Problem 1, CS38 Set 4, Matt Lim

Here is pseudo-code for our algorithm.

```
Decomposition(String $x = x_1x_2 \ldots x_n$)
  1. $OPT(i,j) = q(x_ix_{i+1} \ldots x_j)$ for all $i, j$ $(j \geq i)$
  2. for $m = 0$ to $n-1$
      3. for $i = 1$ to $n$
          4. $j = i + m$
          5. if $j \leq n$
              6. for $k = i$ to $j-1$
                  7. $OPT(i,j) = max\{OPT(i,j), OPT(i,k) + OPT(k+1,j)\}$
  8. return $OPT(1,n)$
```

We will now show that this algorithm runs in $O(n^3)$ time. We can see that we have three loops, with each loop iterating through less than or equal to n times. Then we can see that the work done for each time through the loops is constant time (line 7). Thus this algorithm runs in $O(n^3)$.

Now we will show correctness. First we initialize all cells to be the quality of the entire string. This is done in line 1. Then we iterate down the diagonals. This is done in lines 2-3. This makes our algorithm fill in the table by increasing length of the substring. For example, first it fills out all substrings $x_1, x_2, x_3, etc.$ Then it fills out $x_1x_2, x_2x_3, x_3x_4, etc.$ This is important because it lets each loop through be constant time; that is, it makes line 7 constant time. Let us examine line 7 more closely. For each cell, we do the following. Let the string we are considering be x_{ij}. Then, we consider all $j-i$ ways to break it up into two smaller strings. So for $x_1x_2x_3x_4$, we look at x_1 and $x_2x_3x_4$, x_1x_2 and x_3x_4, and $x_1x_2x_3$ and x_4. For each split into x_L and x_R, where x_L is the left side of the split and x_R is the right side of the split, we calculate the score $OPT(x_L) + OPT(x_R)$. This can be found in constant time, since we already filled out the table for all substrings with length less than x_{ij}. We have that $OPT(x_L) + OPT(x_R)$ is the best possible score for that split, since it is the best possible score for the left side and best possible score for the right side. So, after calculating all these scores, we take the maximum of all the "split" scores along with the score already in the cell (which is just the quality of the entire substring) and insert it into the cell. Note that our algorithm does this after every $OPT(x_L) + OPT(x_R)$ is calculated, but the end result is the same thing. Note that this, given the way we build our table, gives us the maximum score out of all the decompositions for each substring. Now we can see that the value in each cell i, j is the highest quality of the substring x_{ij}. So after we finish filling out the table, all we must do is return the value in cell $1, n$, which gives us the highest quality decomposition value for the entire string.

Problem 2, CS38 Set 4, Matt Lim

This algorithm resembles a modified shortest paths algorithm, where we increment robustness whenever the shortest path to v is equalized and reset robustness when a new shortest path to v is found. Here is the pseudo-code.

Num-Shortest-Paths(V, E, c, s, t)

```
1. foreach node $v \in V$
    2. $M[0, v] = \infty$
    3. $R(v) = 1$
4. $M[0, s] = 0$
    5. for $i = 1$ to $n - 1$
        6. foreach node $v \in V$
            7. $M[i, v] = M[i - 1, v]$
            8. foreach edge $(v, w) \in E$
                9. double $new = M[i - 1, w] + c_{vw}$
                10. if $new == M[i, v]$
                    11. $R(v) = R(v) + R(w)$
                12. elif $new < M[i, v]$
                    13. $R(v) = R(w)$
                    14. $M[i, v] = new$
15. return $R(t)$
```

We will now show that this algorithm runs in time $O(nm)$. Our algorithm is basically the same as the shortest path algorithm shown in lecture 10. The only steps we added are constant time. And since the shortest path algorithm is $O(nm)$, our algorithm is $O(nm)$. This can also be observed from the fact that the first for loop iterates from 1 through n, and the two for loops contained within iterate through at most m edges. So this algorithm is clearly $O(nm)$.

