Homework 5

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it.

Problem 1: Greedy Choice

You are consulting for a trucking company that does a large amount of business shipping packages between New York and Boston. The volume is high enough that they have to send several trucks each day between the two locations. Trucks have a fixed limit W on the maximum amount of weight they are allowed to carry. Boxes arrive at the New York station one by one, and each package i has a weight w_i. The trucking station is quite small, so at most one truck can be in the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise a customer might get upset. At the moment, the company is using a simple greedy algorithm for packing: they pack boxes in the order they arrive, and whenever the next box does not fit, they send the truck on its way.

But they wonder if they might be using too many trucks, and they want your opinion on whether the situation can be improved. Here is how they are thinking. Maybe one could decrease the number of trucks needed by sometimes sending off a truck that was less full, and in this way allow the next few trucks to be better packed.

Prove that, for a given set of boxes with specified weights, the greedy algorithm currently in use actually minimizes the number of trucks that are needed.

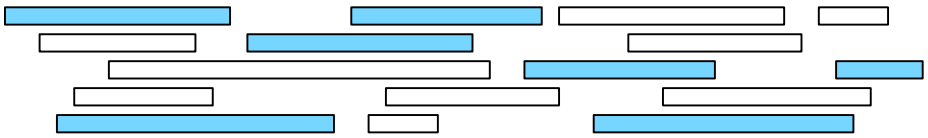
Solution

We prove that the greedy algorithm uses the fewest possible trucks by showing that it "stays ahead" of any other solution. If the greedy algorithm fits boxes $b_1, b_2, \ldots b_j$ into the first k trucks, and the other solution fits $b_1, \ldots b_i$ into the first k trucks, then $i \leq j$. We will prove this by induction on k. The base case is easy; $k = 1$ and we only require a single truck to send all of the packages. Now, assuming the above holds for $k - 1$: the greedy algorithm fits j' boxes into the first $k - 1$ trucks and the alternate (possibly optimal) algorithm fits i' boxes into the first $k - 1$ trucks; $i' \leq j'$. Now for the k^{th} truck, the alternate solution puts in boxes $b_{i'+1}, \ldots, b_i$. Thus since $j' \geq i'$, the greedy algorithm is able at least to fit all the boxes $b_{j'+1}, \ldots, b_i$, if not more into the k^{th} truck.

Problem 2: Another Interval Problem

Let X be a set of n intervals on the real line. 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.

Describe an algorithm to compute the smallest tiling path of X. Assume that your input consists of two arrays $X_L[1..n]$ and $X_R[1..n]$, representing the left and right endpoints of the intervals in X. Argue that your algorithm is correct (hint: remember that an argument of correctness for any greedy algorithm has two components). The figure below shows an example of a tiling path, though a non-optimal one.



A set of intervals. The seven shaded intervals form a tiling path.

Solution

Sort both X_L and X_R so that the jobs with the earliest start times are first. Include the first interval (the one with the earliest start time). It has to be in your tiling. If there are multiple intervals with the same start time (and they're all the earliest start time), choose the one that has the lastest finish time (after all, it gets you the most bang for your buck). Then remove everything that is overlapped completely by the interval you just selected. (There's no use including any of these since they don't add anything. Now recurse. After the first subproblem, you want to look at all of the intervals that include the finish time of the interval you just included. Of these intervals, you want to pick the one of them that has the latest finish time. That is, say the greedy choice in the previous step was to include interval i. After removing all of the intervals that are completely overlapped by interval i, the next greedy choice is an interval that starts on or before $X_R[i]$ and finishes the latest of all such intervals.

Optimal Substructure. The problem exhibits optimal substructure. After making the greedy choice and removing all of the intervals that are completely overlapped by the greedily chosen interval, we're left with a smaller problem (the original set X minus some intervals) and a smaller piece of the line to cover (specifically, we're left with the portion of the line that is NOT covered by the greedily chosen interval).

The Greedy Choice is Good. Suppose that the greedy choice was not good. That is, suppose that you hand me an optimal tiling path that does not contain the earliest starting interval (and of the earliest starting intervals the one with the latest finish time). I can take your tiling path and remove the interval that has the earliest starting time in it and replace it with the greedy choice. Either my interval starts before yours (in which case, your solution wasn't correct, since you didn't cover the entire line) or they start at the same time. In the latter case, I've only covered *more* of the line since my selected interval has a later finishing time than yours (since I selected the earliest starting one with the latest finishing time). Either that or our intervals were exactly the same.

Problem 3: Coin Changing

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

(a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution.

Solution

Select the largest coin denomination that is less than n. Let this denomination be c_k. Use one of these coins. Repeat for $n - c_k$ cents.

Proof that the greedy algorithm is optimal: At a step of the algorithm, we say a particular choice of coin is safe if it preserves the invariant that the coins chosen so far form a subset of an optimal solution. We need to show that every choice the greedy algorithm makes is safe, which will then guarantee that the final set of coins will be an optimal solution. (Similar proof strategy as in the MST algorithms.)

