Problem Set 4 Solutions

This problem set is due **at 11:59pm** on **Thursday, October 6, 2016.**

EXERCISES (NOT TO BE TURNED IN)

Network Flow / Matching

- Do Exercise 26.3-4 in CLRS on Page 735.
- Do Exercise 26.3-5 in CLRS on Page 735.

All Pairs Shortest Paths

- Do Exercise 25.1-5 in CLRS on page 692.
- Do Exercise 25.2-4 in CLRS on page 699.
- Do Exercise 25.2-6 in CLRS on page 700.

Dynamic Programming

- Do Exercise 15.1-4 in CLRS on page 370.
- Do Exercise 15.2-4 in CLRS on page 378.
- Do Exercise 15.4-5 in CLRS on page 397.

Problem 4-1. Favorite Numbers [50 points]

There are n students in 6.046, and the i^{th} student's favorite number is a positive integer f_i. Two students are friends if and only if their favorite numbers differ by exactly a multiplicative factor of 2, 3, or 5. That is, students i and j are friends if and only if f_i/f_j is in the set $\{2, 3, 5, 1/2, 1/3, 1/5\}$.

(a) [20 points] Let G be a graph with a vertex for each student and an edge between any two students who are friends. Show that G is bipartite.

Solution: Using prime factorization, express each student's favorite number in the form $2^a 3^b 5^c k$, where a, b, and c are positive integers and k is a positive integer not divisible by 2, 3 or 5. Any two students who are friends will have $(a + b + c)$ values that differ exactly by 1. This means that for any two friends, their values of $(a+b+c)$ cannot be the same parity (both even or both odd). We can divide the students into two groups based on the parity of their $(a + b + c)$ values. Edges will never connect two vertices in the same group, making G a bipartite graph.

Alternative solution: Any graph with no odd-length cycles is a bipartite graph. We will show that G does not contain any odd-length cycles. Suppose that an odd cycle does exist. Let f_1, f_2, ..., f_m be the favorite numbers on that cycle. Let $d_i = f_{i+1}/f_i$ and $d_m = f_1/f_m$. All the d_i values must be in the set $\{2^1, 3^1, 5^1, 2^{-1}, 3^{-1}, 5^{-1}\}$.

$$
\begin{aligned}
f_2 &= d_1 f_1 \\
f_3 &= d_2 f_2 = d_2 d_1 f_1 \\
&\vdots \\
f_m &= f_1 \cdot \prod_{i=1}^{m-1} d_i \\
f_1 &= f_1 \cdot \prod_{i=1}^{m} d_i
\end{aligned}
$$

The last equation tells us the product of all the d_i values (let's call it D) must equal 1. 2^1 must appear in the sequence of d_i's the same number of times that 2^{-1} appears, because otherwise they wouldn't cancel out and D would have a nonzero power of 2 in its prime factorization. The same goes for 3^1 and 3^{-1}, and for 5^1 and 5^{-1}. There must be an even number of d_i values that are a power of 2, an even number of powers of 3, and an even number of powers of 5. Thus, the total number of d_i values must be even. This contradicts the fact that our cycle had odd length, so there cannot be an odd-length cycle in G. This makes G a bipartite graph.

(b) [10 points] For this week's assignment, students will work in pairs. Assuming that no two students share the same favorite number, show how you can use G to find the way to pair students such that you maximize the number of students paired with their friends. Analyze the runtime of your algorithm.

Solution: We run any maximum bipartite matching algorithm on G. Because these matchings in G correspond exactly to legal pairings of students, this maximum matching will give us exactly the maximum number of disjoint pairs of friends we can form.

Runtime analysis: Each student can have at most 6 friends ($f/2$, $f/3$, $f/5$, $2f$, $3f$, and $5f$). The number of vertices is $O(n)$ and the number of edges is at most $6n = O(n)$. The maximum possible matching has size $n/2$. Using Ford-Fulkerson will give a runtime of $O(Ef) = O(n^2)$.

This problem can also be done using the same flow network as in part (c).

