Solutions to Final Exam Practice Problems

Solution 1:

(a) By the recursive nature of DFS, descendants are discovered later and finished earlier. Therefore, if u is a descendant of v then $d[v] < d[u] < f[u] < f[v]$.

(b) LCS($\langle ababa \rangle, \langle babab \rangle$) = $\langle abab \rangle$ or $\langle baba \rangle$ (either is acceptable).

(c) In the *k-center problem* we are given a point set P, and the objective is to identify a subset $C \subseteq P$ of size k so as to minimize the maximum distance of every point of P to its closest point in C. A factor-2 approximation algorithm outputs a set of k centers such that the maximum distance of any point to its closest center is at most twice as high as the maximum such distance in the optimal solution.

(d) (i) False: The maximum flow is limited by the minimum cut. If we double all the capacities, the capacity of *this* cut doubles, but there may be a different cut with a smaller capacity.

(ii) True: If we half the capacities of the edges that cross the minimum cut, then (X, Y) is still the minimum cut, and hence the maximum flow is equal to its new capacity, which is half of the original.

(e) True. One easy way to see this is to divide all the edge capacities by 3. They are now all integers, and we know that an integer flow can be computed. Then, multiply all the flow values by a factor of 3. The resulting flow values all satisfy the original capacity constraints and are all multiples of 3.

(f) True. The key is that 100 is a "constant," since it does not depend on n. The number of subsets of size 100 among n vertices is $\binom{n}{100} \leq n^{100}$, which is a polynomial in n. We could simply enumerate them all, and check whether each subset is an independent set. (If the number 100 was changed to something that could increase with n, e.g., $\lfloor \sqrt{n} \rfloor$, then the answer would be "False".)

Solution 2: The basis case ($i = 0$ or $j = 0$) are the same (the LCS length is zero), so we will assume that both i and j are positive.

(a) (LCS with wild cards) If either x_i or y_j is equal to "?", but not both, we may match this wild card with the last character of the other string, increasing the length of the LCS by 1. In fact, there is not advantage to be gained by forgoing this match. If both x_i and y_j are equal to "?", they cannot be matched together, so we know one of them will be skipped. We try both and take the better of the two results. This is the same as if the characters did not match.

$$\text{lcs}(i,j) = \begin{cases} \text{lcs}(i-1, j-1) + 1 & \begin{cases} \text{if } x_i = y_j \neq \text{"?"} \\ \text{or either } x_i \text{ or } y_j \text{ (but not both) equals "?"} \end{cases} \\ \max(\text{lcs}(i-1, j),\ \text{lcs}(i, j-1)) & \text{otherwise.} \end{cases}$$

(b) (LCS with swaps) We add an additional rule to the standard LCS formulation. If $i, j \geq 2$, and the last two characters of X_i and Y_j are swaps of each other, that is, $\langle x_{i-1}x_i \rangle = \langle y_j y_{j-1} \rangle$, then we add both characters to the LCS and recurse on X_{i-2} and Y_{j-2}.

$$\text{lcs}(i,j) \;=\; \begin{cases} \text{lcs}(i-2, j-2) + 2 & \text{if } i,j \geq 2 \text{ and } \langle x_{i-1}x_i \rangle = \langle y_j y_{j-1} \rangle, \\ \text{lcs}(i-1, j-1) + 1 & \text{otherwise, if } x_i = y_j, \\ \max(\text{lcs}(i-1, j),\ \text{lcs}(i, j-1)) & \text{otherwise.} \end{cases}$$

To establish correctness, we assert that, if the last two characters form a swapped matched, we may as well match them. The reason is that forgoing this match could not result in a longer LCS.

Solution 3: We reduce this to a circulation problem involving lower and upper flow bounds. Given the n pharmacists and the m days of the month, we create a network as follows. First, we create a source vertex s and a sink vertex t. We create n pharamacist vertices and m day vertices. We join s to each of the pharmacist vertices. If this pharmacist lists d available days, we set the lower-upper bounds for this edge to be $[\lceil d/2 \rceil, d]$. Next, we join each pharmacist vertex to each of the day vertices corresponding to the days that this pharmacist is available to work. We set the lower-upper bound on these edges to $[0, 1]$. Finally, we add an edge between each day vertex and t and set its lower-upper bounds to $[3, 3]$. Because each day must have exactly three pharmacists working, we know that the total demand of pharmacists over the entire month is $3m$. Therefore, we set the demand at s to be $-3m$ (meaning that it must supply $3m$ units of flow) and we set the demand at t to be $3m$ (meaning that $3m$ units of flow must arrive here). We set the demands for all the pharmacist and day vertices to zero (see Fig. 1(a)). (By the way, an alternative solution would be to create an edge from t to s with capacity $[3m, 3m]$ and then set the demands at s and t both to zero.)

