Lab 10

Greedy algorithms

Collaboration level 0 (no restrictions). Open notes.

1. From class: **A greedy pharmacist problem:** A pharmacist has W pills and n empty bottles, where all bottles cost the same and bottle i can hold p_i pills. The goal is to find the minimum cost for storing the W pills using the bottles.

   Greedy algorithm: Pick the bottles in decreasing order of p_i until they can cumulatively store W pills.

   Here we want to justify correctness. To prove that a greedy algorithm is correct it is sufficient to prove that there exists an optimal solution that includes the first greedy choice. Once you prove this claim about the first greedy choice, you can apply it repeatedly to all subsequent choices, and it will follow that there exists an optimal solution consisting entirely of greedy choices — this part can be skipped in an informal justification.

   Claim: There exists an optimal solution that includes the bottle with largest capacity.

   Proof:

2. **Art gallery guarding:** In the *art gallery guarding* problem we are given a line L that represents a long hallway in an art gallery. We are also given a set $X = \{x_0, x_1, x_2, ..., x_{n-1}\}$ of real numbers that specify the positions pf paintings in this hallway; assume that each painting is a point. Suppose that a single guard can protect all the paintings within a distance at most 1 of his or her position, on both sides. You can assume that the positions of the paintings are sorted.

   Design an algorithm for finding a placement of guards that uses the minimum number of guards to guard all the paintings in X. Argue why your algorithm is correct and analyze its running time.

   We expect: The algorithm as pseudocode, a brief explanation, analysis and justification of correctness i.e. why does it always assure the minimum nb of guards? Remember that to show that a greedy algorithms is correct it is sufficient to show that there exists an optimal solution that includes the first greedy choice.)

Additional/optional problems

1. **Thanksgiving Turkeys:**[1] On Thanksgiving day, you arrive on an island with n turkeys. You've already had Thanksgiving dinner so you don't want to eat the turkeys (and maybe you prefer tofurkey anyway), but you do want to wish them all a Happy Thanksgiving. However, the turkeys each have very different sleep schedules. Turkey i is awake only in a single closed interval $[a_i, b_i]$. Your plan is to stand in the center of the island and say loudly "Happy Thanksgiving!" at certain times $t_1, ..., t_m$. Any turkey who is awake at one of the timest_j will hear the message. It's okay if a turkey hears the message more than once, but you want to be sure that every turkey hears the message at least once.

   Design a greedy algorithm which takes as input the list of intervals $[a_i, b_i]$ and outputs a list of times $t_1, ..., t_m$ so that m is as small as possible and so that every turkey hears the message at least once. Your algorithm should run in time O(nlog(n)).

   [We are expecting: Pseudocode and an English description of the main idea of your algorithm, as well as a short justification of the running time.]

   Hint: This is very *similar to a problem discussed in class. Can you see how?*

2. **Matching points on a line:**[2] You are given two arrays of n points in one dimension: red points $r_1, r_2, ..., r_n$ and blue points $b_1, b_2, ..., b_n$. You may assume that all red points are distinct and all blue points are distinct. We want to pair up red and blue points, so that each red point is associated uniquely with a blue point and vice versa. Given a pairing, we assign it a score which is the sum of distances between each pair of matched points.

   Find the pairing (matching) of minimum score. Hint: Aim for an $O(n \lg n)$ algorithm. Draw examples for small valus of n to get intuition.

   Example: Consider the input where the red points are $r_1 = 8, r_2 = 1$ and the blue points are $b_1 = 3$ and $b_2 = 9$. There are two possible matchings:

   - match r_1 to b_1 and r_2 to b_2: the score of this matching is $|r_1 - b_1| + |r_2 - b_2| = |8 - 3| + |1 - 9| = 13$
   - match r_1 to b_2 and r_2 to b_1: the score of this matching is $|r_1 - b_2| + |r_2 - b_1| = |8 - 9| + |1 - 3| = 3$

   The algorithm shoud return the cost of the optimal matching, which is 3.

   Hint: Think greedily, and argue with an exchange argument.

[1]Credit: Problem reproduced from Stanford, cs161
[2]Credit: reproduced from Stanford University, cs161