(c) [20 points] Modify your solution to part (b) so that it works even if some students have the same favorite number. Analyze the runtime of your new algorithm. Your algorithm should run in $O(n^2)$ for full credit.

Hint: Consider using the Ford-Fulkerson algorithm.

Solution: We set up a flow network to solve this problem.

- Create a source s and a sink t.
- For each unique favorite number, we create a vertex.
- For all favorite numbers in the left half of the bipartite graph, we draw an edge from s to the favorite number node with capacity equal to its frequency.
- For all favorite numbers in the right half of the bipartite graph, we draw an edge from the favorite number node to t with capacity equal to its frequency.
- Draw an edge of infinite capacity between any two favorite numbers that differ by a factor of 2, 3, or 5 (the friendship edges). *The capacity doesn't have to be infinite, as long as it's at least the frequency of each number involved.*

Now, we run Ford-Fulkerson to find the maximum flow of F. We will show that we can convert this maximum flow into the optimal pairing of students.

Proof of correctness: We prove correctness by showing a one-to-one correspondence between flows in F and pairings between friends. First, we'll show that any flow in F can be converted into a corresponding pairing of students (a). Then, we show that any pairing of students can be converted into a valid flow in F (b).

(a): After finding the maximum flow in F, let $m_{i,j}$ be the flow along the edge between favorite numbers f_i and f_j. For each edge (i, j), we'll match $m_{i,j}$ of the students with favorite number f_i with $m_{i,j}$ of the students with favorite number f_j. This will always generate a valid pairing of students because the amount of flow passing through each favorite number can never be more than the frequency of that favorite number (due to

the conservation of flow). The number of paired students will be equal to the value of the flow, because the sum of the $m_{i,j}$ values is the total amount of flow that enters the right half of the bipartite graph and flows into t. We also know that all the $m_{i,j}$ values are integers because running Ford-Fulkerson on a flow network with integer capacities results in integer flows.

(b): Given any pairing of students, we set the flow along the edge between favorite numbers f_i and f_j to be the number of pairs of students such that one likes f_i and the other likes f_j. The flow along the edges connected to the source or sink is equal to the number of times each favorite number got matched. This always gives a legal flow because 1) the capacities of the edges from the source and sink are upper bounds on the number of times each number can be matched, and 2) conservation of flow is satisfied because both the incoming edge flow and the outgoing edge flow from the favorite number vertices will equal the number of times that favorite number was matched.

Now that we have a one-to-one correspondence between flows in F and student pairings, we know that finding the maximum flow in F will also give us the maximum student pairing (of friends).

Runtime analysis: Each favorite number has at most 6 neighbors (not counting the source or sink). That is, a favorite number f will only have an edge to $2f$, $3f$, $5f$, $f/2$, $f/3$, and $f/5$ at maximum. There are at most n different favorite numbers, so $E \leq 6n = O(n)$. The maximum flow is at most $n/2 = O(n)$ (maximum pairs that can be made). Thus, Ford-Fulkerson runs in $O(Ef) = O(n^2)$ time.

Problem 4-2. Tree Colors [50 points]

You are given a connected graph $G = (V, E)$ such that each edge is part of at most one cycle. Each edge j has a color c_j which is not necessarily unique. We want to remove edges from the graph such that the remaining edges form a tree. Your goal is to find an algorithm to find the set of edges to remove such that the number of different colors remaining in the tree is maximized.

For all parts of this problem, let n be the number of cycles in G.

(a) [10 points] Set up a bipartite graph F, based on G, such that the size of its maximum matching is equal to $n + u$, where u is the maximum number of different colors that could remain after enough edges have been removed from G to turn it into a tree. No proof is required for this part.

Solution: In the left half of F, add a vertex for each edge in G. In the right half, add a vertex for each cycle and for each color in G. For each edge in G, connect its corresponding vertex in F to the cycle that edge is part of. Also for each edge in G, connect its corresponding vertex in F to the color of that edge.

Note: Based on the problem statement, a way one could guess that this setup works is because we are aiming for the matching to have size $n + u$. We want there to be

a matching for every cycle in G and a matching for every color that remains, which means we'll match edges to both entities.

