H6.PATEL.NISARGS

PROBLEM 1 *Ski racing*

You run the Giant slalom course at Loon on weekends. There must always be a ski patrol on duty in case there are injuries during training. There are n patrol certified instructors who can work, but they are busy college students with complex social obligations; instructor i can work starting at time a_i and ending at time b_i on Saturdays. Your goal is to cover the entire day from 8a–8p *using the fewest instructors as possible*. You can assume that there are enough instructors to cover the entire day; the only problem is to find the smallest number of them to do it.

1. The first idea that comes to mind is to start with an empty schedule, and then add the instructor who covers *the most amount of time that is not already covered*. Show that this algorithm fails by providing a counter-example.
2. State and prove an exchange style lemma that you can use to construct a greedy algorithm.
3. Present pseudo-code and running time analysis for an algorithm that uses your lemma from above to output the smallest number of instructors needed.

Ans: Using 24-hour format, we have to cover the day from 8 to 20.

1. For the instructors having start and end time: [(8,10), (10, 18), (18, 20), (8, 14), (14, 20)]. The mentioned algorithm selects i_2 first and then i_1 and i_3 thus selecting 3 instructors. Whereas a better solution exists by selecting 2 instructors, i_4 and i_5.
2. **Exchange Lemma:** For time between t_i and t_j, instructor available at time t_i which has the greatest end time is always part of some $SOL_{i,j}$.

   Proof: Consider an optimal solution $SOL_{i,j}$ and let I^* be the instructor that can work at time t_i and has greatest end time.
   Case 1: If $I^* \in SOL_{i,j}$, then the claim follows.
   Case 2: If $I^* \notin SOL_{i,j}$, let I be the instructor in $SOL_{i,j}$, selected for time t_i.
   Consider a new solution $SOL^*_{i,j}$ which replaces I with I^*.
   $SOL^*_{i,j} = SOL_{i,j} - \{I\} \cup \{I^*\}$
   This new set is valid because time of instructors I and I^* intersect from t_i till b_I and hence I^* can cover all the time covered by I. Also, $SOL_{i,j}$ and $SOL^*_{i,j}$ have the same size. Thus $SOL^*_{i,j}$ is also optimal.
3. To find $SOL_{8,20}$, we need to sort according to start-time(a_i) and iteratively apply the above lemma to select the instructor and reduce the problem to $SOL_{b_I,20}$.

Pseudocode for SkiRacing:

```
SkiRacing($a(1..n), b(1..n)$)
Sort($a, b$) based on increasing value of $a$
$currentTime \leftarrow 8$
$endTime \leftarrow -\infty$
$count \leftarrow 0$
for $i \leftarrow \{1, ..., n\}$
  if $a_i > currentTime$
    $count \leftarrow count + 1$
    $currentTime \leftarrow endTime$
  if $b_i > endTime$
    $endTime \leftarrow b_i$
if $currentTime < 20$
  $count \leftarrow count + 1$
return count
```

Time Complexity Analysis:
Line 1 sorts a, b on increasing order of a. If used, MergeSort, Sort takes $\Theta(n \log(n))$ time. Line $5 - 10$ is called n times. Thus the overall time complexity of the algorithm is $\Theta(n \log(n))$.

PROBLEM 2 *Latte love*

Every morning, customers $1, \ldots, n$ show up to get their drink. Suppose the omniscient barista knows all of the customers, and knows their orders, $o_1, \ldots, o_n$. Some customers are nicer people; let T_i 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 t_i be the time it takes to make drink o_i. Assume the barista can make the drinks in any order that she wishes, and that she makes drinks back-to-back (without any breaks) until all orders are done. Based on the order that she chooses to complete all drinks, let D_i be the time that the barista finishes order i. Devise an algorithm that helps the barista pick a schedule that minimizes the quantity

$$\sum_i^n T_i D_i$$

In other words, she wants to serve the high-tippers the fastest but 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.)

1. State and prove an exchange lemma that is true about this problem and can be used for a greedy algorithm.

2. Provide pseudo-code and run time analysis for your algorithm.

Ans:

1. Let S_R be the schedule to make the drinks with decreasing order of $\frac{T_i}{t_i}$.
   Let $Val(S) = \sum_i^n T_i D_i$ for any schedule S.
   Exchange Lemma: Let S be a schedule agreeing with S_R for first j orders. Then there exists a schedule S'' agreeing with S_R on first $j+1$ orders and has $Val(S'') \leq Val(S)$.

