HW 6

Yunhao Lin

October 23, 2018

1. In order to maximize our profit from delivering packages, first we still suppose i is the delivery we are going to make, and Y is the total time used for all the previous deliveries. Sort the delivery by due date. We call our method $maxpro(i, Y)$. Every time we decide if the i_{th} delivery should be included in the delivery schedule or should we exclude it. By including the i_{th} delivery, we pass the next delivery and plus the total time used Y by t_i. And here we add the p_i so thorough all the recursive calls, we can get the total money earned, since this is the case where we need to delivery the i_{th} item. Also if we exclude the i_{th} item, then we continue to the next delivery and pass Y as it is. From these two, we chose the maximum profit. The expression is listed below:

$$maxpro(i, Y) = max \begin{cases} maxpro(i+1, Y+t_i) + p_i \\ maxpro(i+1, Y) \end{cases}$$

And here is the pseudo-code for the algorithm:

Algorithm 1 Maximum profit

```
Initialize Arraylist $q$
Sort the due date by ascending order
procedure MAXPRO($i, Y$)
    if $i$ = $n$+$1$ then
        return 0;                                            ▷ the bundary situation
    else
        if $Y + t_i - 1 > d_i$ then                           ▷ the item cannot be delivered
            return $maxpro(i+1, Y)$
        else if $maxpro(i+1, Y+t_i) + p_i > maxpro(i+1, Y)$ then
            Record index $q.add(0, i+1)$                      ▷ Add to the first position in $q$
            return $maxpro(i+1, Y+t_i) + p_i$
        else
            return $maxpro(i+1, Y)$
```

Proof of Correctness:
Claim: The algorithm would correctly return an list of deliveries to make.
Base case: Base case is when $i = n$. When we maxpro with $(i = n)$, there is three different situations. If Y, the time used before the delivery of the i_{th} item plus the time to delive the i_{th} item is longer than the deadline of this item, then we can only choose to not include this item and return maxpro(i+1, Y) which would be 0. And if we can include this last item grants more profit as in line, here we record the index because that's when we need to deliver the package. At the last case when $i = n$, both function call on line eight would return 0 but HS has the p_i plused. So it guarentee maxprofit if the item is eligible to be delivered.
Inductive Hypothesis: When $i = k$, when $k < n$, the function call would return the maximum profit from $k + 1$ to n packages.
Inductive Steps: For this we need to prove that when $i = k - 1$, the algorithm would still return the maximum profit from k to n. When the algorithm calls on $maxpro(k - 1, Y)$, there is three possible situations.

- If $(k-1)_{th}$ package cannot be delivered, then we can only choose not to deliver this $(k-1)_{th}$ package. And we call on maxpro(k, Y) and by the inductive hypothesis, it is going to return the maximum profit from $k+1$ to n package. Since we are not delivering this $(k-1)_{th}$ item, the maximum profit is the max porfit from k_{th} package to the last pacakage.
- If the item can be delivered, then here we compare the maxprofit of including the package for delivery and not include it for delivery. As in line 8, we already know from the inductive hypothesis that $maxpro(k, Y+ti)$ and $maxpro(k, Y)$ is going to return the maximum profit from k to n packages. So comparing these would help us decide if we need to include the package, which is the p_i term on the Left Hand Side. If yes, then record the package and return the maxprofit now including the profit of this k_{th} package. Else return the max profit not excluding the k_{th} package. In any case, the profit is maximized from k to n packages after $maxpro(k-1, Y)$.

So $P(k)$ holds indicates $P(k-1)$, where $k-1>0$. By induction, the algorithm would correctly return the max profit. And since we record the choice that would result in this max profit at each point, we can get a sorted list of all the packages to deliver.
Time complexity: Since there is n number of packages and the worst case is to check $Y>T$ at the last package, the size of the subproblems would be $O(nT)$ and each subproblem takes $O(1)$ time to compute since the value from next level is already calculated and all we do is compare and addition. So in total the time complexity of this algorithm would be $O(nT)$.