Another note: the reason we don't ask for proof here is because parts (b), (c), and (d) walk you through the proof. This is based on proofs such as that of lemma 26.9 in CLRS.

(b) [10 points] Consider the optimal solution to this problem (the set of edges S to remove from G such that no cycles remain and the number of different colors remaining is maximized). Show that if you know S, you can convert S into a corresponding matching in F with size equal to $n + u$, where u is the number of different colors remaining in G in the solution.

Solution: Given the solution S, we generate a matching in F as follows:

1. To convert G into a tree, we remove exactly one edge from each cycle. Thus, we match each removed edge to the cycle it was removed from.
2. Among the remaining edges, there are u colors left. For each color, match it to any remaining edge that has that color.

This matching has n matchings from (1) and u matchings from (2), satisfying the size requirement. In addition, this matching is valid because it only uses connections that existed from our part (a) description: edges can only be matched their own cycle or color. No edge will be matched to two entities because cycles are matched to removed edges while colors are matched to edges that weren't removed.

(c) [20 points] Consider the maximum bipartite matching in F. If you are given this maximum matching, show how you can always convert it into a corresponding solution to this problem (a set S of edges to remove from G such that no cycles remain) such that the size of the matching is equal to $n + u$, where u is the number of different colors remaining after the edges in S are removed from G.

Solution: Given the maximum matching in F, we can't always directly convert it into a possible solution. The maximum matching might not match every cycle, but our problem requires that every cycle have an edge removed (in order to form a tree). To deal with this case, we show that we can always convert any maximum matching in F into an "all-cycle" matching of equal size, which we'll define as a matching in which all cycles are matched.

To convert any matching into an all-cycle matching, we observe that for any cycle w that is unmatched in F's maximum bipartite matching, there must be some edge e that belongs to w but is matched to its color instead of w. We then unmatch e from its color and match e to w instead, which doesn't change the total number of matchings. *Proof of existence of e:* suppose to the contrary that w is unmatched in the maximum matching but none of its edges are part of the maximum matching either. This means

that we could have added one more match by pairing w with one of those unmatched edges, contradicting the fact that our matching was a maximum matching.

Now that we have an all-cycle matching, we can convert it into a solution to this problem. The edges matched to cycles are the ones we remove, and the edges matched to colors are the ones that remain and represent the set of different colors remaining. This solution is always legal because exactly one edge is removed from each cycle.

To show that the size of the matching equals $n + u$, we first note that the n part comes from the fact that n cycles are matched in the all-cycle matching. The remaining edges must represent u different colors in G after the edge removals. *What if the remaining edges actually represent more than u different colors, but the matching didn't match all the remaining colors? That can't be the case because our matching was a maximum matching, so no color can remain and not be matched to the edge with that color. This shows that our matching has the size we want it to have.*

(d) [10 points] Based on parts (a), (b), (c), show that finding the maximum matching in F and then converting it into a solution to this problem (using part (c)) will give you the optimal solution to this problem.

Solution: First, we know by part (c) that this conversion will give us a valid solution to this problem. We just have to show that this solution is optimal.

Let our current solution be S, which has u different colors remaining. Our matching in F must have had size $n + u$. Suppose that it's not optimal, and that there is a better solution S' that wasn't found by our algorithm. We know by part (b) that we can convert S' into a bipartite matching of size $n + u'$, where u' is the number of colors remaining after the edges in S' are removed from G. Supposedly, $u' > u$ if our original solution S was not optimal. However, this means that there exists a bipartite matching in F of size $n+u'$, which is greater than $n+u$. This contradicts the premise that we found S from the *maximum* matching in F. Due to this contradiction, we know that if we are finding a solution based on the maximum matching in F, then we also have the best possible solution to this problem, the one that maximizes u.

