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: Even though we do not know whether P = NP, we do know that P $\subseteq$ NP. The reason is that any problem in P can be solved in polynomial time, and therefore it trivially can be verified in polynomial time (by just solving it).

(g) 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".)

(h) (iii). Reductions do not preserve approximation bounds, and we cannot infer anything about A's approximability.

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.

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

The final answer is $\text{lcs}(m,n)$.

(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}. (While I believe that it can be proved that it is always safe to take such a swapped match whenever it emerges, just to be safe, we will simply consider this among the possible options and take the best of all of them.)

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

The final answer is $\text{lcs}(m,n)$.

Solution 3: 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 $c_j \in C$ (see Fig. 1). 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 tower assingment exists if and only if all the edges exiting s are saturated in the maximum flow. The running time of the algorithm is dominated by the time for the network flow.

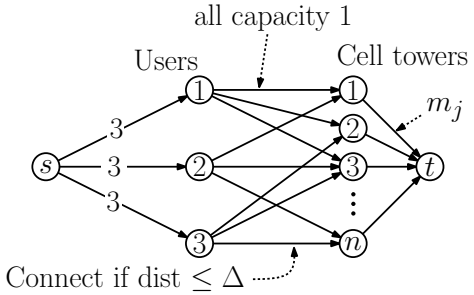

Figure 1: Cell phone tower assignment.

Claim: A feasible tower assignment exists if and only if all the edges exiting s are saturated in the maximum flow.

Proof: ($\Rightarrow$) Suppose there is a valid tower assignment. We may assume without loss of generality that every user is assigned to exactly three towers (since we can always assign users to fewer

towers). If user u_i is assigned to tower c_j, then $\text{dist}(u_i, c_j) \leq \Delta$, and so they are connected by an edge in G. We apply one unit of flow along the edge (u_i, c_j), three units along edge (s, u_i), implying this edge is saturated. Assign (c_j, t) a flow equal to the number of users assigned to this tower. Since this is a valid assignment, the number of users assigned to any tower c_j is at most m_j, and so this flow satisfies the capacity constraints. Clearly, flow conservation is satisfied.

($\Leftarrow$) Suppose that G has a flow that saturates all the edges coming out of s. We may assume that this is an integer-valued flow. 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.

Solution 4: 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 (see Fig. 2(a)). 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 $[3, 3]$.

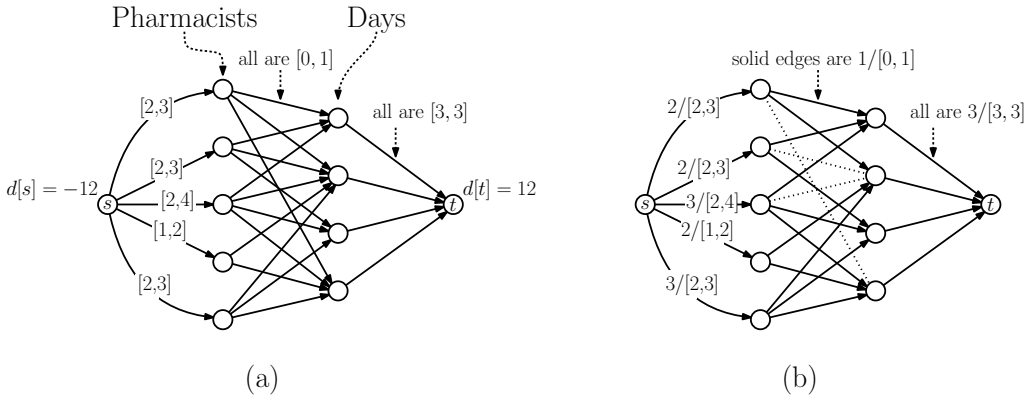

Figure 2: Pharmacist assignment.

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. (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.)

We pass the resulting network to any circulation algorithm. If there is a valid circulation, we report that a schedule exists.

Claim: A feasible schedule exists if and only if a feasible circulation exists.

Proof: ($\Rightarrow$) Suppose that there is a feasible schedule. If pharmacist i is scheduled to work on day j, we route one unit of flow along the associated edge (see Fig. 2(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 three pharmacists working each day. Therefore, there are three units of flow entering each day vertex, and we route these three 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.

($\Leftarrow$) Conversely, suppose that there is a valid circulation in the network. We may assume that it is integer-valued. 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, because these are the only edges present in the network. 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 three, there will be three pharmacists working each day. Therefore, this is a valid work schedule.

Solution 5:

(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 $O(n + m + kn)$.

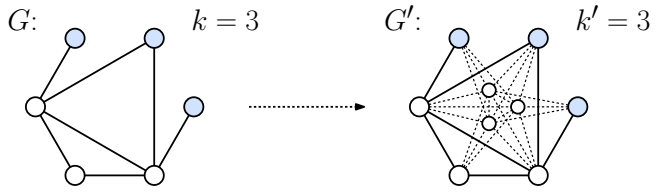

Figure 3: Reducing Independent Set to High-Degree Independent Set.

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 6: 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 7: To prove that the SqIS problem is NP-complete, we need to show the following.

SqIS $\in$ NP: Given a graph $G = (V, E)$ and integer k, we guess a subset V' of k vertices of G. We compute the degrees of all these vertices and determine whether they are all are perfect squares. We also check whether for each pair $u, v \in V'$, $(u, v) \notin E$. If so, we report "yes," and otherwise "no." If G has a SqIS, one of these guesses will succeed, and the global answer is "yes." Otherwise, they will all fail, and the overall answer is "no."

SqIS is NP-hard: We will show that the standard independent-set problem is polynomially reducible to SqIS (IS $\leq_P$ SqIS). The idea is to artificially increase the degree of every vertex of G to be a perfect square. We need to take care that we do not create any new independent sets.