2. First sort the d_i in descending order to an array called α and we declare k as the indexs of α, k is from 1 to need. So the first index of k is the latest deadline which is T as defined in 1. We call our recursive function maxdel(k, j). Here, k is index of the sorted array of deadline and j is the number of deliveries in the previous k_th packages.

$$maxdel(k,j) = min\begin{cases} maxdel(k+1, j+1) - t_i \\ maxdel(k+1, j) - (d_{k+1} - d_k) \end{cases}$$

Here t_i is the time need to finish the delivery at the due date $\alpha[k+1]$. We are going to record all the value in an 2-D array.

And here is the pseudo-code for the algorithm:

Algorithm 2 Maximum delivery

```
1: Sort the pacakge as their deadline in descending order
2: Name this sorted array as$\alpha$
3: Initialize a 2-D array maxdel to all -1 maxdel[n, 0] = T;
4: for k = n+1 to 1 do
5:     for j = 1 to k do
6:         if k = n+1 then
7:             maxdel[k, j] = 0;                                    ▷ Base case
8:         if $min(maxdel[k+1, j+1] - t_i, maxdel[k+1, j] - (d_{k+1} - d_k)) - t_i < 0$ then
9:             maxdel[k,j] = maxdel[k+1, j];
10:        else if $maxdel[k+1, j+1] - t_i \leq maxdel[k+1, j] - (d_{k+1} - d_k)$ then
11:                                                    ▷ $t_i$ is corresponding to $k$
12:            maxdel[k, j] = $\alpha[k]$ - $t_i$ + 1
13:        else
14:            maxdel[k, j] = maxdel[k+1,j]
15:
16: for k = n to 1 do
17:     for j = k to 1 do
18:         find max j of maxdel[k, j] that is not -1          ▷ Find the maximum j value
19: Find the traverse that lead to the maximum j is the delivery order we want.
```

Proof of Correctness:
The algorithm above is going to find the maximum delivery number and the order to deliver.
Base case: When we call $MinDdl(n, 0)$, we reached our base case where no deliveries is made. So in this case, the total time remaining for us to deliver is T. So, we set $MinDdl[n, 0]$ equal to T. Since we would call $MinDdl(n + 1, j)$ where we actually don't have $(n + 1)^{th}$ item, we set $MinDdl[n + 1, j]$.
Inductive Hypothesis: For k = s, s is between 1 and n, maxdel(s, j) is going to return the minimum deadline available, and j holds the value of the total deliverys in the previous s packages.
Inductive Steps: In order to prove we need to verify that when k = s-1, maxdel(s,j) would still get the correct minimum deadline. In the case there is coulple situations that might happen.

- First if the current package cannot be delivered in the current minimum deadline, then we choose to exluce it and keep the deadline as it is.
- If the package can be fit in the schedule, and $maxdel[k+1, j+1]-t_i \leq maxdel[k+1, j]-(d_{k+1}-d_k)$ Therefore we decide to include the package in the delivery then update the deadline to be the start state of the current package. Since in order to deliver the package, we need to start at least at $\alpha[s] - t_i + 1$. Here, t_i is the corresponding time need to deliver the package.
- If we decide not to include the package in the delivery, then just change the minimum due date to the due date of the next item to deliver. Therefore, this is going to guarentee us the minimum deadline is met at the end.

So, by induction, the algorithm is going to return the minimum deadline. So at the end our deadline should be very close to 0. We save all the possiblities during the recursion and after the recursion finished we know all the j values thourgh out the possiblities and stored them in the 2d array. The algorithm traverse the array and find the maximum j possible and choose one path that led to j. So the algorithm would determine which deliveries to make and in what order to maximize the number of deliveries we make.
Time complexity: Since the two for loop both has O(n), the time complexity here is $O(n^2)$ and we need to find the maximum j and the route to j is also $O(n^2)$. So the total time complexity is $O(n^2)$