   Proof: Let o^* be the order with maximum value of $\frac{T_{o^*}}{t_{o^*}}$ in S between indices $j+1$ and n.
   Case 1: If o^* is at index $j+1$, then S follows S_R for $j+1$ orders and the lemma holds.
   Case 2: o^* is between indices $j+2$ and n. Let o^* be at index l. Let o be the order in S at index $l-1$.
   Consider another schedule $S'(T', t', D')$ that follows S except at indices $l-1$ and l. Instead S' contains o^* at index $l-1$ and o at index l, thus swapping these two order's sequence from S.
   Since both schedules differ only on indices $l-1$ and l and time taken to complete o^* and o in any order is same, we have $T_i D_i = T'_i D'_i$ for all $i \neq l-1, l$. Let k be the time taken to complete first $l-2$ orders in S and S'.

So we have,

$$\begin{aligned} Val(S) &= \sum_{i \neq l-1,l}^{n} T_i D_i + T_o D_o + T_{o^*} D_{o^*} \\ Val(S') &= \sum_{i \neq l-1,l}^{n} T_i D_i + T'_{o^*} D'_{o^*} + T'_o D'_o \end{aligned} \tag{1}$$

Also, we have:

$$\begin{gathered} T_o = T'_o, T_{o^*} = T'_{o^*} \\ D_o = k + t_o, D'_{o^*} = k + t_{o^*} \\ D_{o^*} = k + t_o + t_{o^*}, D'_o = k + t_{o^*} + t_o \end{gathered} \tag{2}$$

Using equations in (1) and (2), we have

$$\begin{aligned} Val(S) - Val(S') &= T_o(k + t_o) + T_{o^*}(k + t_o + t_{o^*}) - T_{o^*}(k + t_{o^*}) - T_o(k + t_{o^*} + t_o) \\ &= T_{o^*}(t_o) - T_o(t_{o^*}) \\ \frac{Val(S) - Val(S')}{t_o t_{o^*}} &= \frac{T_{o^*}}{t_{o^*}} - \frac{T_o}{t_o} \\ \frac{Val(S) - Val(S')}{t_o t_{o^*}} &\geq 0 \\ Val(S') &\leq Val(S) \end{aligned} \tag{3}$$

Similarly we can continue swapping o^* with its previous order in S', and get a schedule S'' which follows S_R on first $j+1$ orders, such that $Val(S'') \leq ... \leq Val(S') \leq Val(S)$. Hence the lemma holds.

Let S^* be the optimal schedule that follows S_R till 0 indexes. Let S_1 follow S_R on 1 order, S_2 follow S_R on 2 orders, ... S_n follow S_R on n orders which is the schedule S_R itself. Based on the above lemma we have:

$$Val(S^*) \leq Val(S_1) \leq Val(S_2) \leq ... \leq Val(S_n)(= Val(S_R))$$

Since S^* is optimal, we have all of the above values equal and hence S_R is optimal.

2. **Pseudocode** for LatteLove:

```
LatteLove($o_1(T_1, t_1), \ldots, o_n(T_n, t_n)$)
1  Sort($o_i, T_i, t_i$) based on decreasing value of $\frac{T_i}{t_i}$
2  $(val, D, schedule) \leftarrow (0, 0, \{\})$
3  for $i \leftarrow \{1, ..., n\}$
4      $D \leftarrow D + t_i$
5      $val \leftarrow val + (t_i * D)$
6      $schedule \leftarrow$ add $o_i$ in $schedule$
7  return $(schedule, val)$
```

Time Complexity Analysis:
Line 1 sorts o_i, T_i, t_i on decreasing value of $\frac{T_i}{t_i}$. If used, MergeSort, Sort takes $\Theta(n \log(n))$ time. Line $3 - 6$ loop takes $\Theta(n)$ time. Thus the overall time complexity of the algorithm is $\Theta(n \log(n))$.

PROBLEM 3 *5800 is tough*

This class is hard and so are all of your others. You are given n assignments $a_1, \ldots, a_n$ in your courses. Each assignment $a_i = (d_i, t_i)$ has a deadline $d_i > 0$ minutes from now when it is due and an estimated amount of time it will take to complete, $t_i > 0$ in minutes. You would like to get the most out of your 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 $d_1, \ldots, d_n$; indeed, some assignments may have to be turned in late. Your goal as a sincere student is to minimize the lateness of any assignment. If you start assignment a_i at time s_i, you will finish at time $f_i = s_i + t_i$. The lateness value—denoted ℓ_i—for a_i is the value

$$\ell_i = \begin{cases} f_i - d_i & \text{if } f_i > d_i \\ 0 & \text{otherwise} \end{cases}$$

Your goal is to compute a schedule O, starting now at time 0, that specifies the order in which you complete assignments which minimizes the maximum ℓ_i for all assignments, i.e.

$$\min_O \max_i \ell_i$$

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.

