CIS 675 Homework 5

Design and Analysis of Algorithms

Saket Kiran Thombre

SU ID: 899913802

NetID: sthombre

Email: sthombre@syr.edu

Collaborators for Q1, Q2:

1. Saurav Shashank Narvekar

   NetID: sanarvek
2. Chandan Pothumarthi

   NetID: cpothuma

Problem 1 Kubal (Point 5)

Every morning, customers 1, . . . , n show up to get their espresso drink at Kubal. Suppose the barista at Kubal knows all of the customers, and knows their orders, o1, . . . , on. Some customers are nicer people and give more tip; let Ti be the barista's expected tip from customer i. Some drinks like a simple double-shot, are easy and fast, while some other orders, like a soy-milk latte take longer. Let ti be the time it takes to make drink oi for customer i. Assume the barista can make the drinks in any order that he/she wishes, and that he/she makes drinks back-to-back (sequentially without any breaks) until all orders are done. Based on the order that he/she chooses to complete all drinks, let Di be the time that the barista finishes order i. Devise an algorithm that helps the barista pick a schedule that minimizes the quantity.

$$\sum_{t}^{n} TiDi$$

In other words, he/she wants to serve the high tippers the fastest, but he/she also wants to take into consideration the time it takes to make each drink. (Hint: think about a property that is true about an optimal solution.) Now, the solution of the problem is a greedy algorithm. Hence, you would need to proof that you approach/algorithm will give the optimal solution. You will get 1 point for the algorithm, and 4 points for the proof.

Solution:

In this problem, we can see that the barista will want to keep the high tippers happy as well as manage the time required for to make each drink. So, we try to solve this using greedy approach. In this approach we will choose the drink with maximum tip first and will sort all the orders in descending order of their tip amount.

This can help us in maximizing the amount of tip we receive and eventually maximize the profit.

To Prove:

Here, we have two criteria that must be considered while determining whose espresso should be prepared by the barista.

We sort all the job in descending order of their tips.

Iterate the jobs in descending order for tips

- Find a slot where the slot is empty, and the deadline is the highest time.
- Select tis job and put this into the slot and mark it.
- If nothing similar exists, ignore the job.

We will consider a ratio as of all the incoming orders as orderRatio = $\frac{Ti}{ti}$ where,

Ti is the tip received from the customer,

And ti is the time required for order to be prepared by the barista.

This ratio will give us a ratio of a time required to complete orders Oi.

According to greedy algorithm, there must exist a optimal solution where the barista will find a way to make drinks in descending order according to the largest order first.

Now let's consider that we have two consecutive expresso orders, with the later one having a higher ratio than the earlier one (as technically we are supposed to do the orders in the sequence when there are no constraints, but if we can show a particular order is better for our case, we have proved our theorem). This means that if we can demonstrate that performing the later order first has a lower total cost than performing the prior order first, the proof is complete. By using the same condition repeatedly, what is done for two orders can be extended to all orders.

Therefore, when we select two orders p and q, where q is finished first, and when Rp > Rq, then we should swap the orders schedule and lower its cost. This could be represented as:

$\frac{Tq}{tq} > \frac{Tp}{tp}$

This also means that Tqtq > Tptp. Hence when we add both sides we can get the following factoring:

Tq(tq + tp) + Tptp > Tqtq + TP(Tq + tp)

This demonstrates that if order p is made first on a schedule, the overall cost will be higher than if order q were done first.

If we iterate this, we can conclude that the order with the minimal cost begins the order with the largest ratio. Subsequently the second drink made by barista should also be highest Rp. □

This could be computed in the following way.

