Problem Set 2

In this second problem set, you'll get to explore graph algorithms applied in a variety of contexts. By the time that you're done with this problem set, we hope that you'll have a much stronger intuition for the graph algorithms we've covered so far and how to apply them to solve meaningful problems.

Please read over the "Problem Set Advice" handout before starting this problem set. It contains information about our grading policies, procedures, and expectations for the assignments. In particular, **please be sure to write your answers different problems on separate pages** to make it easier for us to grade.

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 34 possible points. It is weighted at 12% of your total grade. The earlier questions serve as a warm-up for the later problems, so the difficulty of the problems increases over the course of this problem set.

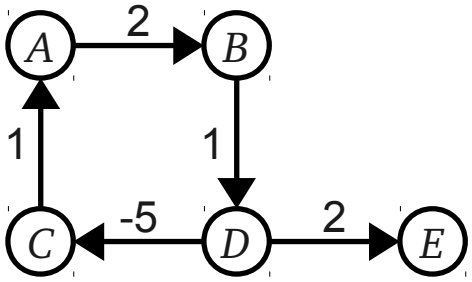
Good luck, and have fun!

Due Friday, July 12 at 2:15 PM

Problem One: Dijkstra's Algorithm and Negative Weights (3 Points)

Suppose that you have a graph $G = (V, E)$ where each edge (u, v) has a length $l(u, v)$, which can be positive, zero, or negative. You are interested in finding the lengths of the shortest paths from some node $s \in V$ to each node $v \in V$. As usual, if there is no path from s to v, we'll say that the cost of the shortest path is ∞.

We have to be careful when edges have negative costs. For example, we can't allow a graph like the one pictured to the right because there is no shortest path from A to B; we can go around the cycle $\langle A, B, D, C \rangle$ arbitrarily many times before following edge (D, E), giving paths of arbitrarily negative cost. Consequently, we will assume that the graph G does not contain any cycles with a negative length, since there might not be shortest paths in that case.



i. Find an example of a graph G with arbitrary edge lengths (but no negative cycles) where running Dijkstra's algorithm starting from some node s will not find the lengths of the shortest path from s to each node in the graph. Describe what the lengths of the shortest paths actually are and what path lengths Dijkstra's algorithm would find.

Now, consider the following modification to Dijkstra's algorithm to handle negative edge lengths: find the edge (u, v) where $l(u, v)$ is minimal and subtract $l(u, v)$ from the length of each edge in the graph. Then, run Dijkstra's algorithm on the resulting graph.

ii. Will this modification to Dijkstra's algorithm always find shortest paths in graphs with no negative cycles? If so, prove that the algorithm is always correct. If not, find an example of a graph G where running Dijkstra's algorithm starting from some node s will not find the lengths of the shortest path from s to each node in the graph. Describe what the lengths of the shortest paths actually are and what path lengths this algorithm would find.

Problem Two: Archaeological Consistency (10 Points)

Suppose that you have found a collection of historical records indicating the relative order in which various people lived and died. Each record tells you one of the following:

- Person A and person B were alive at the same time.
- Person A died before person B was born.

Your task is to determine whether the historical records are *consistent*; that is, whether it was actually possible for all of these people to have lived and died in such a way that all of your records are accurate.

Suppose that your records involve n total people and you have m records relating their lifespans. Design an $O(m + n)$-time algorithm that determines whether or not the records are consistent with one another. Then:

- Describe your algorithm.
- Prove that your algorithm correctly determines whether the records are consistent.
- Prove that your algorithm runs in time $O(m + n)$.

For simplicity, you can assume the people are numbered $0, 1, 2, \ldots, n - 1$.

Problem Three: Constrained Clustering (8 Points)

You are given a collection of points P and want to split P into k nonempty, disjoint sets S_1, S_2, … S_k for some number k of your choosing. Without any restrictions on how you're allowed to split the points apart, this problem is easy – any way of splitting the points apart into nonempty disjoint sets is permissible.

