CSCI 570 Summer 2016 Homework 2 Solutions

On the first page of your homework, please write your name and USC ID # clearly. Please do something to indicate which is your last (family) name, such as underlining it. If your submission is multiple pages, you must *staple* them together. These are requirements of every homework assignment this semester.

These two problems require a *greedy* algorithm. Please strive to describe your algorithm concisely; for example, for the scheduling with deadlines example from lecture, a sufficient description of the algorithm would be "sort in non-decreasing order of deadline." You do not need to write pseudo-code or provide the running time (although both problems can be solved in $O(n \log n)$ time; please don't make yours significantly worse).

In addition to providing the algorithm, prove that your algorithm is correct and optimal. Both of these can be proven with relatively short proofs; you shouldn't need multiple pages.

1. Let X be a set of n intervals on the real line (each has a start coordinate s_i and an end coordinate f_i). A subset of intervals $Y \subseteq X$ is called a *tiling path* if the intervals in Y "cover" the intervals in X; that is, any real value that is contained in some interval in X is also contained in some interval in Y. The size of a tiling cover is just the number of intervals; our goal is to find the smallest tiling cover.

   Give an efficient **greedy algorithm** to solve this problem and prove it finds the optimal sized cover.

- Algorithm: Of all intervals that start at the start time, take the one that ends last. For the second (and future) intervals, choose the latest ending interval out of those that start no later than the finish time of the previous selection. If there are multiple independent subsets of intervals, repeat this for each subset.

  This can be implemented in time $O(n \log n)$ by sorting by start time, using later finishing times as a tie breaker (later finishing time among equal starts should be *earlier* in the sort). Select the first interval, and then keep track of the finishing time of the last selected interval; whenever you encounter an interval that starts later than this finish time, take the latest finishing interval of the just-explored subset (you can track this max element along the way).

- Proof of correctness: because this is a covering problem, we will prove that there exists an optimal solution that includes the first chosen interval. We can imagine the above algorithm as taking any overlapping intervals and changing their start times to be the finish time of the selected interval before recursively solving on the remaining subset; the effect is the same, in both coverage and intervals selected.

  Suppose we are wrong in our claim and there does not exist an optimal solution (defined, as always for a covering problem, as any subset that achieves the covering goal but for which a smaller set cannot achieve this goal) that includes this first interval. Let OPT be any optimal solution, and let g be the earliest-finishing interval it selects. Refer to our first selection as f.

  Note that g and f must start at the same time: f has the earliest start of any interval, and if g starts any later than that, then OPT is an invalid solution, as there would be a value covered by an interval that is not covered by the solution. You might argue that this doesn't apply inductively: strictly speaking, it doesn't; however, on subsequent selections, thinking of both f and g as having any portions of their segment removed that overlap with previously selected intervals doesn't change the solution nor its correctness in any way.

  Note also that g cannot end later than f, again by definition of our algorithm (we're selecting the latest ending of all intervals in the set that includes both f and g). Therefore, because g starts (or effectively starts) at the same time as f, and ends no later than f, anything covered by g is also covered by f. We can therefore form a second set, OPT2, by starting with OPT, removing g, and adding f. This gives us a valid set (because anything not covered by OPT-$\{g\}$ is covered by f) of equal size to an optimal set: this makes OPT2 an optimal set.

2. We're asked to help the captain of the USC tennis team to arrange a series of matches against UCLA's team. Both teams have n players; the tennis rating (a positive number, where a higher number can be interpreted to mean a better player) of the ith member of USC's team is t_i and the tennis rating for the kth member of UCLA's team is b_k. We would like to set up a competition in which each person plays one match against a player from the opposite school. Because we get to select who plays again whom, our goal is to make sure that in *as many matches as possible*, the USC player has a higher tennis rating than his or her opponent. Give an efficient **greedy** algorithm for this problem and prove that it is correct.

Algorithm: begin by sorting both teams' rosters by tennis rating. My explanation in these solutions will begin with the highest-ranked USC player, although similar algorithms exist that start with the lowest-ranked USC player, the highest ranked UCLA player, and the lowest ranked UCLA player.
For each Trojan player, starting at the highest rated, pair him or her up against the best Bruin player that has a lower rating; in other words, the best possible opponent that we will still beat. If there are no such opposing players, pair our player up against the best overall opposing player.
This can be accomplished in $O(n \log n)$ time by first sorting and then performing a binary search to find an opponent.

Proof of correctness: this is a permutation problem; imagine it as fixing the order of one roster and permuting the other roster to achieve the best possible win count. As suggested in lecture, a good proof for a permutation problem tends to be via exchange argument.
Consider any alternate solution ALT. Fix the order of the Trojan players in both our solution and ALT to be highest-to-lowest ranked (permuting the Bruins players list to correspond to the pairings produced). Renumber the Bruins' players so that our matching can be described as all pairs of Trojan i against Bruin i.
Now, let's look at the pairings our algorithm produces and that ALT has selected. There exists at least two values for i for which Trojan i is not paired against Bruin i in ALT's solution; let's talk about the smallest such value of i (the first disagreement). That is, Trojan i is the *best* USC player for whom our algorithm and ALT have selected a different opponent. In ALT's solution, Trojan i is paired against Bruin k, and Bruin i is paired up against Trojan j. We do not know if $j = k$ or not (nor does it matter). If we can show that swapping Bruins k and i in ALT's solution produces an alternate arrangement that is no worse than ALT, we will have shown that our pairings are optimal (as any alternate solution that doesn't fully agree with us can have one more agreement without being made worse; because this is always true, we can transform any alternate solution into ours without making it worse at any point, and thus no pairings exist that are strictly better than ours).
Note that $t_i > t_j$ by our definition of i.
These two matches produce either two wins, one win, or zero wins in ALT.

- Suppose the pairings produce two wins. We now know that $t_i > b_k$ and $t_j > b_i$, because our school won both of those matches. We also know that $b_k < b_i$ – because t_i is capable of winning a match, she must be paired in our solution against the best Bruin player not already claimed by a better teammate. The Bruins claimed by a better teammate than Trojan i (in both our solution and ALT) are $b_1 \dots b_{i-1}$. Therefore, Bruin i has a higher rating than any other Bruin who Trojan i would beat.

  After making the swap in ALT, we form ALT$'$. In this, Trojan i takes on Bruin i and wins: $t_i > t_j > b_i$. Trojan j takes on Bruin k and wins also: $t_j > b_i > b_k$. As such we have two wins in ALT$'$, which is no fewer than in ALT. So this situation doesn't make it any worse.
- Suppose the pairings produced zero wins in ALT: then ALT$'$ clearly doesn't cost us any wins, because it will have 0, 1, or 2 wins, each of which is no worse than zero.
- What if there is one win in ALT? Either Trojan i won or Trojan j won. Let's examine each case. If Trojan i won in ALT, then Trojan i is capable of beating some Bruin not numbered less than i; because our algorithm paired her against Bruin i, this must be a winning match (because our algorithm ensures that any Trojan who can win when selection comes up will always be assigned a match he or she can win).

  What is Trojan j is winning in ALT but Trojan i is not? In ALT, Trojan j was winning against Bruin i, who is now Trojan i's opponent. However, $t_i > t_j > b_i$, so Trojan i can handle Bruin i.