CS4102 - ALGORITHMS - SPRING 2022

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

CADV(RTG5XKH)

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: mpm7cf

Sources: Cormen, et al, Introduction to Algorithms.

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: For this DP problem, we will make use of a 3D memory which will contain indices for the number of items assigned to person 1, the number of items assigned to person 2, and the current item to add. Thus $mem[p1, p2, i]$ = the minimum difference between the sum of weights with p1 = number of items for person 1, p2 = number of items for person 2, and i indicating that in this memory entry, all weights from w_1 to w_i, inclusive, are included in the difference calculation. We will define difference specifically, allowing it to be a negative number. If person 2 carries more weight than person 1, the difference number will be negative.
To determine the minimum difference for some configuration of p1, p2, and i, we will need to iterate through all combinations of i, p1, and p2 (with i as the outer loop - assuming a bottom-up implementation), where $p1 \leq i$, $p2 \leq i$, and $1 \leq i \leq n$ (we will of course only check the combinations where p1 and p2 add up to i - there will still be some invalid combinations in the final table but this is ok, we will filter them out at the end). For each of these combinations, we will need to do the following for the current item i:

MinDiff(p1, p2, i) = *closest_to_0*(
  MinDiff(p1-1, p2, i-1) + w_i, minDiff(p1, p2-1, i-1) - w_i
)

To be clear, what above means is: we want to determine the minimum difference in weight possible when person 1 has p1 items, person 2 has p2 items, and the total items being carried consists of the items w_1 through w_i, inclusive. To do this, we can calculate the best of 2 possibilities: if we select item w_i to be carried by person 1, the new difference will be the best difference when person 1 was carrying p1-1 items and person 2 was still carrying p2 items plus w_i. else if we select item w_i to be carried by person 2, the new difference will be the best difference when person 2 was carrying p2-1 items and person 1 was still carrying p1 items *minus* w_i. If these two MinDiff values turn out to be the same, we will choose the one that makes p1 and p2 closest together.

Notice that we add w_i in the first case and subtract it in the second case, because if p2 carries the weight then our difference metric as we defined it decreases, while if p1 carries the weight the difference metric should increase. Also notice that for "minimum" we use the closest value to 0f because this is the value that is the best given our definition of the difference.
We need a base case, which will be when $i == 0$. This means that no items have been assigned, so of course $mem[0,0,0]$ will contain the value 0 to represent this base case.
Once the above is complete, our memory will be filled out enough that we can determine an answer. We will need to look in a specific place in mem to find the answer. To do this, we will iterate through all valid options in mem, which are defined by the problem as those in which the number of items carried by each person differs by at most 1. And, of course, we need to find the places where there are the full n items. Thus we can iterate through the i index of the table where $i == n$, from these options we can find the minimum entry where $p1 == p2$, $p1 == p2 + 1$, or $p1 + 1 == p2$. In this case, by "minimum entry" we actually mean the entry that's closest to 0 because of how we defined how we're storing the differences. The absolute value of this result of the is the minimum difference in weights between the 2 people given the problem constraints.
Time complexity:
$\Theta(n^3 M)$

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: First, this algorithm will sort the courses by their start times.
We will also keep a min priority queue containing classrooms, where the value by which the priority queue is structured is the *latest finish time* of courses in that classroom. We'll call this pq.
At the beginning, we'll insert 1 classroom into pq with no courses and a latest finish time of 0.
Starting from the beginning of the sorted list of courses, we will iterate through each course, i, and assign each one to a classroom in the following greedy manner:
Extract the min of pq. call it classroom c (if there is nothing in pq, insert a new classroom into pq with the course i and a $last\_time$ of e_i) If $s_i >= c.last\_time$, assign i to c, and update $c.last\_time$ to e_i. re-insert the updated course back into pq. Otherwise, insert a new classroom into pq with the course i and a $last\_time$ of e_i. Once we reach the end of this list of courses, all courses will have been assigned, and the number of elements in pq will be the minimum number of classrooms required.
Correctness:
An optimal solution to this problem will contain the minimum number of classrooms needed, which we'll call o. Consider the maximum number of conflicting courses at any one time for a certain input, which we will call cc. Since each of these conflicting courses need their own classroom, $o \geq cc$. Further, we can simplify this expression more to say that $o = cc$, because if cc is the maximum number of conflicting courses at any one time, all other courses could be scheduled around these courses in a manner that would not need any additional classrooms.
Given this, assume that the algorithm does not find o as the number of classrooms needed. This would mean that in a state where o classrooms have already been allocated in the algorithm, another classroom would need to be allocated. However, this is impossible. Since o represents the maximum number of conflicting courses, the minimum last time in pq cannot conflict with the next course that needs to be assigned. This is because if it did, it would mean that there is a number of conflicting courses at one time that exceeds cc. However cc is the maximum number of conflicting courses at one time. This is a contradiction. Thus, the algorithm will allocate o courses