In the *constrained clustering problem*, you are given a set of constraints that govern whether certain pairs of points must or must not be in the same cluster as one another. Specifically, each constraint will be one of the following two types:

- A **must-link** constraint of the form **ML**(p_1, p_2), which states that points p_1 and p_2 must be placed into the same cluster, and
- A **cannot-link** constraint of the form **CL**(p_1, p_2), which states that points p_1 and p_2 cannot be placed into the same cluster.

For example, given the points p_1 , p_2, … , p_9 and these constraints:

ML(p_4, p_7)	**CL**(p_8, p_2)	**CL**(p_1, p_7)
ML(p_1, p_3)	**ML**(p_5, p_9)	**CL**(p_8, p_1)
CL(p_3, p_8)	**ML**(p_9, p_4)	**CL**(p_8, p_9)

One possible clustering would be to split the nodes as $\{p_1, p_3\}$, $\{p_2\}$, $\{p_4, p_5, p_7, p_9\}$, $\{p_8\}$.

In some cases, it is not possible to split the points into different sets while respecting the constraints. For example, given the constraints **ML**(x, y) and **CL**(x, y), there is no way to legally partition the data, since x and y simultaneously must be and must not be in the same set. Similarly, given **ML**(x, y), **ML**(x, z), and **CL**(x, z), no partition of the points is legal.

Design an algorithm that, given a collection of n points, m **ML** constraints, and c **CL** constraints, determines whether or not it is possible to cluster the points while satisfying all the constraints. Your algorithm doesn't need to *find* such a clustering; it just needs to report whether or not one exists. Your algorithm should run in time $O(m + n + c)$.

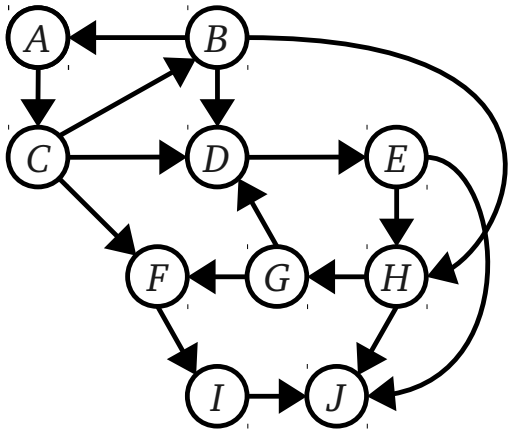
When answering this question, you should

- Describe your algorithm.
- Prove that your algorithm correctly determines whether a clustering exists.
- Prove that your algorithm runs in time $O(m + n + c)$.

For simplicity, you can assume that the points are numbered 0, 1, 2, …, $n - 1$ and that the **ML** and **CL** constraints are given to you in separate arrays.

Problem Four: Tournament Champions (12 Points)

Suppose that there is a tournament involving n players, where certain pairs of players play a single game against one another. We can then visualize the tournament as a directed graph, where each node is a player and there is an edge from u to v iff u played v and won her game against him.



For example, in the tournament to the right, player B directly beat A, D, and H, but lost to C; player G directly beat D and F, but lost to H; player C beat players B, D, and F, but lost to A; and player J won no games and lost to E, H, and I.

We'll say that a player p_1 transitively beat a player p_2 iff there is a path from p_1 to p_2 in the graph of the tournament. Intuitively, if p_1 and p_2 are different players, either p_1 directly beat p_2 in a game, or p_1 won against a player who in turn transitively beat p_2. Note that it's possible that two players transitively beat one another. In the above graph, player G transitively beat player H (since G beat D, who beat E, who beat H), but H also transitively beat G (since H directly beat G.)

Finally, define a *tournament champion* to be a player c who transitively beat every player in the tournament. A tournament can have multiple champions; for example, in the above tournament, players A, B, and C are champions, and no other players are. Some tournaments might not have any champions at all.

Let n be the number of players and m the total number of games played. Design an $O(m + n)$-time algorithm for finding all of the tournament champions in a tournament. If there aren't any winners, your algorithm should not output anyone. Then:

- Describe your algorithm.
- Prove that your algorithm is correct.
- Prove that your algorithm runs in time $O(m + n)$.

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?