Introduction to Algorithms	**Solution for Prelim 1 Practice**
CS 4820, Spring 2016	

1.a. The statement is true. Consider any perfect matching M. Suppose m and w are not matched in M. Then the pair (m, w) is an instability in M because both m and w prefer each other over their matched partners in M.

1.b. The statement is true. In every round of the Gale–Shapley algorithm, an unmatched man proposes to his highest preference among all women he hasn't proposed to yet. In the algorithm, women accept proposals unless they are already matched to a man that they rank higher.

Hence in the first n rounds, every man proposes to his matched partner in M. The women accept all proposals. After the first n proposals, all men are matched and the algorithm terminates.

1.c. True, as she has a choice between $k + 1$ men.

1.d. not always true, as her first and only proposal can be from any men, including her top choice.

2. Solution A For each student, consider the initial sequence of lectures he/she attends, Let i_1 be the first lecture student i doesn't attend. Now let the student i with maximum i_1 value tape the first $i_1 - 1$ lectures. We will have a handoff before this lecture $h_1 = i_1$, and a different student has to start taping. Again, look for the student that attends lecture h_1, and as many of the next lectures after h_1 as possible, and let h_2 be the first lecture after h_1 that this student doesn't attend. He/she will be in charge of taping lectures $h_1, \ldots, h_2 - 1$. We continue this way till all lectures are covered.

Running time. The main part of the algorithm is to find the next handoff after a handoff before lecture h, we need to look through all sets S_i starting from $i = h$, and see who is the student that is in all these sets. This can be easily done by setting up an array of length m for the students, and checking marking each member of S_i on this array, will we find that no students attends all lectures from h all the way to h'. Now h' is our next handoff. Each S_i has at most m elements, and there are at most $h' - h \leq n$ lectures to consider, so this takes at most $O(mn)$ time. There can be at most n handoffs, so the total time is at most $O(mn^2)$. Alternately, we can also notice that we took $O(m)$ time to process a set S_i for a lecture, and there are only n lectures, so the total time is also bounded by $O(mn)$.

Correctness. We offer here a "greedy stays ahead" argument. We claim that the first j students selected for taping cover as long an initial segment of the lecture sequence as possible. This will prove the claim of optimality, as if the algorithm uses some k students, and by our statement the first $k - 1$ students, who do not cover all lectures, cover as long an initial segment as any $k - 1$ students can do, than clearly we need at least k students.

We prove the statement by induction on j. For $j = 1$ this was exactly our selection criteria for the first student. Now consider $j > 1$. Assume the algorithm has the jth student cover the interval $[h_{j-1} - h_j - 1]$. By the induction hypothesis any $j - 1$ student cannot cover the whole interval $[1, h_{j-1}]$, so in any schedule the lecture h_{j-1} is covered by student j or later. Among the

students who are at lecture h_{j-1}, we selected to one that does as long an interval of the following lectures as possible, which proves our claim.

Alternately, an exchange argument works well also. Consider any alternate optimal schedule. Let i be the first lecture where the algorithm selected some student s different from the optimal solution, that picked a student s'. There are two cases. If s already is taping lecture $i-1$, we can delay the handoff in the optimal solution to lecture i and hence making the two schedules identical for longer. Alternately, if s not taping lecture $i-1$ (and at lecture $i-1$ the two schedules were identical), than the person taping lecture $i-1$ is not attending lecture i (as the greedy wouldn't do a handoff otherwise), and we can change the optimal solution to use s instead of s' for the whole period that the optimal solution used s' (as s was chosen to have the longest sequence).

Solution B Alternately, we can reformulate the problem to make the description of the algorithm more convenient. Since we can assume that an optimal plan chooses different students for lecture i and $i+1$ only if the student for lecture i is not available for lecture $i+1$. Hence, we can represent the availability of students as a collection of intervals $I_1, \ldots, I_k$ with integer endpoints in the range from 1 to n. Each interval $I_r = [i, j]$ indicates that a particular student is available through lectures i to j. (A single student might contribute multiple intervals.) Since every student can contribute as most n intervals the total number of intervals is at most mn, polynomial in the number of students and the number of lectures. In this interval representation of student availability, the goal is to cover all n lectures with as few intervals as possible.

Algorithm. We describe a natural greedy algorithm. The algorithm constructs a subset A of the intervals. Initially, we let A be empty. In each iteration of the algorithm, we modify A by the following procedure: (1) select an interval I_r with latest finish time among all intervals that cover the first lecture i that is not covered by one of the intervals in A and (2) add this interval to A. We repeat this procedure until all intervals are covered.

Running time. We will express the running time in terms of the number n of lectures and the number m of students.

To compute all intervals that a particular student contributes a single pass through the input is enough, which takes $O(mn)$ time. Hence, the total time for constructing all intervals is $O(m^2n)$.

Each iteration of the algorithm can be implemented by a single pass through all intervals, which takes $O(mn)$ time. The number of iterations is $O(n)$ since in each iteration we cover at least one lecture. Hence, the total running time of the algorithm is $O(mn^2)$.