We will argue that an optimal solution for $n \geq 25$ must include at least 1 quarter. Thus the greedy choice in this case is safe. Similarly, we'll argue that an optimal solution for $10 \leq n < 25$ must include at least one dime, and that an optimal solution for $5 \leq n < 10$ must include at least one nickel, implying that the greedy choice in every case is safe.

Note the following properties of optimal solutions: No optimal solution can include 5 pennies. Similarly, no optimal solution can include 2 nickels, or 1 nickel and 2 dimes. (These can be replaced with 1 nickel, 1 dime, and 1 quarter, respectively.) Finally, we note that no optimal solution can include 3 dimes because this can be replaced with 1 quarter and 1 nickel. This implies that the largest optimal solution not including quarters is for $n = 24$ (2 dimes + 4 pennies). Thus an optimal solution for $n \geq 25$ must include at least 1 quarter. Similarly, the largest optimal solution not including quarters or dimes is for $n = 9$ (1 nickel + 4 pennies). Thus an optimal solution for $10 \leq n < 25$ must include at least one dime. Finally, the largest optimal solution not including quarters, dimes, or nickels is for $n = 4$ (4 pennies). Thus an optimal solution for $5 \leq n < 10$ must include at least one nickel.

(b) Give a set of coin denominations for which your greedy algorithm does not yield an optimal solution. Your set should include a penny to ensure that you can always successfully make change.

Solution

One example is a scheme with a 1 cent coin, a 6 cent coin, and an 8 cent coin. To make 12 cents, the greedy algorithm would use 5 coins, while the optimal algorithm would use 2.

Problem 4: Phone Base Stations

Consider a long, quiet country road with houses scattered sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint.) Further, let's suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within 4 miles of one of the base stations.

(a) Give an efficient algorithm that achieves this goal, using as few base stations as possible.

Solution

Start at the western end of the road and begin moving east until the first moment when there's a house h exactly four miles to the west. We place a base station at this point (if we went any further east without placing a base station, we wouldn't cover h). We then delete all of the houses covered by this base station and iterate on the remaining houses.

For any point on the road, define its *position* to be the number of miles it is from the western end. We place the first base station at the easternmost point (i.e., largest position) s_1 with the property that all houses between 0 and s_1 will be covered by s_1. In general, having placed $\{s_1, \ldots, s_i\}$, we place base station $i+1$ at the largest position s_{i+1} with the property that all houses between s_i and s_{i+1} will be covered by s_i and s_{i+1}.

(b) Prove that the greedy choice that your algorithm makes is the optimal choice.

Solution

Let $S = \{s_1, \ldots, s_k\}$ denote the full set of base station positions that our greedy algorithm places, and let $T = \{t_1, \ldots, t_m\}$ denote the set of base station positions in an optimal solution, sorted in increasing order (i.e., from west to east). We must show that $k = m$.

We do this by showing a sense in which our greedy solution S "stays ahead" of the optimal solution T. Specifically, we claim that $s_i \geq t_i$ for each i, and prove this by induction. The claim is true for $i = 1$ since we go as far as possible to the east before placing the first base station. Assume now that it is true for some value $i \geq 1$; this means that our algorithm's first i centers $\{s_1, \ldots, s_i\}$ cover all of the houses covered by the first i centers $\{t_1, \ldots, t_i\}$. As a result, if we add t_{i+1} to $\{s_1, \ldots, s_i\}$, we will not leave any house between s_i and t_{i+1} uncovered. by the $(i+1)^{st}$ step of the greedy algorithm choose s_{i+1} to be *as large as possible* subject to the condition of covering all houses between s_i and s_{i+1}, so we have $s_{i+1} \geq t_{i+1}$. This proves the claim by induction.

Finally if $k > m$, then $\{s_1, \ldots, s_m\}$ fails to cover all houses. But $s_m \geq t_m$, and so $\{t_1, \ldots, t_m\}$ also fails to cover all houses—a contradiction to it being the optimal solution.

Problem 5: Phone Base Stations around the Lake

Remember the problem with the phone base stations. Here we will see a variation of this problems. This time there are houses scattered sparsely along the perimeter of a cyclic lake with radius far longer than 4 miles. The positions of the houses are given to you. Your goal is to place as few phone base stations as possible, along the perimeter, ensuring that every house is within 4 miles from some base station (you can assume that every phone base station is covering an arc of 8 miles with its position as the middle point of the arc). Give an algorithm with time complexity $O(n^2)$

Solution

Here we will work in a similar way as in the case where the houses are on a line. To do so we have to decide from where to begin. However, choosing different starting point can affect the optimality of our solution and thus we have to try all the possible n starting points.
To this end we define our first starting point by placing the first base station 4 miles to the right of the house with index 1. We continue using the greedy algorithm as we saw in the line-case and we keep the total number of base stations we needed.
We repeat the same process increasing the index of the house which we use as a starting point. Taking the best of all these tries will give us the optimal solution.
The time complexity is $O(n)$ repeated n times i.e. $O(n^2)$. The correctness of the algorithm follows from the proof of correctness we gave for the line-case and the fact that we checked all the starting points that could result in different outcomes (easy to see why).