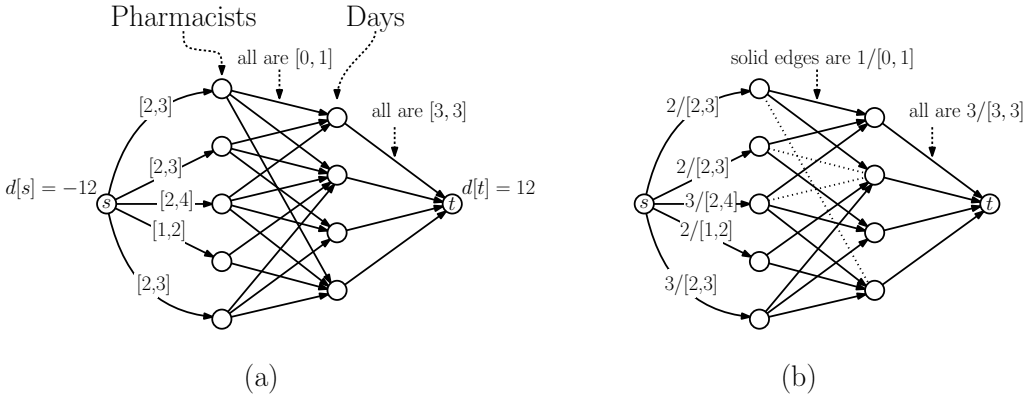

Figure 1: Pharmacist assignment.

We pass the resulting network to any circulation algorithm. We claim that a feasible schedule exists if and only if a feasible circulation exists. Suppose that there is a feasible schedule. Suppose that pharmacist i is scheduled to work on day j. We route one unit of flow along the associated edge (see Fig. 1(b)). If this pharmacist is working d_i total days, we route d_i units of flow from s to i. Because the schedule is valid, each pharmacist is working from $\lceil d/2 \rceil$ up to d days, and so this

flow satisfies the lower and upper flow bounds. Finally, because the schedule is valid, there are 3 pharmacists working each day. Therefore, there are 3 units of flow entering each day vertex, and we route these 3 units to t. Since there are $3m$ units of flow entering t, its demand is satisfied. Similarly, there are $3m$ units being supplied by s, and so its demands are satisfied. Therefore, this is a valid flow.

Conversely, suppose that there is a valid flow in the network. We may assume that it is an integer flow. If there is a flow of one unit on the edge between pharmacist i and day j, we schedule this pharmacist to work on that day. Observe that pharmacists are only asked to work on days when they are available. Since the lower and upper flow bounds from s to i are satisfied, each pharmacist is working the required number of days. Finally, because the flow from each day j to t must be exactly 3, there will be 3 pharmacists working each day. Therefore, this is a valid work schedule.

Solution 4: We will do this by reduction to network flow. (It is also possible to reduce to the circulation problem.) Create an s-t network $G = (V, E)$, where $V = U \cup C \cup \{s\} \cup \{t\}$. Create a unit capacity edge (u_i, c_j) if cell-phone user $u_i \in U$ is within distance Δ of cell-phone tower $c_j \in C$. Create an edge from s to each $u_i \in U$ with capacity 3, and create an edge from $c_j \in C$ to t with capacity m_j.

Compute the maximum (integer) flow in this network. We claim that a feasible schedule exists if and only if all the edges exiting s are saturated in the maximum flow. To prove the "only if" part, we consider the following mapping between any valid assignment and a flow in G. We first observe that if there is a valid assignment, there is a valid assignment in which each user is assigned to exactly three towers (since if it was assigned to more than three, removing the additional assignments cannot violated any of the constraints of the problem). If user u_i is assigned to tower c_j, then $\text{dist}(u_i, c_j) \leq \Delta$, and so an edge exists between them. We apply one unit of flow along the edge (u_i, c_j). Since u_i is assigned to three towers, it has three units of flow entering it, and so the incoming edge (s, u_i) is saturated. Finally, since this is a valid assignment, the number of users assigned to any tower c_j is at most m_j, and so the flow leaving each c_j is at most m_j.