Suppose that we are given an input graph $G = (V, E)$ and integer k for the independent-set problem (see Fig. 4(a)). We define a *lollipop* to be a collection of three vertices, joined to form a triangle (the candy), with an additional edge coming out of any one of the three (the stick). Observe that the lollipop vertices have degrees 2 and 3, which are not perfect squares. For each vertex $v \in V$, let $d(v)$ denote its degree, and let $p(v) = \left\lceil \sqrt{d(v)} \right\rceil^2$ be the smallest perfect square that is greater than or equal to $d(v)$. We join v to the end of the sticks of $p(v) - d(v)$ lollipop structures (see Fig. 4). Let G' be the resulting graph. We output (G', k) (see Fig. 4(b)).

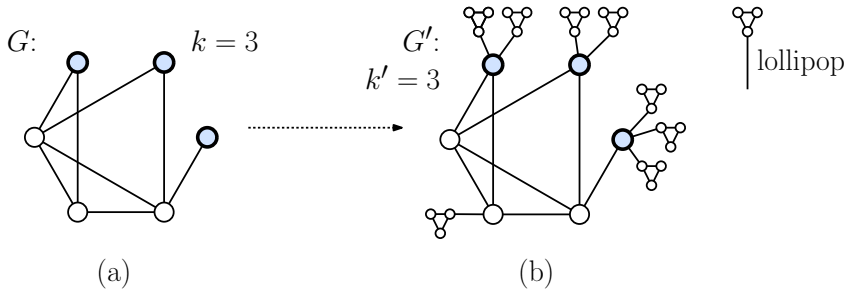

Figure 4: IS to SqIS reduction.

Observe that each of the original vertices of G now has degree $p(v)$ in G', which is a perfect square, and all the remaining (lollipop) vertices have degrees that are not perfect squares. The number of lollipop structures added to any vertex is $O(n^2)$, and hence the total size of G' is at most $O(n^3)$, which is a polynomial. To establish correctness, we will show that G has an IS of size k if and only if G' has an SqIS of size k.

($\Rightarrow$) Suppose that G has an independent set V' of size k. Clearly, the same vertices form an independent set within G'. By construction, every vertex of V in the new graph has a degree which is a perfect square. Therefore, these vertices form an SqIS of size k in G'.

($\Leftarrow$) Suppose that G' has an SqIS V' of size k. We assert that the vertices of the lollipop structures cannot contribute to the SqIS, because their degrees are not perfect squares. Therefore, all the vertices of V' are part of the original graph G, and hence they form an independent set in G, as desired.

Note: A common (but incorrect) solution is to attach each vertex to a clique of size $p(v)-d(v)$. This is problematic if this number happens to be a perfect square. For example, if $d(v) = 21$ and $p(v) = 25$, this connects v to a clique of size 4, but then each of these vertices has degree 4, which is a perfect square. If v does not contribute to the SqIS, then one of the vertices of the attached clique could be included in the SqIS. Fixing this bug is a bit tricky, but it is possible. Our lollipop reduction avoids this complication.

Solution 8:

(a) Let C^* denote the cost of the optimum bottleneck TSP tour, and let C' be the cost of the alternating TSP tour. Let ℓ denote the maximum distance between any two consecutive points, that is $\ell = \max_{1<i\leq n} |x_i - x_{i-1}|$. Observe that *any* TSP tour must make at least one hop of length at least ℓ, therefore $C^* \geq \ell$. On the other hand, the alternating heuristic never makes a hop of length greater than 2ℓ, since this is the maximum distance between two alternating points. Therefore $C' \leq 2\ell$. Combining these, we have $C' \leq 2\ell \leq 2C^*$, which implies that the alternating heuristic is a factor-2 approximation to bottleneck TSP on the line.

(b) Consider any path that is not alternating. Find the first instance where the alternating pattern is violated. One of the paths (outbound or inbound) must hit a consecutive pair of points. Assume, for the sake of concreteness that it is outbound path. This implies that the inbound path must skip both. This implies that there are four points $x_a < x_b < x_c < x_d$, such that the outbound path has the edge (x_b, x_c) and the inbound path has the edge (x_d, x_a) (see Fig. 5). To simplify the proof, let's assume that these points are consecutive.

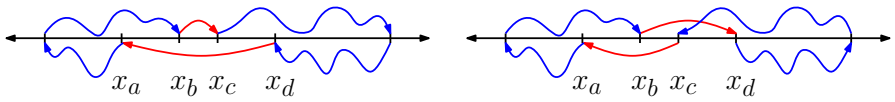

Figure 5: Alternating heuristic for the bottleneck TSP tour.

Let's swap these edges in both paths. The outbound path now takes the edge (x_b, x_d) and the incoming path takes the edge (x_c, x_a). To fix the rest of the tour, let's reverse the portion of the tour from x_c to x_d. This is a valid tour, and except for the two swapped edges, the cost of the tour has not changed. How does the bottleneck cost change as a result? Other than the reversal (which does not alter the cost), the only change arises by replacing the $\max(|x_c - x_b|, |x_d - x_a|)$ with $\max(|x_c - x_a|, |x_d - x_b|)$. We assert that this cannot increase the overall bottleneck cost. This is because, due to the ordering of the points, we have

$$\max(|x_c - x_b|, |x_d - x_a|) \;=\; |x_d - x_a| \;\geq\; \max(|x_c - x_a|, |x_d - x_b|).$$

By repeatedly applying this swapping operation wherever the alternating pattern is violated, we eventually convert any tour into the alternating path, while never increasing its cost. This implies that the alternating heuristic is optimal.