(1) Scheduling Interviews II. In Problem 2 from problem set 6, we were assigning candidates to interviewers for a company without scheduling times for the interviews. In this problem, you are asked to set up a more detailed schedule. As in the previous problem, you are scheduling initial phone interviews for n job candidates at a company, with $k \leq n$ potential recruiters. Each candidate needs only one interview for now: concretely, each candidate needs to be matched to exactly one recruiter to be interviewed. The candidates are applying for different kinds of jobs, and recruiters are qualified to interview candidates only for some of the jobs. For each candidate i, you are given a list L_i of the recruiters who are qualified to interview that candidate. You also don't want to overload the recruiters, so each recruiter j has a limit q_j of the maximum number of interviews he or she can do.

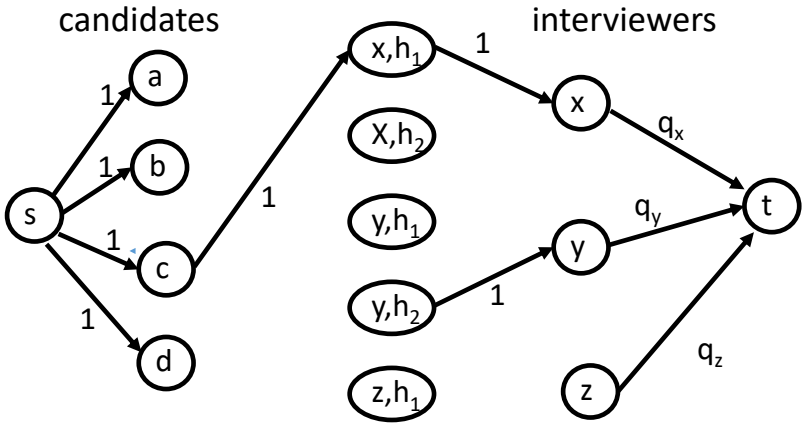
Each interview will take one hour, and will be scheduled in one of T possible non-overlapping one-hour time slots $h_1, \ldots h_T$, each starting at the top of the hour (e.g. 10 AM, 1 PM, 5 PM ...). In addition to the input above, each interviewer and each candidate is asked to list the hours that they would be able to interview or be interviewed. Please note that each interviewer is asked to list strictly more than q_j possible slots to make scheduling easier.

Give an algorithm that finds an interview schedule if one exists. Let $m = \sum_i |L_i|$. Your algorithm should run in time polynomial in n, k, m, and T.

Solution: Solve via maximum flow using Ford-Fulkerson. The solution is illustrated at the figure below. For nodes, we have s and t, a node for each candidate (n total), a node for each interviewer (k total), and a separate node (x, h) for each time h an interviewer x can make ($O(kT)$ total).

For edges, we have the following

- edges (s, c) for each candidate c with capacity 1.
- edges (x, t) for each interviewer x with capacity q_x
- edges $((x, h), x)$ for all nodes associated with a time slot h of interviewer x with capacity 1
- edges $(c, (x, h))$ if candidate c can be interviewed by interviewer x and is available at time h.



Finally, the assignment exists if the maximum flow has value n, and then each candidate c is assigned to interviewer x at time h if the edge $(c, (x, h))$ has flow 1.

The resulting graph has at most $N = 2 + n + k + Tk$ nodes, and at most $M \leq N^2$ edges (as is the case for any graph on N vertices); more precisely, we can also say $M \leq n + nT + kT + k$ edges. We have a capacity-1 edge leaving s for each of the n candidates, for a total of n capacity leaving s, so our min cut must be no larger than n, leading to a total running time of $O(n(n + Tk)^2)$: at each step, at

least 1 flow is added, so there are up to n steps, and in each step, finding an augmenting path takes $O(M) = O(N^2) = O((n+Tk)^2)$ time.