We will now show correctness. We already have that our algorithm correctly computes shortest paths, because it is the same as the algorithm shown in lecture 10. So, we must only look at the additional pseudo-code we added. We can see that our algorithm increments $R(v)$ by $R(w)$ for a vertex v whenever we find a new path to v that uses the edge $(v, w) \in E$, with the new path being of equal weight to the current shortest path to v. That is, when $new == M[i, v]$ (new from line 9), we increment the count by $R(w)$. We increment it by $R(w)$ because if there are $R(w)$ shortest paths to w and we find a shortest path from s to v that has the last edge of (v, w), with that path having the same weight as the current shortest path to v, then there are clearly $R(v) + R(w)$ shortest paths to v, $R(v)$ being the number of shortest paths there were to v before we considered edge (v, w). In other words, we increment by $R(w)$ because it is the robustness of the shortest path from s to w (the weight of which combined with c_{iw} equals $R(v)$), which means that since we have the edge (v, w), we can add this robustness to the current value for v, $R(v)$. Next, we can see that our algorithm resets $R(v)$ to $R(w)$ whenever we find a path to v that is of lesser weight to the current path of v. This is because we have found a new shortest path. And clearly, since we just found this path, there are only $R(w)$ of them, since there are $R(w)$ shortest paths to w and we are considering the one edge (v, w). So this continues until we have found the robustness of the shortest paths from s to every vertex $v \in V$. Then it is clear that $R(t)$ is the "robustness" of the shortest path from s to t.

Problem 3, CS38 Set 4, Matt Lim

Given a bipartite graph $G = (L \cup R, E)$, we will formulate our matroids I_1 and I_2 as following. First however, we will assume that every vertex in L has at least one edge going to some vertex in R, and that every vertex in R has at least one edge going to some vertex L. Note that if we don't assume this, those vertices don't matter when finding the maximum matching anyways. Now, onto the problem. Let the vertices in L be labeled $1, 2, 3, \ldots, n$ and the vertices in R be labeled $1', 2', 3', \ldots, m'$. The bases of I_1 will be of cardinality $|L| = n$. They will be all sets of the form $\{(1, x'_1), (2, x'_2), \ldots, (n, x'_n)\}$, $x'_i \in R$, where x'_i can equal x'_j, and none of the vertices from L repeat. The bases of I_2 will be of cardinality $|R| = m$. They will be all sets of the form $\{(1', x_1), (2', x_2), \ldots, (m', x_m)\}$, $x_i \in L$, where x_i can equal x_j, and none of the vertices from R repeat. We will have I_1 and I_2 contain all subsets of their respective bases. We have that I_1 and I_2 are indeed matroids because they satisfy the three axioms. The first axiom is clearly satisfied since we include all subsets of the bases, and the empty set is a subset of everything. The second axiom is clearly satisfied because we said that I_1 and I_2 contain all subsets of their bases. The third axiom is satisfied because, WLOG, if we have A and B be subsets of I_1 with $|B| > |A|$, then there exists an edge in B of the form (i, x'), where the vertex i is not contained in any edge in A. So we can just add this edge to A, and it is still in I_1 because it is still a subset of the bases we described above. Then we have that finding the maximum cardinality independent set in the intersection of I_1 and I_2 is the same as finding the maximum matching of G.

We will now explain why this works. First we will show why this gives us a actual matching. We have that every set $A \in I_1$ only contains one edge with any such vertex i (no repeat vertices from L), and that every set $B \in I_2$ only contains one edge with any such vertex i' (no repeats from R). Thus we have that the intersection of any two sets from I_1 and I_2 will not have any repeat vertices; that is, the same vertex in two different edges. So we have that the maximum cardinality independent set in the intersection of I_1 and I_2 is a valid matching. Now we will show that it is the maximum matching. Let $k = min(m, n)$. Then we have that maximum matching cardinality is less than or equal k. Let the maximum matching cardinality be c. Now, assume to the contrary that the maximum cardinality independent set in the intersection of I_1 and I_2 is $c' \neq c$. We will first show that $c' < c$ cannot be true. So, assume to the contrary that $c' < c$. Let the maximum matching set of edges be C. We have that $C \in I_1$ and $C \in I_2$; this is clear given how we defined our matroids. But then we have that $c' < c$ cannot be true, since the cardinality of the maximum independent set in the intersection would be at least c. Now we will show that $c' > c$ cannot be true. So, assume to the contrary that $c' > c$. Now we will consider the maximum independent set in the intersection. We have that it has c' edges. We showed above that these edges form a valid matching. But then we have a contradiction, since this valid matching has more edges than the maximum matching. Thus we can conclude that $c' = c$ and thus the proof is complete.