and therefore is correct.
Time Complexity: This algorithm will iterate through the list of n courses, and for each course in the worst case it will run pq.ExtractMin, which is logarithmic, and pq.Insert, which is logarithmic. Thus, in the worst case we are doing a $\Theta(logn)$ operation n times, so the overall time complexity is $\Theta(nlogn)$.

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: Greedily pick guest i such that for all guests remaining, the Tip-to-time ratio $\frac{T_i}{t_i}$ is **maximized**. By doing this we minimize the result of the sum above. We can simply repeat this greedy choice until all guests have been taken home.
Correctness:
Consider an optimal solution to the problem that correctly selects the order in which n guests are taken home such such that the following quantity is minimized:

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

For this optimal ordering, we will say guest 1 is taken home first, guest 2 is taken home second, etc...
Consider two guests, guest k and guest $k+1$ (meaning guest k was taken home right before guest $k+1$ in the optimal ordering).
If $\frac{T_k}{t_k} >= \frac{T_k+1}{t_k+1}$, then the ordering of these two guests matches my greedy choice described above, thus the greedy algorithm's sum will be the same as the optimal sum.
Else if $\frac{T_k}{t_k} < \frac{T_{k+1}}{t_{k+1}}$, we could switch the ordering of these two guests.
Consider the part of the sum that does not include these two guests (the part including the guests before and after these 2), which will not change when their ordering changes (D_i will be the same after the 2 are added regardless). We'll call this quantity C. Thus, with the current ordering in the optimal solution, the sum will be:
$sum_{opt} = C + T_k * (D_{k-1} + t_k) + T_{k+1} * (D_{k-1} + t_k + t_{k+1})$.
If we switch the ordering of these 2 guests, however, which our algorithm would have done, the

sum will instead be this:
$sum_{alg} = C + T_{k+1} * (D_{k-1} + t_{k+1}) + T_k * (D_{k-1} + t_{k+1} + t_k)$.

So, if we can show that: $sum_{alg} < sum_{opt}$,
then we will have proven that the optimal ordering wasn't actually optimal after all, thus our algorithm makes the correct greedy choice and is optimal.

$$sum_{alg} < sum_{opt}$$
$$C + T_{k+1} * (D_{k-1} + t_{k+1}) + T_k * (D_{k-1} + t_{k+1} + t_k) < C + T_k * (D_{k-1} + t_k) + T_{k+1} * (D_{k-1} + t_k + t_{k+1})$$
$$T_{k+1}D_{k-1} + T_{k+1}t_{k+1} + T_kD_{k-1} + T_kt_{k+1} + T_kt_k < T_kD_{k-1} + T_kt_k + T_{k+1}D_{k-1} + T_{k+1}t_k + T_{k+1}t_{k+1}$$
$$T_kt_{k+1} < T_{k+1}t_k$$
$$\frac{T_k}{t_k} < \frac{T_{k+1}}{t_{k+1}}$$

We have reached the same inequality as mentioned above which we know is true in this case (the basis of our Else clause), thus it must also be true that $sum_{alg} < sum_{opt}$.
We could use this same strategy with any other pair of 2 guests, and we would find that our algorithms choice would never be worse and might be better than an optimal ordering of the guests.
Thus, the optimal ordering must select the $\frac{Tip}{time}$ ratios of the guests from highest to lowest (our greedy choice property).

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.