Aditya Shirode, Rhythm Shah, Shrenuj Gandhi

5. [7 points] *Dynamic programming.*

   Suppose n jobs, numbered $1, \ldots, n$ are to be scheduled on a single machine. Job j takes t_j time units, has an integer deadline d_j, and a profit p_j. The machine can process only one job at a time and each job must be processed without interruption (for its t_j time units). A job j that finishes before its deadline d_j receives profit p_j, while a tardy job receives no profit (and might as well not have been scheduled). Give an efficient algorithm to find a schedule that maximizes total profit. You may assume that the t_j are integers in the range $[1, n]$. You should be able to come up with an $\mathrm{O}(n^3)$ algorithm if you make the right observation about the maximum relevant deadline.

Answer:

Given n jobs such that -
t_j is time required to complete j^{th} job, $t_j \in [1 \ldots n]$,
d_j is integer deadline for j^{th} job,
p_j is profit earned after j^{th} job is completed.

Let D be the deadline time for which we have to schedule jobs and earn maximum profit.
It is safe to say that $D \geq$ Maximum value of the deadlines.

For each job, we have to decide whether it is part of the final solution or not. As we include this job, we add its profit p_j to the overall profit earned, p, and decrease the deadline D by t_j.
Thus for each j^{th} job, we chose -
max(profit if job is not included, profit if job is included) ...which takes $\mathrm{O}(2^n)$

To solve this dynamically, we break this recurrence and we create a matrix $profit[0 \ldots n][1 \ldots D]$, where $profit[j][k]$ is given by -

$$profit[j][k] = \begin{cases} 0, & \text{if } \; j = 0, k \leq t_j, k \geq d_j \\ max(profit[j-1][k-1], profit[j-1][k-t_j] + p_j), & \text{otherwise} \end{cases}$$

$0 \leq j \leq n, \;\; 1 \leq k \leq D$
and the jobs are sorted in non-decreasing order of their deadlines.

Since our task is to find the schedule that maximizes total profit, we create another matrix $job[1 \ldots n][1 \ldots D]$, where $job[j][k]$ is given by -

$$job[j][k] = \begin{cases} 1, & \text{if } \; profit[j-1][k-1] \leq profit[j-1][k-t_j] + p_j \\ 0, & \text{otherwise} \end{cases}$$

Hence, the algorithm looks like -

```
Algorithm(n, t[], p[], d[], D)

        sortDeadlines(t[], p[], d[]);                         ...takes O(nlgn)

        for j ← 0 to n, do
                for k ← 1 to D, do
                        if j = 0 OR k ≤ t[j] OR k ≥ d[j]
                                profit[j][k] ← 0

                        else if (profit[j-1][k-1] ≤
                                profit[j-1][k-t[j]] + p[j]) then
                                                              ...takes O(nD)
                                profit[j][k] ← profit[j-][k-t[j]] + p[j]
                                job[j][k] ← 1
                        else
                                profit[j][k] ← profit[j-1][k-1]
                                job[j][k] ← 0

        for j ← n down to 1, do                               ...takes O(n)
                if job[j][D] = 1
                        schedule job j
                        D ← D - t[j]

end
```

Among the 3 stages, the second stage grows the fastest, and hence the algorithm is O(nD), where n is # of Jobs, and D is overall deadline.

In the worst case, the first job can take n units, similarly the second job can take n units, and so on. Thus the deadline D has to be atleast $n * n$, so that all n jobs (with n runtime) have equal chance to be scheduled.

Hence, in this case, the algorithm behaves like O(n^3).