Problem 4, CS38 Set 4, Matt Lim

(a) Let G be a k-regular bipartite graph. We proved in lecture that that max cardinality of a matching in G equals the value of max-flow in G'. So, let's consider G', as described in lecture 12. That is, given $G = (L \cup R, E)$, G' directs all edges from L to R and assigns infinite capacity for them, adds a source s and unit capacity edges from s to each node in L, and adds a sink t and unit capacity edges from each node in R to t. Note that $|L| = |R|$ because G is a k-regular bipartite graph. We can see that $|L|$ must equal $|R|$ in a k-regular graph by the following argument. It is trivially true for $k = 1$. Then, if we increment k, it is also clearly true, since we again must add exactly one edge to every vertex, which can only be done if $|L| = |R|$. And we can extend this argument to see that $|L| = |R|$ for all k-regular bipartite graphs, where $k \geq 1$. Another way to argue this is that each set of nodes L and R have the same number of edges. And since we can get the number of nodes in each set by dividing the number of edges by k, both set of nodes has the same number of nodes. Now note that the min-cut that separates s and t has the value of $|L| = |R|$. This is because the min-cut will clearly just go through all the unit capacity edges from s to all nodes in L or through all the unit capacity edges from t to all nodes in R (or some combination of those edges), since the capacity of all the other edges is infinity. Then, by the max-flow min-cut theorem, we have that the max-flow has the value of $|L| = |R|$. Then we have that the max cardinality of a matching in G equals $|L| = |R|$, by the theorem stated at the beginning. But by definition of a matching, this means that this is a perfect matching. So we are done.

(b) Let $G = (U \cup V, E)$ be an undirected bipartite graph with $|U| = |V|$. For a subset $S \subseteq U$, let $N(S) \subseteq V$ denote the neighbors of the vertices in subset S. We will prove that there exists a perfect matching in G if and only if we have $|N(S)| \geq |S|$ for all subsets $S \subseteq U$.

We will first prove the direction "If there exists a perfect matching in G, then $|N(S)| \geq |S|$ for all subsets $S \subseteq U$." So, assume there exists a perfect matching in G. Let us look at an arbitrary subset S. Then we have that every node in S goes to a node in $N(S)$ (because of the perfect matching). This gives us that $|N(S)|$ is at least as large as $|S|$, which gives us that $|N(S)| \geq |S|$ for any $S \subseteq U$. So we have proved the forward direction.

Now we will prove the direction "If $|N(S)| \geq |S|$, then there exists a perfect matching in G." We will prove this direction by proving the contrapositive: "If there does not exist a perfect matching in G, then $|N(S)| < |S|$." So, assume there does not exist a perfect matching in G. Let us consider the graph G' as described in lecture and in $4(a)$, with s the source node connected to all vertices in U and t the sink node connected to all vertices in V. We have that there is no perfect matching, which means that the value of the max-flow is less than $|U|$, which means by the max-flow min-cut theorem that the value of the min-cut is less than $|U|$. Let C be the min-cut and $|C|$ be the value of the min-cut. Then we have that $|C| < |U|$. Note that the min-cut cannot go through edges in the middle because they have infinite capacity, and $|U|$ is finite which means $|C|$ is finite. Then we have that the min cut contains a cut A with $|A|$ edges from s to U (which means the value of that part of the cut is $|A|$ since those edges are unit capacity) and contains a cut B with $|B|$ edges from V to t (which means the value of that part of the cut is $|B|$ since those edges are unit capacity). Then we have that $C = A + B$ and $|C| = |A| + |B|$. Now, let S be the subset of U with edges to s that were not cut by A, and $N(S)$ be the neighborhood of S in V. Then the following two equations are true:

