Note: Your TA may not get to all the problems. This is totally fine, the discussion worksheets are not designed to be finished in an hour. The discussion worksheet is also a resource you can use to practice, reinforce, and build upon concepts discussed in lecture, readings, and the homework.

1 Shortest Paths with Negative Weights

(a) Dijkstra's algorithm doesn't work on graphs with negative edge weights. Here is one attempt to fix it:

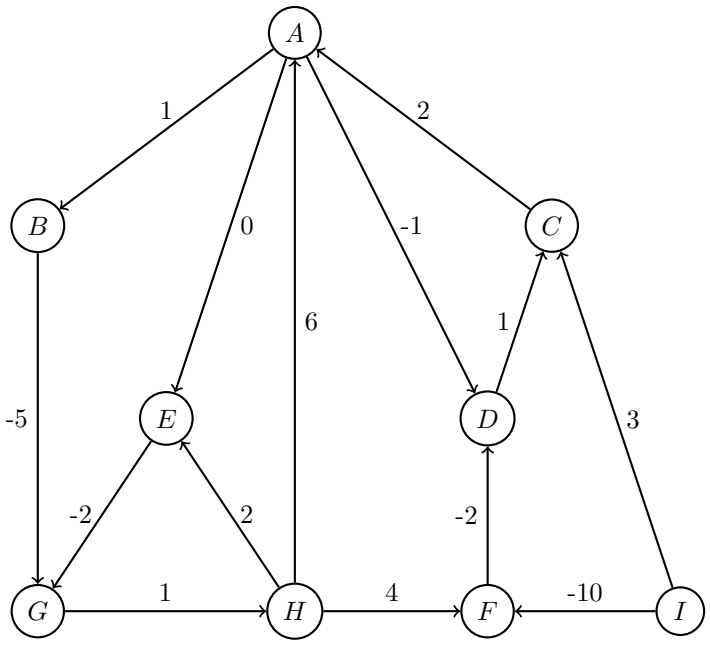
(a) Add a large number M to every edge so that there are no negative weights left.

(b) Run Dijkstra's to find the shortest path in the new graph.

(c) Return the path found by Dijkstra's, but with the old edge weights (i.e. subtract M from the weight of each edge).

Show that this algorithm doesn't work by finding a graph for which it must give the wrong answer.

(b) Run the Bellman-Ford algorithm on the following graph, from source A. Process edges (u, v) in lexicographic order, sorting first by u then by v.



(c) What problem occurs when we change the weight of edge (H, A) to 1? How can we detect this problem when running Bellman-Ford? Why does this work?

(d) Let $G = (V, E)$ be a directed graph. Under what condition does the Bellman-Ford algorithm returns the same shortest path tree (from source $s \in V$) regardless of the ordering on edges? (Hint: Think about how there could be multiple shortest path trees).

2 Service scheduling

A server has n customers waiting to be served. Customer i requires t_i minutes to be served. If, for example, the customers were served in the order $t_1, t_2, t_3, \ldots, t_n$, then the i-th customer would wait for $t_1 + t_2 + \cdots + t_{i-1}$ minutes. For simplicity, assume the t_i are distinct.

We want to minimize the total waiting time

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)\,.$$

For example, if there are three customers with waiting times $3, 5, 4$ minutes and they are served in that order, the total waiting time is 11 minutes: the first customer waits 0 minutes, the second waits 3 minutes, and the third waits 8 minutes.

(a) Given the list of the t_i's, give an efficient algorithm for computing the optimal order in which to serve the customers (no runtime/correctness needed).

(b) Consider any solution besides the one you found in part (a). Show how to swap two people in that solution's order to improve its cost.

(c) Conclude from the previous part that your solution is optimal.

3 Activity Selection

Assume there are n activities each with its own start time a_i and end time b_i such that $a_i < b_i$. All these activities share a common resource (think computers trying to use the same printer). A feasible schedule of the activities is one such that no two activities are using the common resource simultaneously. Mathematically, the time intervals are disjoint: $(a_i, b_i) \cap (a_j, b_j) = \emptyset$. The goal is to find a feasible schedule that maximizes the number of activities k.

Here are two potential greedy algorithms for the problem.

Algorithm A: Select the shortest-duration activity that doesn't conflict with those already selected until no more can be selected.

Algorithm B: Select the earliest-ending activity that doesn't conflict with those already selected until no more can be selected.

(a) Show that Algorithm A can fail to produce an optimal output.

(b) Show that Algorithm B will always produce an optimal output. (Hint: To prove correctness, show how to take any other solution S and repeatedly swap one of the activities used by Algorithm B into S while maintaining that S has no overlaps)

(c) **Challenge Problem:** Show that Algorithm A will always produce an output at least half as large as the optimal output.

4 Finding Counterexamples

In this problem, we give example greedy algorithms for various problems, and your goal is to find a counterexample where they do not find the best solution.

(a) In the travelling salesman problem, we have a weighted undirected graph $G(V, E)$ with all possible edges. Our goal is to find the cycle that visits all the vertices exactly once with minimum length.

One greedy algorithm is: Build the cycle starting from an arbitrary start point s, and initialize the set of visited vertices to just s. At each step, if we are currently at vertex u and our cycle has not visited all the vertices yet, add the shortest edge from u to an unvisited vertex v to the cycle, and then move to v and mark v as visited. Otherwise, add an edge from the current vertex to s to the cycle, and return the now complete cycle.

(b) In the maximum matching problem, we have an undirected graph $G(V, E)$ and our goal is to find the largest matching E' in E, i.e. the largest subset E' of E such that no two edges in E' share an endpoint.

One greedy algorithm is: While there is an edge $e = (u, v)$ in E such that neither u or v is already an endpoint of an edge in E', add any such edge to E'. (Challenge: Can you prove that this algorithm still finds a solution whose size is at least half the size of the best solution?)