CS4102 - ALGORITHMS - SPRING 2022

DUE FRIDAY, APRIL 22, 2022 AT 11:30 PM VIA GRADESCOPE

CADV(HZ9XS)

Collaboration Policy: You are encouraged to collaborate with up to 3 other students, but all work submitted must be your own *independently* written solution. List the computing ids of all of your collaborators in the `collabs` command at the top of the tex file. Do not share written notes, documents (including Google docs, Overleaf docs, discussion notes, PDFs), or code. Do not seek published or online solutions for any assignments. If you use any published or online resources (which may not include solutions) when completing this assignment, be sure to cite by naming the book etc. or listing a website's URL. Do not submit a solution that you are unable to explain orally to a member of the course staff. Any solutions that share similar text/code will be considered in breach of this policy. Please refer to the syllabus for a complete description of the collaboration policy.

Collaborators: hl8zd, xs7tng

Sources: Cormen, et al, Introduction to Algorithms. *(add others here)*

PROBLEM 1 *Backpacking*

You are going on a backpacking trip through Shenandoah National park with your friend. You two have just completed the packing list, and you need to bring n items in total, with the weights of the items given by $W = (w_1, w_2, \ldots, w_n)$. You need to divide the items between the two of you such that the difference in weights is as small as possible. The total number of items that each of you must carry should differ by at most 1. Use dynamic programming to devise such an algorithm, and prove its correctness and running time. You may assume that M is the maximum weight of all the items (i.e., $\forall i,\ w_i \leq M$). The running time of your algorithm should be a polynomial function of n and M. The output should be the list of items that each will carry and the difference in weight.

Solution: Algorithm: Consider function $S(j,k,d)$, which returns true if from among the first j items, k are assigned to you, the total weight of your backpack minus the total weight of your friend's backpack is exactly d. j represents for the current number of items that have already been put into the backpack. k represents for the current number of items in your backpack. d represents for the current weight of your backpack minus the current weight of your friend's backpack. We will use two item lists to store the items in your backpack and your friend's backpack. $S(j,k,d) = S(j-1,k-1,d+w_j) \vee S(j-1,k,d-w_j)$. Initialize $S(0,0,0)$ = True. If the last thing is brought by you, add that item to your list. Otherwise, add that item to your friend's list. For this problem, we just need to calculate all true cases for $S(n, \frac{n}{2}, |d| < \frac{nM}{2})$. After we find a true case, we will need to store the difference d in a list. After we calculate all d for every cases, compare them and choose the smallest d value to be the difference in weight. And find its corresponding item lists.

Prove the correctness: so basically we are trying to find the smallest difference between two people by laying out all possible outcomes using dynamic programming and compare them to get the smallest one. For the DP part, there are only two possible "last thing done" for this algorithm: either the item is put into your bag or the item is put into your friend's bag, resulting in two different outcomes for d(the difference between your weight). For function $S(j,k,d)$, j and k are set by n and $n/2$ as the problem asks, while d could vary from $-\frac{nM}{2}$ to $\frac{nM}{2}$. So, what we need to do is to list out all possible outcomes and find the smallest.

Prove the runtime: the loop needed for this dynamic approach is $n * n * nM$, which causes $\Theta(n^3 M)$ runtime. Because M is a value rather than the size, the runtime would be pseudo-polynomial.

PROBLEM 2 *Course Scheduling*

The university registrar needs your help in assigning classrooms to courses for the fall semester. You are given a list of n courses, and for each course $1 \leq i \leq n$, you have its start time s_i and

end time e_i. Give an $O(n \log n)$ algorithm that finds an assignment of courses to classrooms which minimizes the *total number* of classrooms required. Each classroom can be used for at most one course at any given time. Prove both the correctness and running time of your algorithm.