1. State an exchange lemma that captures the right choice to make in this problem for a greedy algorithm.
2. Describe with pseudo-code and analyze and algorithm that uses your lemma to solve the problem.

Ans:

1. Let S_{EDF} be the schedule to complete the assignment with increasing value of deadlines. Thus doing earlier deadline assignment first.
   Let $Val(S) = \max_i \ell_i$ for any schedule S.
   An optimal solution will not contain any breaks in between 2 assignments as that break can be utilized to do an assignment and potentially reduce the maximum lateness.
   Exchange Lemma: Let S be a schedule agreeing with S_{EDF} for first j assignments. Then there exists a schedule S'' agreeing with S_{EDF} on first $j+1$ orders and has $Val(S'') \leq Val(S)$.

   Proof: Let a^* be the assignment with earliest deadline in S between indices $j+1$ and n.
   Case 1: If a^* is at index $j+1$, then S follows S_{EDF} for $j+1$ assignments and the lemma holds.

Case 2: a^* is between indices $j+2$ and n. Let a^* be at index l. Let a be the assignment in S at index $l-1$.
Consider another schedule $S'(a'(d', t', \ell'))$ that follows S except at indices $l-1$ and l. Instead S' contains a^* at index $l-1$ and a at index l, thus swapping these two assignment's sequence from S.
Since both schedules differ only on indices $l-1$ and l and time taken to complete a^* and a in any order is same, we have,

$$\ell_A = \ell'_A \text{ for all assignments } A \text{ except } a \text{ and } a^* \tag{4}$$

Let k be the time taken to complete first $l-2$ assignments in S and S'.
So we have for assignement a and a^* we have,

$$\begin{aligned} \ell_a = \max(k + t_a - d_a, 0), \ell_{a^*} = \max(k + t_a + t_{a^*} - d_{a^*}, 0) \\ \ell'_{a^*} = \max(k + t_{a^*} - d_{a^*}, 0), \ell'_a = \max(k + t_{a^*} + t_a - d_a, 0) \end{aligned} \tag{5}$$

Since, we have $d_{a^*} \leq d_a$,

$$\begin{aligned} \ell_{a^*} &= \max(k + t_a + t_{a^*} - d_{a^*}, 0) \\ &\geq \max(k + t_{a^*} - d_{a^*}, 0) \\ \ell_{a^*} &\geq \ell'_{a^*} \\ \ell'_{a^*} &\leq \ell_{a^*} \end{aligned} \tag{6}$$

Now for assignment a,

$$\begin{aligned} \ell'_a &= \max(k + t_{a^*} + t_a - d_a, 0) \\ &\leq \max(k + t_{a^*} + t_a - d_{a^*}, 0) \\ \ell'_a &\leq \ell_{a^*} \end{aligned} \tag{7}$$

As $d_{a^*} \leq d_a$, we have, $\ell_a \leq \ell_{a^*}$ and thus from equations (4), (6), and (7), we have $Val(S') \leq Val(S)$.
Similarly we can continue swapping a^* with its previous assignment in S', and get a schedule S'' which follows S_{EDF} on first $j+1$ orders, such that $Val(S'') \leq ... \leq Val(S') \leq Val(S)$. Hence the lemma holds.

Let S^* be the optimal schedule that follows S_{EDF} till 0 indexes. Let S_1 follow S_R on first assignment, S_2 follow S_R on 2 assignments, ... S_n follow S_{EDF} on n orders which is the schedule S_{EDF} itself. Based on the above lemma we have:

$$Val(S^*) \leq Val(S_1) \leq Val(S_2) \leq ... \leq Val(S_n)(= Val(S_{EDF}))$$

Since S^* is optimal, we have all of the above values equal and hence S_{EDF} is optimal.

2. **Pseudocode** for TOUGH5800:

```
TOUGH5800($(a_1, d_1, t_1), ..., (a_n, d_n, t_n)$)
1  SORT($a_i, d_i, t_i$) based on increasing value of $d_i$
2  $(val, f, schedule) \leftarrow (0, 0, \{\})$
3  for $i \leftarrow \{1, ..., n\}$
4      $f \leftarrow f + t_i$
5      $\ell \leftarrow \max\{f - d_n, 0\}$
6      $val \leftarrow \max\{val, \ell\}$
7      $schedule \leftarrow$ add $a_i$ in $schedule$
8  return $(schedule, val)$
```

Time Complexity Analysis:
Line 1 sorts a_i, d_i, t_i on increasing value of d_i. If used, MERGESORT, SORT takes $\Theta(n \log(n))$ time. Line $3 - 7$ loop takes $\Theta(n)$ time. Thus the overall time complexity of the algorithm is $\Theta(n \log(n))$.