Alternate analyses: The capacities entering t are $C = \sum_j q_j$, where q_j denotes the capacity of interviewer j, so the running time is $O(CM)$. We can assume $q_j \leq T$ for all j (from the problem statement), and so $C \leq Tk$, and hence a total running time of $O(CM) = O(Tk(n+Tk)^2)$. A worse (but still polynomial, and thus allowed) running time can be obtained by reasoning that the capacity of the min cut cannot exceed the sum of all capacities on all edges. This gives $C \leq M + \sum_j q_j \leq M + Tk$ (since every edge that does not leave a recruiter has capacity 1, and we can also assume each edge leaving a recruiter j has capacity $q_j + 1$ since we are providing an upper bound), which yields the running time $O(CM) = O((M+Tk)M) = O((N^2+Tk)N^2) = O(((n+Tk)^2+Tk)(n+Tk)^2)$. Note that we could have also assumed each q_j that exceeds n is actually $\leq n$, since it is impossible to push more than n units of flow from the source into a recruiter node. We could then proceed with the above runtime analysis using the bound $q_j \leq n$ instead of $q_j \leq T$.

To prove that the solution is correct, we show that a max flow of value n is equivalent to a schedule in which every candidate is assigned a recruiter in a valid way. Note that the Ford-Fulkerson algorithm finds an integer valued flow, so the max flow found is integer valued. If the total flow has value n, then each (s, c) must have flow 1, and exactly one of the edges leaving c will have flow. This edge with flow arrives at a node that describes the recruiter who will interview that candidate, and the time the interview will occur. Since the edges $((x, h), x)$ have capacity 1, at most one candidate can be scheduled with x at time h. Since the edges (x, t) have capacity q_x, interviewer x will conduct at most q_x interviews.

Conversely, note that if there is a schedule that validly assigns all candidates to be interviewed, then the max flow in the constructed network has value n: for each candidate, simply send 1 unit of flow from source to sink along the path that starts at the source, goes to the candidate, goes to the time/recruiter of the candidate's interview, goes to the recruiter of the interview, and finally ends at the sink. That is, put 1 unit of flow on edges $(s, c), (c, (x, h))$, and $((x, h), x)$ whenever x interviews c at time h. Finally, add z units of flow on edge (x, t), where z is the number of candidates that recruiter x interviews.

Common mistakes:

- Failing to include a proper running time analysis. For example, claiming that FF runs in $O(E)$ time with E is the number of edges. More commonly, some students knew the runtime was $O(MC)$, but didn't give a bound on M or C using the requested variables (that is, do not explain why M and C are polynomial in the size of the input description of the problem.)
- Many students don't point out the integrality property of Ford-Fulkerson, which is necessary in a correctness argument (i.e., in constructing a schedule from a flow).
- Some students did not reduce to max flow or circulation, and instead tried to do pairwise swapping to find an assignment. This will not be viable for large instances.

(2) Monitoring nodes in a graph. Consider a directed network $G = (V, E)$ that represents a transportation network in a war-torn region, where some edges of the network are not always reliable. The node $h \in V$ represents the location of the headquarters for an important strategic operation. In addition, there is a set of outposts or *terminals* $T \subseteq V$ that the headquarters will need to send supplies to. You want to know how vulnerable the operation is to disruption. For each terminal $v \in T$, you have a value $w_v \geq 0$ of the importance of being able to reach v from the headquarters. You wonder how easily the enemy could damage the supply network by disconnecting a few specific edges.

We make this precise as follows. Deleting a set of edges $F \subseteq E$, some nodes in T will no longer be reachable from h, i.e., these are the nodes such that there is no h-v path in the subgraph $(V, E - F)$. We will call such terminals *disconnected by* F, and let $q(F)$ denote the total value of the terminals $v \in T$ disconnected by F. In other words, $q(F)$ is value unreachable terminals in T when losing access to the edges F.

Give a polynomial-time algorithm to find a set F of edges that maximizes the quantity $q(F) - |F|$. (Note that setting F equal to the empty set is an option, so this maximum will be nonnegative.) Update 10/20: you may assume that the values w_v are non-negative integers, and your algorithm may also be polynomial in the values of the terminals.

Solution: Add a sink t with edges (v, t) of capacity w_v for each $v \in T$. Have all other edges have capacity 1, let $s = h$ be the source, and find a (s, t) min cut G. Now let $Q = \sum_{v \in T} w_v$. Consider any (s, t) cut (A, B). Let F be the set of original edges cut by this cut (not counting the newly added edges to t.