To prove the "if" part, suppose that G has a flow that saturates all the edges coming out of s. We show how to map this to a valid schedule. Because of the edge (s, u_i) is saturated, and the outgoing edges are of unit capacity, each user has three outgoing edges carrying flow. We assign u_i to these three towers. Since edges are created only when users and towers are within distance Δ, these are valid assignments. Since the flow coming out of tower j is at most m_j, tower j is assigned to more than m_j users. Thus, this is a valid assignment.

The running time of the algorithm is dominated by the time for the network flow, which is $O(V^3)$, where $|V| = m + n$.

Solution 5: Create an s-t network by making the root node r the source and creating a super-sink node t, which will have edges coming from all the terminal nodes $t_i \in T$. Observe that if $C = (X, Y)$ is any cut of the resulting s-t network, then removing the edges that cross the cut from X to Y separates r from all the terminals. Therefore, this problem is equivalent to computing a minimum weight cut in this network.

We can reduce this to a network flow problem. First, set all the capacities of all the nodes of the network to 1 except the edges that enter t to capacity ∞, or more practically, any numeric value that is larger than the maximum possible flow (see Fig. 2). Now, run any network flow algorithm on the resulting network. Let f denote the resulting flow.

We compute the minimum cut as follows. First, construct the residual graph G_f, and determine the subset of nodes X that are reachable from r, and let $Y = V \setminus X$ be the remaining nodes. Note that none of the nodes of T can be reachable from r in G_f, for otherwise we could push more flow through this terminal into t. Therefore, $T \subseteq Y$. By the max-flow/min-cut theorem, the edges (x, y) of G that cross the cut (that is, where $x \in X$ and $y \in Y$) define the minimum cut in G. Since these edges are of capacity 1, the capacity of this cut is equal to the number of edges in the cut. Therefore, this is the minimum number of edges needed to separate r from all the vertices of T.

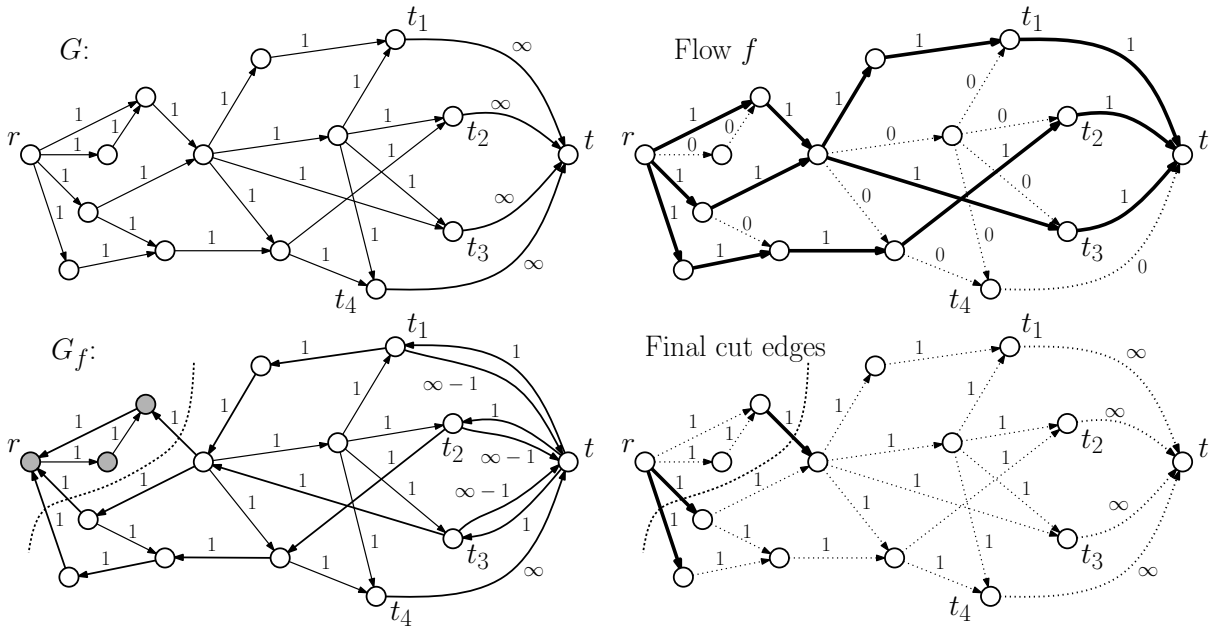

Figure 2: Eliminating edges to separated r from terminals.

