Minor I (COL 702)

- Write brief answers. You can assume any result which was used or proved in the class.

1. You are given a directed graph $G = (V, E)$. A vertex v is called nice if there is a directed path from v to each of the other vertices in G. Suppose we run DFS on G starting from some vertex s and let w be the vertex with the highest finish time $F[w]$. Prove that if G has a nice vertex, then w is a nice vertex.

   Let v be a nice vertex in G. For a vertex u, let I_u denote the interval $[D[u], F[u]]$. By definition of w, we know that I_w ends after I_v ends. By the nesting property, two cases arise: (i) I_v and I_w are disjoint, and so I_v lies to the left of I_w, or (ii) I_v is contained in I_w .

   If the second case happens, we have shown in the class that there is a path from w to v and so w is also nice . So assume that the first case happens. Let P be a path from v to w (such a path exists because v is nice). Let u be the first vertex in this path whose interval I_u is to the right of I_v (such a vertex must exist because I_w has this property) . Let u' be the vertex before u on this path. Then $I_{u'}$ is to the left of I_u (if $I_{u'}$ were contained in I_u or to its right, then $I_{u'}$ will also be to the right of I_v, which is not possible) . But now, we have the edge (u', u) with $I_{u'}$ to the left of I_u, which we know is not possible . Thus this case cannot happen.

   Common Mistakes: Many of you tried to show that if w is a nice vertex then $F[w]$ will be highest. This is clearly wrong because there can be more than one nice vertices, and all of them cannot have the highest $F[w]$ value.

2. Recall the coin changing problem: you are given (infinite supply of) coins of denomination$c_1, c_2, \ldots, c_k$. Assume $c_1 = 1$ and c_i divides c_{i+1} for $i = 1, \ldots, k-1$. Suppose we want to have a change for an integer amount M and would like to use as few coins as possible. Let c_i be the largest coin denomination which is at most M. Prove that there is an optimal solution which chooses c_i.

   We first prove the following fact: suppose $a_1, \ldots, a_k$ are positive integers such that $a_1 + \ldots + a_k \geq c$ for some positive c and suppose a_1 divides a_2, a_2 divides a_2 and so on, and a_k divides c, then there is a subset of $\{a_1, \ldots, a_k\}$ which adds up to exactly c . We prove this by induction on c. When $c = 1$, then each of the a_i must be 1 (since it needs to divide c) and so we are done.

   Now suppose the statement is true for all $c' < c$, and let $a_1, \ldots, a_k, c$ be as above. Consider $c - a_k$. Then $a_1 + \ldots + a_{k-1} \geq c - a_k$ Also a_{k-1} divides $c - a_k$ because a_k divides c and a_{k-1} divides a_k. So, by induction hypothesis applied to $c' = c - a_k$, we see that there is a subset of $\{a_1, \ldots, a_{k-1}\}$ which adds to $c - a_k$, and so, by adding a_k to this subset, we get c .

   Now we easily apply this to the question at hand. Suppose a subset of coins adds up to M and let c_i be the largest denomination at most M. If the optimal solution does not use c_i, all its coins are of value less than c_i. But the statement above shows that there is a subset of coins adding upto exactly c_i. In this case, we can replace these coins by one coin of value c_i, contradicting the optimality of the solution .

   Common Mistakes: Many of you completely missed the proof that a subset of coins in any solution can be replaced by c_i. Some of you showed that a multiple of a single lower denomination coin can be replaced by c_i, however logically it is not true in every situation. Suppose you have m coins of denomination c_j and c_i is n times c_j, where $m < n$, you can't really replace $m \cdot c_j$ with c_i.

3. You have n workers. Worker i has skill s_i. You are given a target T. You need to divide the workers into teams of two workers each (assume n is even, and so there will be $n/2$ teams). A team consisting of workers i and j is said to be good if $s_i + s_j \geq T$. Devise an efficient greedy algorithm to divide the workers into teams of two workers each such that the number of good teams is maximized. Prove its correctness in two steps: (i) prove that there is an optimal solution which agrees with the greedy algorithm on the first choice, and (ii) use induction for rest of the input.

