPath}(G,s,t,L)$ which computes the shortest $s-t$ path in $O(|E|+|V|)$ time.

5. (★★★★★ level) Archipelago Adventure

As a tourist visiting an island chain, you can drive around by car, but to get between islands you have to take the ferry. (more formally, our graph is $G = (V, R \cup F)$, where V is our set of locations (vertexes), R is the set of roads, and F is the set of ferry routes). A road is specified as an **undirected** edge $(u, v, C_{u,v})$ that connects the two points u, v with a **non-negative** (time) cost $C_{u,v}$. A ferry is specified as a **directed** edge $(u, v, C_{u,v})$ which connects point u to point v on another island (one way) with cost $C_{u,v}$. Some of the ferries are actually magical fairies so their costs **can possibly be negative!** Each ferry only runs one way and the routes are such that if there is a ferry from u to v, then there is no sequence of roads and ferries that lead back from v to u.

Assume we have the the adjacency list representation of roads and fairies (two separate data structures).

You are finished with your vacation so you want to find the shortest (least cost) route from your current location s to the airport t. Design an efficient algorithm to compute the shortest path from s to t (given V, R, F, s and t as inputs). Your running time should be $O((|V| + |E|) \log |V|)$, where $E = R \cup F$.

Hint: Notice that the directed (ferry) edges are between different islands, each of which forms a separate (strongly) connected component.

Hint 2: If you are stuck, try thinking about the following questions (no credit for answering).

- *Consider a directed graph with no negative cycles in which the only negative edges are those that leave some node s; all other edges are positive. Will Dijkstra's algorithm, started at s, work correctly on such a graph? Find a counterexample or prove your answer.*
- *Now consider the case in which the only negative edges are those that leave v, a vertex different from the start node s. Given two vertices s, t, find an efficient algorithm to compute the shortest path from s to t (should have the same asymptotic running time as Dijkstra's algorithm).*