Problem Set 5

This problem set explores greedy algorithms, spanning trees, and related topics. We hope that it helps cement your understanding of "greedy stays ahead" arguments, exchange arguments, and properties of trees.

Please be sure to write your answers different problems on separate pieces of paper to make it easier for us to grade. Also, **please put your name on each page of your assignment**.

As always, please feel free to drop by office hours or send us emails if you have any questions. We'd be happy to help out.

This problem set has 35 possible points. It is weighted at 12% of your total grade.

Good luck, and have fun!

Due Monday, August 5 at 2:15 PM

Problem One: Fixing an MST (8 Points)

Suppose that you've computed an MST T^* for a graph $G = (V, E)$. Afterwards, you add a new edge $(u, v) \notin E$ to the graph with cost $c(u, v)$. Depending on the shape of the graph, the initial edge weights, and the weight of (u, v), your old MST T^* might not necessarily be an MST of the new graph $G' = (V, E \cup \{(u, v)\})$.

Assume that all edge weights in G' are distinct (which means that the edge weights in G are also distinct). Design an $O(n)$-time algorithm that determines whether T^* is an MST of G'. Then:

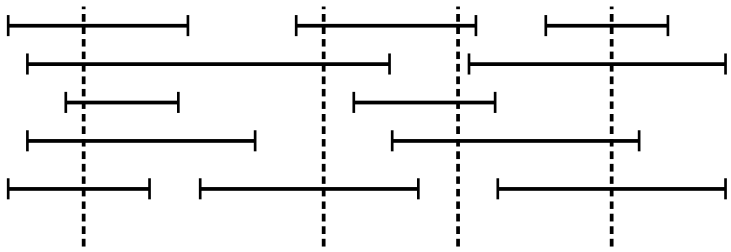
- Describe your algorithm.
- Prove that your algorithm determines whether T^* is an MST of G'.
- Prove that your algorithm runs in time $O(n)$.

You can assume T^* and G' are given to you as adjacency lists.

Problem Two: Making Announcements (8 Points)

Suppose that you are in charge of the PA system at a school. When you make an announcement over the PA system, all classes currently in session hear it. There is a very important announcement you need to make, and to ensure that everyone hears it, you want to announce it enough times to guarantee that every single class hears the announcement at least once. Because some students might be in multiple classes during the day, you want to minimize the number of times that you have to make the announcement. It's fine if you make the announcement two or more times during a particular class; it might be a nuisance, but your priority is ensuring that each class hears it at least once.

For example, if you were given the set of classes (denoted by their timespans), one possible set of times you could make the announcements is given by the vertical dotted lines:



This particular set of times isn't a minimum set of times; it's possible to cover each class using only three announcements.

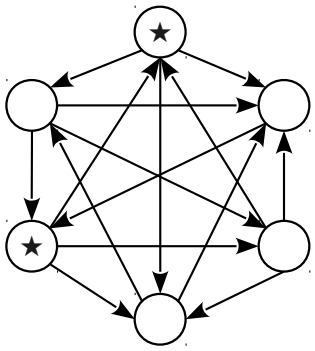
You are given as input a list of intervals representing the start and end times of each class. Design a polynomial-time algorithm that finds a minimal set of announcement times that is guaranteed to reach every class. Then:

- Describe your algorithm.
- Prove that your algorithm finds a minimal set of announcement times.
- Prove that your algorithm runs in polynomial time.

For simplicity, you can assume that no class starts as soon as another one ends and that when you start making the announcement all classes in session will hear it, including those that are just starting or just ending.

Problem Three: Tournament Dominating Sets (8 Points)

A *tournament graph* is a directed graph with $n \geq 1$ nodes where every pair of distinct nodes has exactly one edge between them. Intuitively, a tournament graph represents the result of a tournament in which every player plays exactly one game against every other player, where an edge (u, v) means that u beat v in her game against him. For example, the following graph is a tournament graph:



For any graph $G = (V, E)$, a *dominating set* is a set $D \subseteq V$ such that for every node $v \in V$, at least one of the following are true:

- $v \in D$, or
- There is a node $u \in D$ where $(u, v) \in E$.

In other words, every node either belongs to the dominating set or is a child of a node in the dominating set (or both). The starred nodes in the above graph form a dominating set, though it's not the only dominating set in the tournament.