Alternate solution: Let u^* be the maximum number of different colors that could remain (u^* is the answer to this problem). Let M^* be the size of the maximum matching in F. Part (c) tells us that $M^* \leq n + u^*$ (we can convert a max matching into some solution, not necessarily optimal). Part (b) tells us that $n + u^* \leq M^*$ (we can convert the optimal solution into some matching, not necessarily optimal). Putting both inequalities together gives us $M^* = n + u^*$, proving that the maximum matching also corresponds to the optimal solution.

Problem 4-3. Staircase [50 points]

There is a new staircase on campus with n steps, and the i^{th} step has integer height h_i. The first step always has height 0 and the last step has height m. Unfortunately, this new staircase was

poorly designed. The step heights are not evenly spaced, and some steps are actually lower than the previous steps.

You are hired to fix the staircase so that the step heights form a non-decreasing sequence of integers. It costs $c(x)$ dollars to increase or decrease a single step's height by x, where $c(x)$ is a given price function. In addition, the building regulations state that a good staircase should have each step be exactly d higher than the previous step. If two consecutive steps do not differ by d in height but instead differ by $\triangle$, then you will be fined $p \cdot |\triangle - d|$ dollars. Your total fine is equal to:

$$\sum_{i=2}^{n} p \cdot |(h_i - h_{i-1}) - d|$$

Given n, the initial step heights h_i, p, d, and $c(x)$, design a dynamic programming algorithm to find the set of modifications to form a non-decreasing sequence of step heights that will incur the minimum total cost (cost of fixing plus fines).

The first step and the last step should not be modified. You may assume that $n \geq m \geq d$. For full credit, your algorithm should have runtime $O(nm^2)$. ($O(nm)$ is possible but not required.)

Solution: This problem can be solved using dynamic programming. Let $D[i][x]$ be the minimum cost of fixing steps 1 through i such that the i^{th} has height x. Let $P[i][x]$ be the height of the $(i-1)^{\text{th}}$ step associated with optimal cost of $D[i][x]$.

Base case: For $i = 1$, $D[1][0] = 0$ and $D[1][x] = \infty$ for any $x > 0$. The first step initially has height 0, so it will cost 0 to keep the first step height the same. We are not allowed to modify the first step's height. To force the algorithm to avoid solutions that modify the first step, we assign an infinite cost to modifying the first step.

Recurrence: For $i > 1$, $D[i][x] = \min_{0 \leq k \leq x} (c(|x - h_i|) + D[i-1][k] + p \cdot |(x-k) - d|)$. To find the minimum cost of fixing the first i steps such that the i^{th} step has height x, we consider all possible values of the height of the $(i-1)^{\text{th}}$ step. If the previous step has height k, then the cost of fixing the staircase up to the i^{th} step is equal to the sum of three terms:

- $c(|x - h_i|)$, the cost of changing the height of step i to x.
- $D[i-1][k]$, the cost of fixing the first $i-1$ steps such that the $(i-1)^{\text{th}}$ step has height k.
- $p \cdot |(x-k) - d|$, the additional fine incurred for having steps i and $(i-1)$ not differ by d.

$P[i][x]$ will be set to whichever k produced the minimum cost in the recurrence for $D[i][x]$.

Answer: The minimum cost to modify the staircase is $D[n][m]$, the minimum cost of fixing all n steps and leaving the last step with height m. To find the actual step heights, we use P to backtrack and figure out what step heights are part of the optimal solution corresponding to $D[n][m]$. Given that our last step had height m, the $(n-1)^{\text{th}}$ step in the optimal solution must have had height $P[n][m]$. Then, we know that the $(n-2)^{\text{th}}$ step must have had height $P[n-1][P[n][m]]$. The $(n-3)^{\text{th}}$ step must have had height $P[n-2][P[n-1][P[n][m]]]$. Continuing like this, we can use P to backtrack and build up the optimal step heights backwards.

Runtime analysis: There are a total of $n(m+1)$ subproblems to compute. Each subproblem requires looking through m possible values of k, which takes $O(m)$. Thus, the total runtime is $O(nm^2)$.

Problem 4-4. Raining [50 points]

There are n dormitories connected by exactly $n-1$ roads, forming a tree. Some roads are above ground, while others are underground tunnels.

