CS 577: Introduction to Algorithms Fall 2020

Homework 5

Instructor: Dieter van Melkebeek TA: Ryan Moreno

This homework covers the greedy-stays-ahead-paradigm. **Problems 1 and 3 must be submitted for grading by 11:59pm on 10/19**. Please refer to the homework guidelines on Canvas for detailed instructions.

Warm-up problems

1. You are given a knapsack with a weight limit of W and n items with nonnegative weights $w_1, w_2, ..., w_n$. You want to fill your knapsack as close to the weight limit W as possible but without exceeding it. You can see this as a specific case for the general knapsack problem for which each item has the same value as its weight.

   Consider the case where the weight of each item is at least as large as the weight of all previous items combined, i.e. $w_i \geq \sum_{j=1}^{i-1} w_j$ for each $1 < i \leq n$. Design an algorithm to solve this problem that performs no more than $O(n)$ elementary operations. Arithmetic operations like the addition of two numbers and the comparison of two numbers are considered elementary for this problem. Your algorithm should output an optimal list of items to put in your knapsack.

2. 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 a number of 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 at the station at any time. Company policy requires that boxes are shipped in the order they arrive; otherwise, a customer might get upset upon seeing a box which arrived later than their own make it to Boston faster. 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.

   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.

Feedback problem

3. You are going to compete in a puzzle competition. There will be n different puzzles released throughout the competition; the i-th puzzle is released at time r_i. Once a puzzle has been released, you can access it whenever you choose to, but you must complete the puzzles in order. You will have t minutes to complete a puzzle once you begin and you cannot move on to the next puzzle (or nap) until those t minutes are complete. Because the puzzles are

mentally taxing, you can only do up to k puzzles in a row before requiring an m minute nap. You have just received the schedule of events, a list of puzzle titles ordered by the time they will be released. You are not only awarded points based on your solutions, but also based on how fast you finish the competition. So, you need to design an algorithm that computes your schedule for napping and problem solving such that you finish the competition at the earliest time possible. Your algorithm should take O(n) time.

Example: Consider the case when there are $n = 3$ puzzles, you can complete up to $k = 2$ consecutive puzzles before napping for $m = 30$ minutes, and each puzzle will take $t = 15$ minutes. You are given the following schedule: [("Puzzle 1", 1pm); ("Puzzle 2", 2:30pm); ("Puzzle 3", 3pm)]. In this case, an optimal schedule is: [1-1:15pm: solve puzzle 1, 1:15-1:45pm: nap, 2:30-2:45pm: solve puzzle 2, 3pm-3:15pm: solve puzzle 3]. An alternate schedule that is **not** optimal is: [1-1:15pm: solve puzzle 1, 2:30-2:45pm: solve puzzle 2, 2:45-3:15pm nap, 3:15pm-3:30pm: solve puzzle 3].

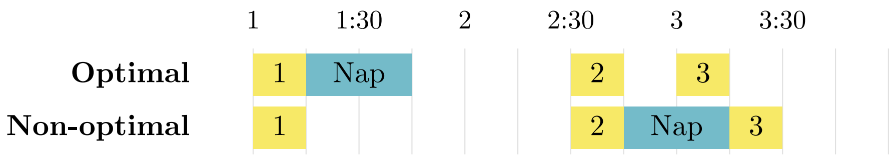


Figure 1: Possible Competition Schedules

Additional problems

4. The year is 1922. You are organizing a big dance in your university and you want to invite everyone. Unfortunately, emails and cellphones do not exist so you have to visit each class during its lecture to invite the students to the event. The lecture time of each class is a half-closed interval $(s_i, f_i]$ of a start and finish time. To minimize your trips to the university you try to visit it during times where many classes *overlap* and invite the students of all these classes. For simplicity we assume that the start and finish times of the lectures are non-negative rational numbers. One possible schedule of lectures could be

$$(1, 3.1], (2.2, 4], (1.7, 5], (4, 5]$$

In this case the smallest number of visits to the university is 2, for example one visit at 2.5 and one at 4.5. Visiting at 4 does not cover the (4, 5] class. However, it does cover the (2.2, 4] class.

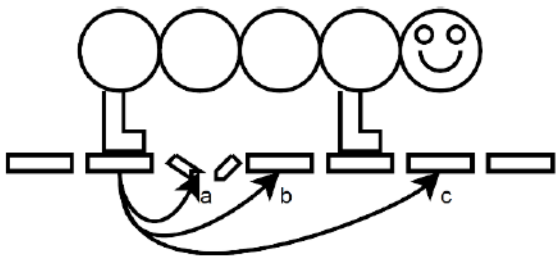
   (a) You decide that you should pick your first visit to be a time that overlaps with the maximum number of lectures, that is a number t that is contained in the maximum number of intervals $(s_i, f_i]$. Then you remove these intervals and pick your second visit with the same rule. You continue until no more lectures are uncovered. Construct a counter-example where this greedy strategy fails to compute an optimal solution.

   (b) Design a greedy algorithm that computes the smallest number of visits to the university that cover all lectures. Your algorithm should run in time $O(n \log n)$.

5. For the opening scene of a computer game, you want the main character, Wormly, to cross a bridge. Wormly is a worm made of k equal circular bubbles and ℓ legs. At all times each leg has to be under one of the bubbles, and under each bubble there can be at most one leg. The bridge was supposed to be composed of n planks with the width of each plank equal to the diameter of each of Wormly's bubbles. However, some of the planks are missing.

   At every moment, Wormly can do exactly one of the following:

   - Move one of its legs forward over any number of (possibly missing) planks. After the move, the leg should be on a plank and underneath one of Wormly's bubbles. A leg isn't allowed to overtake other legs.
   - Move all of its bubbles forward one plank while its legs remain on the same planks. After the move each leg must still be under one of Wormly's bubbles.



   In the above figure, the only possible move for the last leg is to position b. This is because the plank at position a is missing, so the leg cannot move there; to get to position c, the last leg would have to overtake the first leg. Also, in this example, moving all the bubbles forward is not allowed because Wormly's last leg would end up without a bubble over it.

   Initially Wormly's bubbles are directly above the leftmost k planks of the bridge and its legs are on the leftmost ℓ planks. At the end of the animation Wormly's bubbles have to be directly above the rightmost k planks and its legs have to be on the rightmost ℓ planks. The left- and rightmost ℓ planks of the bridge are not missing.

   Design an algorithm to determine the smallest number of steps for the animation when given zk, ℓ, and a binary string of length n where the ith bit indicates whether the ith position has a plank. Your algorithm should run in time $O(n)$.

Challenge problem

6. In some courses you can choose a certain number k of the n assignments that will be dropped in the calculation of your grade. If all the assignments counted equally, the choice would be easy: simply drop the assignments with the lowest scores. However, each assignment may have a different maximum score. Your final homework grade will be the percentage ratio of your total score to the maximum possible score for the retained assignments. This leads to the following problem. You are given a value of k and a list of n assignment results (s_i, m_i), $1 \leq i \leq n$, where s_i denotes your score on the ith assignment and m_i denotes the maximum possible score on that assignment. Your goal is to find a set $I \subseteq \{1, 2, \ldots, n\}$ with $|I| = k$ such that $\sum_{i \notin I} s_i / \sum_{i \notin I} m_i$ is as large as possible.

Design an algorithm that runs in time $O(n^2 \log n)$. For starters, aim for an algorithm that runs in time polynomial in the number of bits in the input.

Programming problem

7. SPOJ problem I AM VERY BUSY (problem code BUSYMAN).