We claim that $c(A, B) \geq |F| + (Q - q(F))$. To see why this is true, notice $|F|$ is the capacity of the edges cut of the original graph. Now consider the edges entering t. The set of nodes on the A side of the cut must contain all the nodes reachable from h after cutting F. The total capacity of these edges is $Q - q(F)$, and the capacity of these edges all contribute to the capacity of the cut.

If we let A' be all the nodes reachable in G from h after removing F, and $B' = V \setminus A'$, then the cut (A', B') will have capacity $c(A', B') = |F| + (Q - q(F))$. If A had any additional terminals, than $c(A', B') < c(A, B)$, which would mean we do not have a min (A, B) cut. So, for a min cut, we must have the equation above: $c(A, B) = |F| + (Q - q(F))$, where A is the set of nodes reachable by h in G given the edges in F were cut.

Rearranging the above equation as $Q - c(A, B) = q(F) - |F|$, shows that the minimum cut maximizes the quantity $q(F) - |F|$, and we get this maximum value as the difference $Q - c(A, B)$.

Common mistakes:

- A wrong algorithm that is closest to correct solution is to run an (h, v) min-cut algorithm for each terminal $v \in T$, and greedily disconnect the terminals. That is, for each terminal v in turn, if at most w_v edges are needed to disconnect v do so. This is effectively a greedy algorithm. It is possible that none of the individual terminals can be disconnected by w_v or fewer edges, but we can disconnect a lot of them together by using just a few more than $\max_x w_v$.
- There are a number of exponential time solutions. For example: try all subsets of edges, compute $q(F) - |F|$ for each F and take the maximum.

(3) Reduction. Suppose you have a subway map that you can represent as an undirected graph $G = (V, E)$ of stops, with n different subway lines described as paths through the graph. Assume no more than one train is scheduled to be at a stop at a time. (An edge can belong to one or more subway lines.)

One day a week, you want someone to quickly clean the front window of every train. To do this, you can hire somebody to stay at a particular train stop $v \in V$ and every time a train is at that stop, they will clean the front window of it, but will not board. You want to ensure that every train will, at some point, be at a stop with one of those people. However, you only have the budget to hire k window cleaners to occupy k distinct stops. You would like to find out if it is possible to use only k window cleaners to reach all the trains. We call this the WindowCleaner Problem.

Show that the WindowCleaner Problem is at least as hard as one of the two hard problems we considered in class on Friday, Oct 19th: the Vertex Cover problem or the Independent Set problem. (This coming week, we'll show that both of these are NP-complete.)

Solution: We show that Vertex Cover is no harder than the WindowCleaner Problem, that is Vertex Cover $\leq$ WindowCleaner Problem. To do this, we have to do a reduction turning Vertex Cover problems into an equivalent WindowCleaner Problem. Before we do so, notice that this is the natural problem choice: Independent Set is a problem that gets easier as k gets smaller, and Vertex Cover and the WindowCleaner Problem both gets harder as k gets smaller, so they are more similar.

Consider an undirected graph $G = (V, E)$ and an integer k as an instance of the Vertex Cover problem. We use G as our subway map. Each node is a subway station, and we'll have very short

subway lines: each edge is a separate subway line. (This reduction lets us reuse the original graph and k, so we consider it a constant-time reduction.) We claim that the resulting WindowCleaner Problem is equivalent to the original Vertex Cover problem. We need to show two directions.

- If the WindowCleaner Problem has a solution, let A be the set of k stations that allow all train windows to be cleaned. We claim that A is a vertex cover: if there were any edges $e = (v, w)$ not covered by A, then the windows on the subway line between v and w won't be cleaned.
- If the Vertex Cover problem has a solution $A \subset V$, then posting a window cleaner at each of the stations at A solves the WindowCleaner Problem; all subway line windows will cleaned, as subway lines correspond to edges, and all edges are covered by A.

Mistake of reducing the wrong way: An important mistake to avoid is reducing the wrong way. If we take an instance of the WindowCleaner Problem, and turn it into Vertex Cover that would show that Vertex Cover is at least as hard as the WindowCleaner Problem. This is the wrong direction! Our goal is to show that WindowCleaner Problem is hard.

In fact, the above reduction really shows that Vertex Cover is a special case of the WindowCleaner Problem, the special case when all subway lines go back and forth between two stops without having any intermediate stops. In this pair of problems, reduction in the wrong direction is much harder to show! (though that is possible also.)