Solution:

Algorithm: First, we need to sort all courses based on their start time using merge sort. Then, pick the course that has the earliest start time. We assign a classroom for that course. Create a min heap(sorted by the end time of the last course in that classroom node) and put the classroom as the root node of the min heap. Next, we pick the course with the second earliest start time. If its start time is after the end time of the last course in the classroom of the root node of the heap (which means the second course is not conflict with the first one), we just simply add the course to the classroom at the root node of the heap, and update the end time of the classroom as the end time of the course added, and percolate-down the root node. If the second course selected has conflict with the course in the root classroom, we need to create a new classroom to hold that course and insert that classroom to the min heap. Go over the other courses in the list just as we did to the second course until all courses are assigned to a classroom.

Prove the correctness: We use a min heap to keep tracking the earliest end time of all classrooms. Whenever we assign a new course, we will check it with the classroom with earliest end time. If the classroom with earliest end time has conflict with the new course, then we must have a new classroom for that course because all other classrooms can not hold that course as well. If the classroom with earliest end time can hold that new course, that is what we want, just add that course to the most available classroom you can find for now. This greedy algorithm works perfectly find here.

Prove the runtime: We need $nlog(n)$ for merge sort. Inserting or updating the min heap n times is $nlog(n)$. So, the overall runtime would be $\Theta(nlogn)$.

PROBLEM 3 *Ubering in Florin*

After the adventures with Westley and Buttercup in *The Princess Bride*, Inigo decides to turn down the "Dread Pirate Roberts" title and to instead moonlight as the sole Uber driver in Florin. He usually works after large kingdom-wide festivities at the castle and takes everyone home after the final dance. Unfortunately, since his horse can only carry one person at a time, he must take each guest home and then return to the castle to pick up the next guest.

There are n guests at the party, guests $1, 2, ..., n$. Since it's a small kingdom, Inigo knows the destinations of each party guest, $d_1, d_2, ..., d_n$ respectively, and he knows the distance to each guest's destination. He knows that it will take t_i time to take guest i home and return for the next guest. Some guests, however, are very generous and will leave bigger tips than others; let T_i be the tip Inigo will receive from guest i when they are safely at home. Assume that guests are willing to wait after the party for Inigo, and that he can take guests home in any order he wants. Based on the order he chooses to fulfill the Uber requests, let D_i be the time he returns from dropping off guest i. Devise a greedy algorithm that helps Inigo pick an Uber schedule that minimizes the quantity:

$$\sum_{i=1}^{n} T_i \cdot D_i.$$

In other words, he wants to take the large tippers the fastest, but also want to take into consideration the travel time for each guest. Prove the correctness of your algorithm. (Hint: think about a property that is true about an optimal solution.)

Solution:

Algorithm: Calculate T_i/t_i respectively for each guest. Sort all guests based on the value we just calculated from T_i/t_i. Take guests home in order of decreasing T_i/t_i value.

Prove the correctness: We want to take the large tippers first, but also need to consider about the travel time for each guest. Intuitively, we just need to calculate the cost-performance ratio for each guest and take guests home based on that. Here, T_i/t_i is basically the cost-performance ratio for each guest so this algorithm is intuitively correct.
We can also prove this mathematically. Suppose we have two guests: guest1 and guest2. $T_1 = a$ and $t_1 = b$. $T_2 = c$ and $t_2 = d$. Now we want to calculate $\sum_{i=1}^{n} T_i \cdot D_i$. If we pick guest1 first and then guest2, the total sum will be $ab + c(b + d)$. If we pick guest2 first and then guest1, the total sum will be $cd + a(b + d)$. Let's compare these two expression. Minus the first expression by the second, we can get $bc - ad$. If $\frac{a}{b} > \frac{c}{d}$, $bc - ad < 0$ and the first expression will be smaller than the second one, so we like the first expression more, which is the case where we pick guest1 first. Otherwise, $bc - ad > 0$ and the first expression will be larger than the second one, so we like the second expression more, which is the case where we pick guest2 first. In this way, we can see that we should pick the guest with higher T_i/t_i first, which does also make sense in math.

PROBLEM 4 *Gradescope Submission*

Submit a version of this `.tex` file to Gradescope with your solutions added, along with the compiled PDF. You should only submit your `.pdf` and `.tex` files.