You want to pick one of the dormitories as the location for your club meeting. For each dormitory i, there are s_i students who will travel across campus from dormitory i to this meeting point.

Because it is raining today, students will use umbrellas whenever they travel above the ground. One umbrella can only cover one student, and the umbrellas are so flimsy that they will break after being used on just one road. As a result, some students will have to buy multiple umbrellas.

Design an $O(n)$ runtime algorithm to pick the meeting point such that the number of total umbrellas students need is minimized. See Figure 1 for an example scenario.

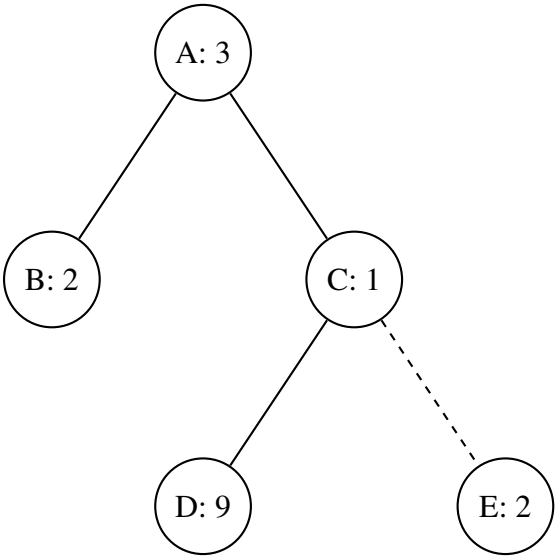

Figure 1: Example dormitory layout, where a dashed edge represents an underground tunnel. Each dormitory is labeled with its s_i value. If the meeting point is at dormitory A, then students will need a total of 23 umbrellas: the 5 students from B, C, and E each need one umbrella while the 9 students from D each need two umbrellas (along (D, C) and again along (C, A)). The optimal meeting point is at D, with only 15 umbrellas necessary.

Solution: First, arbitrarily root the tree. For each node (dormitory), figure out which node is the parent and which nodes are the children. Let $C(i)$ be the set of node i's children. Define u_{ij} to be 1 if the road between i and j is above ground, and 0 otherwise. Next, let S be the total number of students: $S = \sum_i s_i$. This takes $O(n)$ to compute.

Then, a precomputation: let $T(i)$ be the number of students contained within the subtree rooted at node i. Each node's T value is equal to s_i plus the sum of T values of its children: $T(i) = s_i + \sum_{j \in C(i)} T(j)$. This takes $O(n)$ to compute if we start from the leaves and work upwards. The total number of things we are adding up is the total size of all the C sets, which is n.

Now, let $U(i)$ be the total number umbrellas needed if the meeting point is at node i.

First, we want to compute U for the root. To do this, define a helper value $R(i)$ to be the number of above-ground roads on the path from the root to node i. For each node, $R(i)$ is equal to the R value of its parent, plus 1 if the road from i to its parent is above ground. We can compute all the R values in $O(n)$ total by calculating them top-down. Then, for the root, its U value is equal to $\sum_i R(i)s_i$.

Next, we need to compute the rest of the U values in $O(1)$ each. Consider any node i and any of its children j. If the students were originally going to meet at i but now instead choose to meet at j, the students in j's subtree have to travel one fewer road ($= T(j)$ students) and every student not in j's subtree has one extra road to travel ($= S - T(j)$ students). Thus, we can compute $U(j)$ from $U(i)$ by counting how many more or fewer students use the (i, j) road. If the (i, j) road is underground, then $U(i) = U(j)$. Otherwise, $U(j) = U(i) - T(j) + (S - T(j))$.

If r is the root node, we can compute $U(r)$ in $O(n)$ using the summation of R values. Then, for each other node, we can compute its U value using the formula above, based on its parent's U value. We compute these values top-down and each non-root U value can be computed in $O(1)$. In total, this is another $O(n)$. We pick the node with the smallest U value as the meeting point.

Each step has runtime $O(n)$, so our overall runtime is $O(n)$.