All the algorithm lines are numbered.

```
1. def baristaJob(arr, t)

We define an array of length arr
2.     drinks = len(arr)

Sort all drink orders in descending order of their tip values.
So that we can maximize our profit.
3.     for int1 in range(drinks)
4.         for int2 in range(drinks - 1 - int1)
5.             if arr[int2][2] < arr[int2 + 1][2]
6.                 arr[int2], arr[int2 + 1] = arr[int2 + 1], arr[int2]

To find free slots between customers
7.     freeTime = [False] * t

To save the sequence of jobs we are completing
```

```
8.      jobSequence = ['-1'] * t

Now interating through all the given job sequence stored in "jobSequence"
9.      for int1 in range(len(arr)):

We always start from the last possible drink possible to make.
We find a free slot for making this drink.
10.         for int2 in range(min(t - 1, arr[int1][1] - 1), -1, -1):

If we find a free slot
11.             if result[int2] is False:
12.                 result[int2] = True
13.                 jobSequence[int2] = arr[int1][0]
14.                 break
Now we print the sequence to make the drinks.
15.     print(jobSequence)
```

Problem 2 Late Assignments (Point 5)

College life is hard. You are given n assignments a1, . . . , an in your courses. Each assignment ai = (di , ti) has a deadline di when it is due and an estimated amount of time it will take to complete, ti . You would like to get the most out of your college education, and so you plan to finish all of your assignments. Let us assume that when you work on one assignment, you give it your full attention and do not work on any other assignment. In some cases, your outrageous professors demand too much of you. It may not be possible to finish all of your assignments on time given the deadlines d1, . . . , dn; indeed, some assignments may have to be turned in late. Your goal as a sincere college student is to minimize the lateness of any assignment. If you start assignment ai at time si , you will finish at time fi = si + ti . The lateness value—denoted `i—for ai is the value

$$\text{Li} = \begin{cases} fi - di \; if \; fi > di \\ 0 \; Otherwise \end{cases}$$

Devise a polynomial-time algorithm that computes a schedule O that specifies the order in which you complete assignments which minimizes the maximum `i for all assignments, i.e.

$$\min_{o} \max_{i} li$$

In other words, you do not want to turn in any assignment too late, so you minimize the lateness of the latest assignment you turn in. Prove that your algorithm produces an optimal schedule. Analyze the running time of your algorithm. Hint: The solution is very similar to the scheduling problem discussed in class. 1 point for the algorithm 4 point for the proof

Solution:

Claim

We will consider x and y as two consecutive jobs such that y = x+1.

For $d_{O(y)}$ <= $d_{O(x)}$ where O is an idle optimal schedule.

As a result, job(y) has an expiration date before job(x). Therefore, if (y) and (x) were switched, we could also have another ideal idle schedule.

Proof:

Hence,

Let's think about the moment job I starts in O.

Since the jobs O(x) and O(y) will be swapped, O' should be identical to O.

So, we infer that,

$$l_{O(y)} = s + t_{O(x)} + t_{O(y)} - d_{O(x)}$$

>= s + $t_{O(x)} - d_{O(x)}$

= $l_{O(x)}$

This will follow a similar trend since the task $t_{O(y)} > 0$ and the deadline $d_{O(x)}$ is earlier than expected than $d_{O(x)}$. This will similarly hold true for

$$l_{O(y)} \geq s + t_{O(x)} - d_{O(y)} = l'_{O(x)}$$

Subsequently,

$$l_{O(y)} \geq S + t_{O(y)} + t_{O(x)} - d_{O(y)} = l'_{O(x)}$$

As we see, this will follow the same $d_{O(y)}$ <= $d_{O(x)}$.

O' must also be ideal as O's lateness in the job wouldn't be more than O's.

As a result, O and O' are equal to one another.

We can then conclude that an optimal solution that is the same as the earliest deadline first is also feasible by applying the claim to obtain the optimal solution schedule O. Thus, using the earliest deadline first may also be the best course of action. □

All the algorithm lines are numbered.

```
1. Sort all the assignments using their deadlines such that as1 <= as2 <= as3
...<= asn
2. Initially, t <= 0
3. Initially, A <= (Null)
4. For j = 1 to n
5.      s_j <= t
6.      f_j <= t + t+j
7.      Add [s_j, f_j] to A
8.      t <= f_j
9. Retun A
```