$$|S| = |U| - |A|$$

$$|N(S)| = |B|$$

The first equation is true because s has an edge to every vertex in U. So if we only include that vertices in U that have edges that were not cut by A, we are left with $|U| - |A|$ vertices. The second equation is true because $N(S)$ can only contain the vertices in V that had their edges to t cut by B, which is exactly $|B|$ vertices. This is because if S had an edge to any other vertices in V, the cut would have cut them, which would make the value of the min-cut infinite. So, let's work with the above two equations. We have $|C| = |A| + |B| \implies |B| = |C| - |A|$ and that $|S| = |U| - |A| \implies |A| = |U| - |S|$. So then we have the following:

$$|N(S)| = |C| - |A|$$

$$|N(S)| = |C| - (|U| - |S|)$$

$$|N(S)| = |C| - |U| + |S|$$
$$|N(S)| - |S| = |C| - |U|$$

But we have that $|C| < |U| \implies |C| - |U| < 0$. So then we have that

$$|N(S)| - |S| < 0$$

Note that this inequality applies to any subset S in U, because we can just modify the cut A to obtain any subset S in U. Thus we have proved the contrapositive and thus proved the desired direction.

We proved both directions, so we can conclude that there exists a perfect matching in G if and only if we have $|N(S)| \geq |S|$ for all subsets $S \subseteq U$.

Problem 5, CS38 Set 4, Matt Lim

(a) We will show how to find a circulation, or determine that one does not exist, by reducing to max-flow. To do this, we will prove the following: "There is a circulation in G if and only if there is a max-flow with saturated capacities in G'." Note that when we are talking about saturation in this problem, we are just talking about saturating the capacities of the edges from s and the edges to t.We will let G' be the graph specified as follows. We will add vertices s and t. s will contain one edge to every vertex in G that has a negative demand d. The capacities of these edges will be $-d$. t will contain one edge to every vertex in G that has a positive demand d. The capacities of these edges will be d.

We will first prove the direction "If there is a circulation in G, then there is a max-flow with saturated capacities of the added edges in G'." So, assume there is a circulation in G. Then we have that for all $v \in V$,

$$\sum_{e \text{ entering } v} f(e) - \sum_{e \text{ exiting } v} f(e) = d(v).$$

This means that we can saturate all the edges going out from s to their full capacity. This is because a node v connected to s with $d(v) = d$ must have $-d$ net amount of flow exiting it. So we can clearly send $-d$ amount of flow into it, which saturates its edge with s. Now we must show that we can saturate all the edges going into t. So, consider a node v connected to t with $d(v) = d$. We have that this node must have d net amount of flow entering it. It then follows that it can send d amount of flow to t, saturating its edge to t. So we have that all the edges to t are saturated. So we have that there is a max-flow with saturated capacities of the added edges in G'.

Now we will prove the direction "If there is a max-flow with saturated capacities in G', then there is a circulation in G." To prove this direction, we will prove the contrapositive: "If there is not a circulation in G, then there is not a max-flow with saturated capacities of the added edges in G'." If G does not have a circulation because $0 \le f(e) \le c(e)$ for all $e \in E$ is not true, then G' clearly does not have a s to t flow because we have violated the definition of a flow as given in lecture 12. So, we have proved this direction for that point. Now for the second point. Assume that

$$\sum_{e \text{ entering } v} f(e) - \sum_{e \text{ exiting } v} f(e) = d(v)$$

is not true for all $v \in V$. That is, the sum of all the $d(v)$s is no longer 0. So WLOG we can just consider two cases: when $d(v)$ increases for one node v or decreases for one node v.