The set of all nodes is a dominating set for any graph G; the challenge is finding small dominating sets. The problem of finding the smallest dominating set in a graph is **NP**-hard, and while it's unknown whether it's still **NP**-hard to find the smallest dominating set in a tournament graph, no polynomial-time algorithms are known for finding one.

However, it turns out that we can guarantee that there is a dominating set in a tournament graph that is not too large relative to the total number of nodes, so even if we can't find the *absolute* smallest dominating set in a tournament efficiently, we can still find a dominating set that is not too much larger than the smallest one.

Design a polynomial-time algorithm that, given a tournament graph containing n nodes, finds a dominating set of size $O(\log n)$. Then:

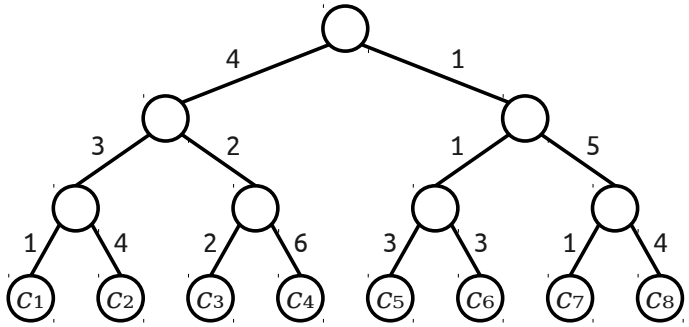
- Describe your algorithm.
- Prove your algorithm returns a dominating set of size at most $O(\log n)$.
- Prove that your algorithm runs in polynomial time.

An interesting aside – your algorithm shows that every tournament graph has a dominating set of size $O(\log n)$. This means that we can find the smallest dominating set in a tournament graph by enumerating all subsets of V of size at most $O(\log n)$ in ascending order of size until we find one that is a dominating set. Since there are $n^{\Theta(\log n)}$ subsets of V of size $O(\log n)$, this means that this problem can be solved in time $n^{\Theta(\log n)}$. Note that this function grows faster than any polynomial function but slower than any exponential function. It's not known whether this problem is in **P** or is **NP**-complete, making it one of the few known problems in **NP** with this property.

Problem Four: Fixing Clock Skew (10 Points)*

When designing hardware devices, certain signals – especially clock signals – need to physically arrive at multiple different devices at exactly the same time. If they don't, the devices can get out of sync with one another, causing the hardware to malfunction. This is called *clock skew*.

Suppose that you have $n = 2^k$ different components that all need to receive a clock signal at exactly the same time. To do so, you create a perfect binary tree, where each internal node represents a junction and each leaf node represents one of the n components. The clock signal originates at the root of the binary tree and propagates downward to the leaves. Each edge (u, v) represents a wire, and the signal takes some amount of time $t(u, v)$ to propagate across the wire. Given arbitrary propagation times, there is no guarantee that the signal will arrive at all of the leaves at exactly the same time, and so you will need to introduce artificial delays into the system by *increasing* the delays along some of the wires (you cannot decrease the delays). The *cost* of adding delays into the system is given by the total amount of extra delay added to all the edges. For example, one way to remove clock skew from the tree to the right would be to increase each edge so that it has delay 6. This has cost 44. A better way to do this would be to raise the costs of all edges in the bottom layer to 6, all edges in the middle layer to 5, and all edges in the top layer to 4. This has cost 36. Even better solutions exist.



Design a polynomial-time algorithm for solving this problem. Then:

- Describe your algorithm.
- Prove your algorithm finds the lowest-cost way to increase edge delays so that all root-leaf paths have the same total delay. *(Hint: If two solutions disagree, there must be at least one deepest edge in the tree where they disagree. Focus on any one of those edges.)*
- Prove your algorithm runs in polynomial time.

Problem Five: Course Feedback (1 Point)

We want this course to be as good as it can be, and we'd appreciate your feedback on how we're doing. For a free point, please answer the following questions. We'll give you full credit no matter what you write, as long as you write something.

i. How hard did you find this problem set? How long did it take you to finish? Does that seem unreasonably difficult or time-consuming for a five-unit course?

ii. Did you attend office hours? If so, did you find them useful?

iii. Did you read through the textbook? If so, did you find it useful?

iv. How is the pace of this course so far? Too slow? Too fast? Just right?

v. Is there anything in particular we could do better? Is there anything in particular that you think we're doing well?

* Adapted from Exercise 4.24 of *Algorithm Design* by Kleinberg and Tardos