CS4102 - ALGORITHMS - SPRING 2022

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

CADV(YLW4SJ)

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: list collaborators's computing IDs

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: This takes a similar approach to the gerrymandering algorithm we learned in class.

```
Initialize S(0, 0, 0, 0) = true and false elsewhere
S(j, k, x, y) =
j = number of items we have gone through
k = number of items that are assigned to backpack1
x = total weight of backpack 1
y = total weight of backpack 2
for j = 1,..,n:
    for k = 1,...,n/2:
        for x = 0,..jM:
            for y = 0,..,jM:
                S[j - 1, k - 1, x - W[j], y] or S[j - 1, k, x, y - W[j]]
True at S(n, n/2, x, y)
```

To get the solution, iterate though the table and search for True at S(n, n/2, x, y) and find where abs(x - y) is at a minimum. Next, backtrack from there to determine which backpack contains what and add each item to its respective list. Backtracking steps: (Do this until n = 0)

1. If S(n, n/2 - 1, x - W[j], y) = True, assign W[j] to backpack 1
2. If S(n, n/2 - 1, x, y - W[j]) = True, assign W[j] to backpack 2

Return backpack 1 and 2's list and abs(x-y)
Runtime: $O(n^4M^2)$ because the first two for loop are $O(n)$ and the last two for loops are $O(nM)$.

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, sort the courses by increasing start time. Keep these courses in an array called *courseList*. Also maintain a priority list called *classroomList* that keeps track of the end time of the last course added for each classroom. The priority queue is structured so that the classroom with the earliest end time has the most priority. Lastly, create a variable $minRooms = 0$. Next, iterate through *courseList*, assigning each course to a classroom. If $minRoooms = 0$, add a new classroom k to *classroomList* and assign $courseList[i]$ to k and increment *minRooms*. If the start time of $courseList[i]$ is after the end time of the top classroom j in *classroomsList*, assign $courseList[i]$ to j. Else, add a new classroom k to *classroomList* and assign $courseList[i]$ to k and increment *minRooms*. After all courses have been assigned to a classroom, return *minRooms*.

Runtime: First, it takes O($n \log n$) to sort the courses. Next, it takes O($n \log n$) to assign courses to classrooms since in the worst case, every course needs a new classroom so we would need to insert a classroom into the priority queue n times. Insertion into a priority queue is $\log n$. Therefore, the time complexity is O($n \log n + n \log n$) = O($n \log n$).

Proof of Correctness: Courses are assigned by increasing start time which allows a classroom's use to be maximized since courses are only added to the end of a classroom's current schedule and never the beginning. Due to this, the only time courses overlap is when the end time of an already added course is after the start time of the next course to be added. Therefore, the algorithm checks if a course (i) is compatible with the classroom that has the earliest end time (j). Only a single classroom needs to be checked because all the other classrooms have end times later than classroom j, so course i also starts before those classrooms end and is incompatible and therefore a new classroom must be used.

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: Sort the guests in decreasing order of the ratio T_i/t_i. Take guests home in order of decreasing ratio until no more guests are left.

Proof of Correctness:
Claim: The optimal solution takes the guest home in decreasing order of T_i/t_i.

1. Let there be an optimal solution OPT
2. Case 1. Our solution matches OPT, our claim holds
3. Case 2. Our solution does not match OPT. Since both lists contain all guests, they only differ in the ordering of guests. If there exists more than 1 optimal solution, this means that multiple guests have the same ratio, which allows the ordering to be different. Guests with the same ratio are able to be swapped without affecting the overall cost. These guests must be adjacent in the ordering. Therefore, we can swap the sets of equal-ratio guests from our solution with the ones in OPT. Therefore, our solution is optimal.

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.