Let us look at the case when $d(v)$ increases for one node. First we will observe what happens if v is connected to s. $d(v)$ increasing then means that there is less flow coming out from s, which means we cannot saturate the edges going to t. Now we will observe what happens if v is connected to t. $d(v)$ increasing then means that we have increased the capacity of an edge to t. But this capacity cannot possibly be saturated since the sum of the capacities of the edges from s is now less than the sum of the capacities of the edges to t (since before, with the circulation, they were equal). Now, if $d(v)$ increases so that $d'(v) = 0$, then we again have less flow coming out of s, which means the edges going to t can't be saturated.

Now let us look at the case when $d(v)$ decreases for one node. First we will observe what happens if v is connected to s. $d(v)$ decreasing then means that there is more flow coming out from s. But we have not changed the amount of flow that can go into t. So an edge from s must not be fully saturated, since otherwise there would be more flow coming out of s then going into t. Now we will observe what happens if v is connected to t. $d(v)$ decreasing then means that we have decreased the capacity of an edge to t. But this means we cannot saturate all the edges coming from s, since otherwise we would have more flow coming out of s than going into t. Now, if $d(v)$ decreases so that $d'(v) = 0$, then we again cannot saturate all the edges coming from s, since otherwise we would have more flow coming out of s than going into t.

So we have proved both cases, and also proved both directions of the contrapositive. So finally we have finished our reduction. Note that our way of finding a circulation (or determining that one does not exist) runs in time comparable to the algorithms for max-flow, since it just consists of adding two vertices s and t and order n edges to the graph, then finding the max-flow of that new graph and seeing if it is equal to the sum of the capacities of the edges from s.

(b) Let graph $G = (V, E)$ be a directed graph as described in the header of problem 5, along with the requirements specified in $5(b)$. We will generate graph G' as follows. For every node v we will do the following. We will change the value of $d(v)$ to be

$$d'(v) = d(v) - \sum_{e \text{ entering } v} l(e) + \sum_{e \text{ exiting } v} l(e).$$

We also change the capacities of each edge to be $c'(e) = c(e) - l(e)$. These changes clearly cannot create a circulation in G. These changes also mean that if $f(e') \geq 0$ for all $e' \in E'$, then $f(e) \geq l(e)$ for all $e \in E$. This is because we are basically subtracting the lower bounds for the flows for each edge in G from the amount of flow that can flow through that edge in G to get the amount of flow that can flow through that edge in G'. So if we can still have some non-negative flow in an edge in G', we can satisfy the lower bound for the flow in the corresponding edge in G. So we have that if there is a circulation for G', the two axioms shown in the problem have to be satisfied, which as we explained above is the same as having a circulation for G which has the modified first axiom satisfied (the lower bound). Then we can do the same thing as we did in part (a) for G' to determine if G' has a circulation. If it does, then G has a circulation with $f(e) \geq l(e)$ for all $e \in E$.

(c) Given $G = (C \cup P, E)$, we will make G' as follows. We will add a source node s that has an edge to every node in C, with each of those edges having a lower bound q_c and capacity q'_c. Then we will add a sink node t that has an edge to every node in P, with each of those edges being directed toward t and having a lower bound s_p and capacity s'_p. The edges between C and P will have lower bound 0 and capacity 1. Finally we will add one edge from t directed to s with lower bound 0 and capacity ∞. Also, for all $v \in C \cup P \cup \{s, t\}$,

$$d(v) = 0$$

Then all we need to do is see if this graph has a circulation with lower bounds by running it through part (b). If it does, then the survey parameters are feasible. This is because if there is a circulation, that means we met the specified lower bounds from s to C and from P to t. That is, we asked each consumer at least q_c questions and at most q'_c questions, since q_c was our lower bound for edges from s to C and q'_c was our capacity for edges from s to C. We also had each product surveyed at least s_p times and at most s'_p times. This is because the amount of "flow" that went into any edge in P equals the amount of "flow" that went out into t, since we set the $d(v)$ of all nodes v to be 0. And since the lower bound for edges from P to t was s_p and the capacity of those same edges was s'_p, we have the desired number of times each product was surveyed.