Solution 6:

(a) To show that HDIS is in NP, the certificate consists a subset V' of k vertices of G to be the HDIS. We verify by checking that no pair $(u, v) \in V'$ are adjacent and that the degree of each vertex of V' is at least as large as k. If so, we output "yes," and otherwise we output "no."If G has an HDIS, then one of the guesses will succeed, yielding a global answer of "yes." If G has not HDIS, all the guesses will fail, yielding a global answer of "no."

(b) We show that IS $\leq_P$ HDIS. Given an input to the independent set problem, (G, k), where G is an undirected graph and k is an integer. Since we do not know which vertices of G are in the independent set (or even whether an independent set exists), our approach will be to add a bunch of bogus vertices whose job it is to make the degree of *every* vertex at least as high as k, but to do so in such a way that does not alter the size of the independent set.

First make a copy of G and add k new vertices. Join all the new vertices together into a k-clique, and for each vertex in the copy of G, add k edges joining it to every vertex in this clique. Let G' denote the resulting graph and set $k' = k$. Output (G', k') (see Fig. 3). Clearly, this can be done in polynomial time, and in fact in time $O(n + m + kn)$.

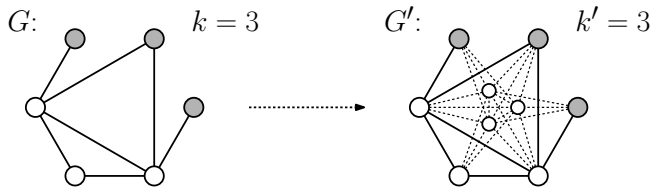

Figure 3: Reducing Independent Set to High-Degree Independent Set.

Notice that G' has $n' = n + k$ vertices, and each vertex of G' has degree at least k (because it is adjacent to all the newly added vertices in the clique). To establish correctness, we assert that G has an independent set of size k if and only if G' has an HDIS of size k'.

- ($\Rightarrow$) If V' is an IS for G, then the corresponding vertices of G' are not adjacent to each other, but they all have degree at least k (since every vertex in G does). Thus, V' is an HDIS in G'.
- ($\Leftarrow$) Conversely, suppose that V' is an HDIS in G'. We may assume that $k \geq 2$, since otherwise the problem is trivial. We claim that there can be no vertices in G' from the clique (excluding the trivial case where $k = 1$). The reason is that these vertices are adjacent to every other vertex in G'. Therefore the vertices of V' come from the original vertices of G, implying that V' forms an independent set in G.

Solution 7: We first show that B3C is in NP. The certificate is the assignment of colors to the vertices. In polynomial time we can check that the each pair of adjacent vertices have different colors, and we can count the number of vertices of each color and verify that each color is used $|V|/3$ times. If so, we output "yes," and otherwise "no." If G has a balanced 3-coloring, one of the guesses will succeed, and global answer is "yes." Otherwise, all the guesses will fail, and the global answer is "no."

Second, we show that B3C is NP-hard by showing that the standard 3-coloring problem (3Col) is reducible to it (3Col $\leq_P$ B3C). Let $G = (V, E)$ be an instance of 3Col. We want to produce an equivalent instance of B3C. The trick is to add vertices whose colors can be assigned arbitrarily in order to guarantee that the sizes of the color classes is balanced. Let $n = |V|$. Let G' consist of a copy of G together with $2n$ additional vertices with no edges attached to them. Observe that G' has $3n$ vertices exactly. Each of these additional vertices is isolated in the sense that it is not adjacent to any other vertex. Clearly G' can be produced from G in polynomial time.

We claim that G is 3-colorable if and only if G' can be 3-colored with colors each occurring equally often.

- ($\Rightarrow$) Suppose that G is 3-colorable. Let k'_1, k'_2, k'_3 be the number of times each of the colors appears. We have $k'_1 + k'_2 + k'_3 = n$. Because they are isolated, we can color $n - k'_i$ of the isolated vertices with color i. The total number of vertices with color i is thus $k'_i + (n - k'_i) = n$. Since G' has $3n$ vertices, each color is used equally often.
- ($\Leftarrow$) Suppose that G' can be 3-colored with balanced color groups. Then, if we discard the $2n$ newly added vertices, this is a coloring for the remaining graph, namely G.