The sum total of the interval construction and the greedy algorithm is $O(m^2n + mn^2)$.

Correctness. We prove correctness by a direct argument. Suppose the greedy interval set has cardinality T. Let i_t be the earliest lecture that Greedy covered in iteration t but that was not covered in iteration $t-1$. Note that $i_1 = 1$.

The following claim implies that any subset of availability intervals that covers the lectures $i_1, \ldots, i_T$ has cardinality at least T.

"No availability interval I_r covers more than one of the lectures $i_1, \ldots, i_T$."

To see why this is true, notice that we may assume that I_r covers at least one of the lectures $i_1, \ldots, i_T$. Let i_t be the earliest of the lectures $i_1, \ldots, i_T$ that I_r covers. Hence, Greedy considered the interval I_r in the t-th iteration. It follows that the interval selected by Greedy finishes no earlier than I_r. Therefore, the lectures $i_{t+1}, \ldots, i_T$ happen after I_r finishes.

2-alt. Algorithm. The newly added edge (v, w) forms a cycle C with the tree T. Find this cycle, which is the path in the tree from v to w plus the new edge (v, w), and delete the most expensive edge e' in the cycle. The cycle is in the graph consisting of T and the new edge, a graph with only n edges, so takes $O(n)$ time, and finding the max weight edge also $O(n)$.

Correctness. Let T' be this new tree. For any edge $e \in T'$ deleting e from T' break T' into two trees. Consider the cut (A_e, B_e) separating these two parts. We claim e is the min weight edge in this cut. This will then imply that e is in the MST of the new graph G proving our claim.

For any edge $e' \in E$ that wasn't in the original MST T, e' cannot be the min-cost edge in any cut, as the min-cost edge is always in the spanning tree. So the min-cost edge in the cut the cut (A_e, B_e) is either edge e, or the edge e' we deleted from the cycle C. However, note if e' is in the cut, another edge of the cycle must also cross the cut, and e' was most expensive, so e' also cannot be the min-cost edge in the cut. This proves that e must be the min-cost edge.

3.a. Consider the sequence $s_i = 1$ for $i = 1, \ldots 5$, $w_1 = 0$, and $w_i = 1000$ for $i = 2, \ldots, 5$. The optimum solution is to choose W each step, loosing only 5 votes, the algorithm chooses S each step and looses 4000.

3.b. Let $opt(i, w)$ be the value of the optimal solution for days 1 through i where the the choice on day i is W, and $opt(i, s)$ is be the value of the optimal solution for days 1 through i where the the choice on day i is S.

Now the solution to the problem is the smaller of $opt(1, w)$ and $opt(1, s)$.

opt(i,w)

We have $opt(1, w) = w_1$ and $opt(i, s) = s_1$.

For $i > 1$ we can find a solution based on the previous say's solution:

$$opt(i, w) = w_i + min(opt(i-1, w), opt(i-1, s) + 1000)$$
$$opt(i, s) = s_i + min(opt(i-1, s), opt(i-1, w) + 1000)$$

We are setting up a table of $2n$ items for the n days, and can get the value for each item by choosing between two options, for a total of $O(n)$ time.

4.

(a) **True.** The value $Opt(k, v)$ is the min cost path from s to v using at most k edges where repeated nodes or edges are allowed. The min cost path without repeated edges or nodes can only be more expensive than the min-cost path that allowed repeated edges. So if the min-sot path of at most $n-1$ edges that allowed for repetition happens not to have repeated nodes, it must be the cheapest.

(b) We can find the min cost path of at most n edges, by using the table to trace back. The value $Opt(n, v)$ was determined by the minimization

$$Opt(n, v) = \min_{(w,v) \in E} c_{wv} + Opt(n-1, w).$$

The edge (w, v) where the minimum occurs, is the last edge on this path, and then we use the expression for $Opt(n-1, w)$ to get the next edge, etc. The path P we find **must** have exactly n edges as $Opt(n, v) \neq Opt(n-1, v)$. A path with n edges in an n node graph must contain a cycle, so tracing back to find the path, we do find a cycle C. Finally, the cycle C we find this way must have strictly negative cost. To see why the cycle must be of negative cost, notice that $P \setminus C$ is a path of fewer edges, and hence by $Opt(n, v) \neq Opt(n-1, v)$ must be of larger cost.

(c) **Yes**, if $Opt(n, v) = Opt(n-1, v)$ than the graph has no negative cycles, or put it differently, if the graph has negative cycles, we must have $Opt(n, v) < Opt(n-1, v)$. To see why first notice that for any v node on a negative cycle, for a sufficiently large k we must have $Opt(k, v) < Opt(n-1, v)$, as we can go around the cycle many times and arbitrarily decrease the cost. Now we have to argue that this is also true for some node v with $k = n$. To see why, we notice that each iteration of the algorithm is completely identical, so if at any iteration, we get that $Opt(i, v) = Opt(i-1, v)$ for all nodes v, than from then on, all further iteration, all values will remain the same, so $Opt(k, v) = Opt(i-1, v)$ for all $k > i$ also.