   The algorithm is the following: suppose $s_1 \geq s_2 \geq \ldots \geq s_n$. Then let s_k be the smallest value such that $s_1 + s_k \geq T$. Pair workers 1 and k, and recursively solve the problem on $2, 3, \ldots, k-1$.

   We prove the first statement. Suppose the greedy algorithm pairs $(1, k)$. Consider an optimal algorithm, which pairs $(1, j)$, where $j \neq k$. Further assume that the optimal algorithm pairs (k, j') (here j or j' could be empty,

in which case 1 or k are unpaired) . Now we replace these two pairs by $(1, k)$ and (j, j') and prove that both these are valid pairs (assuming j, j' are not empty, else we replaced 1 pair by 1 new pair) . The fact that $(1, k)$ is valid is by assumption. Now $s_j + s_{j'} \geq s_k + s_{j'} \geq T$, where the first inequality follows from the fact that $s_j \geq s_k$ by definition of s_k, and the second inequality follows from the fact that (k, j') is a valid pair . Thus we replaced two pairs in the optimal solution by two new pairs which include $(1, k)$.

The second part is by induction on n: the greedy algorithm is optimal for any set of n workers. Base case when $n = 2$ is easy. So assume it is true for less than n workers. Now, suppose the greedy algorithm pairs $(1, k)$. Clearly, we cannot pair $k+1, \ldots, n$ with any other worker, so we can remove them. Also we have show above that there is an optimal solution which pairs $(1, k)$. Now, the remaining pairs in this optimal solution are formed out of $2, \ldots, k-1$, and by induction hypothesis, are at most as many pairs as the greedy algorithm. Thus, overall the number of pairs in the greedy algorithm is at least that in the optimal solution .

Common Mistakes: Many of you wrote started with saying that suppose the first pair in the optimal solution is not same as the first pair in the greedy algorithm. This does not make sense because the optimal solution does not have a notion of which pair came first. It is just a set of pairs.

Another common mistake was that if (i, j) is the first pair in the greedy algorithm, and the optimal solution picks a pair (i, j') where $j \neq j'$, then we can throw away j' and add (i, j). This is wrong, because may be j was also paired with some other worker i' in the optimal solution, and if you are creating (i, j), then you are destroying two pairs in the optimal solution, (i, j') and (i', j). So you need to argue that (i', j') is also a valid pair.

Many people tried to do the induction proof without writing what the induction hypothesis is, this is also wrong.

4. You are given an undirected connected graph G with n vertices and $n + 10$ edges. Each edge has a positive weight, and you can assume that the weights are distinct. Give an $O(n)$ time algorithm to find the minimum spanning tree of G. Justify why your algorithm is correct.

   We first make the following observation: if C is a cycle in the graph and e is the edge with the largest weight, then e cannot be present in the MST . To prove this, suppose for the sake of contradiction, that the MST T contains the edge $e = (u, v)$. Let P be the path (other than e) from v to u in C. When we remove e from T, it splits into two connected components . Since u and v lie on two different connected components, there must be an edge f in P which has its end-points in two different components. So by remove e and adding f to T, we get a cheaper tree, which is not possible. So T cannot contain e .

   So we get the following algorithm: find a cycle in G and remove the most expensive edge . The graph stays connected, and after 11 steps, it has $n-1$ edges, and so must be the MST by the argument above . Each such step takes $O(n)$ time since finding a cycle can be done in $O(n)$ time.

   Common Mistakes: Many of you have tried to implement Kruskal's algorithm using counting sort(which is wrong since they don't always execute in $O(n)$ time, the running time depends on the encoding length of the numbers). Further, the cycle detection step in Kruskal (using Union-Find) won't execute in $O(n)$ time.