Solution 8:

(a) To show that HP is in NP, the certificate consists of the sequence of n vertices along the path, $\langle u_1, u_2, \ldots, u_n \rangle$. In polynomial time we can verify (a) that the path is simple in that no vertex is repeated and (b) that this is a valid path in that (u_i, u_{i+1}) is an edge for each i. As usual, if so, we output "yes," and otherwise "no." If G has a Hamiltonian Path, the verification succeeds for at least one of the guesses, yielding a global answer of "yes." If not, all the guesses fail, for the global answer of "no."

(b) Prof. Farnsworth's trivial reduction from HC to HP satisfies two out of three of the criteria for being a valid reduction. It runs in polynomial time, and if G has a Hamiltonian Cycle, then G has a Hamiltonian Path. However, the converse fails. It is possible that G has a Hamiltonian Path but no Hamiltonian Cycle. Thus, this is not a correct reduction.

(c) The (correct) reduction function works as follows. Let $G = (V, E)$ be the input for the HC problem. Let u be any vertex of G. We know that if G has a Hamiltonian Cycle, then this cycle must pass through u. The idea is to add a gadget that will effectively "split" u into two vertices that will serve as the beginning and end of a Hamiltonian Path. We must do this in a way that does not introduce any new Hamiltonian Paths in the modified graph, other than those that came from valid Hamiltonian Cycles in the original graph.

We replace u with two new vertices, u' and u''. We attach both u' and u'' to all the neighbors of u in the original graph. In order to "force" the path to start and end at these vertices, we create two new vertices w' and w'' and add the edges (w', u') and (w'', u'') (see Fig. 4(a)). Let's call the resulting graph G'.

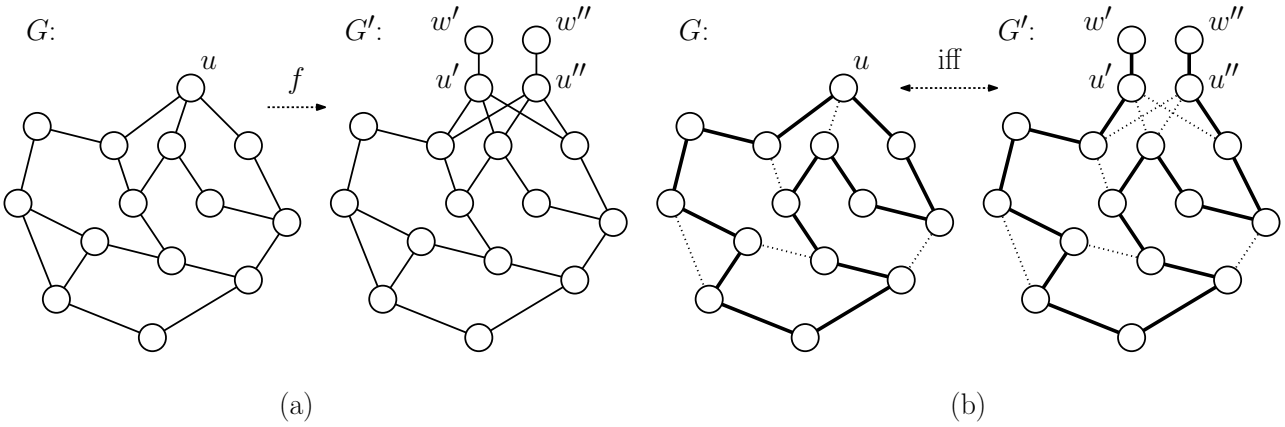

Figure 4: HC to HP reduction.

To establish the correctness of the reduction, we show that G has a Hamiltonian Cycle if and only if G' has a Hamiltonian Path. Suppose that G has a Hamiltonian Cycle $\langle u_1, \ldots, u_n \rangle$. Since we can start the cycle wherever we like, we may assume that $u_1 = u$ in our construction. It is easy by our construction that $\langle w', u', u_2, \ldots, u_n, u'', w'' \rangle$ is a Hamiltonian Path in G' (see Fig. 4(b)).

Conversely, suppose that G' has a Hamiltonian Path. The path must start and end at w' and w'', because these vertices have degree 1. (By the way, this is the reason for adding these two

nodes. Without them, it is conceivable that there is a Hamiltonian Path that does not begin and end at u' and u'', and this would be trouble.) Therefore, the path must have the form $\langle w', u', u_2, \ldots, u_n, u'', w'' \rangle$ (or its reverse), for some sequence $u_2, \ldots, u_n$ that forms a simple path in G'. We can convert this into a cycle in G by removing w' and w'' and replacing u' and u'' with the single node u. Therefore, G has a Hamiltonian Cycle.