CS 577: Introduction to Algorithms — Homework 6 Solutions

Problem 1

We will give a dynamic program which constructs the optimal schedule for deliveries in a manner similar to the knapsack algorithm. That is, we build up solutions to sub problems restricted to the first i items, then increment i. For the knapsack problem, the order in which we considered items didn't matter. Here, however, the order in which the deliveries are made matters, so we must carefully consider the order in which we choose to do these deliveries.

Consider any schedule of k deliveries σ = (a_1,a_2,...,a_k), where a_1 is the the first delivery being made and a_k is the last. For some delivery i, let d_i be the deadline and t_i be the amount of days required to deliver it. We use an exchange argument to show that the deliveries in σ can be made in increasing order of deadlines: suppose, for some index i, that $d_i > d_{i+1}$. Then we can swap the order of delivery i and delivery $i+1$. This is because delivering package $i+1$ before package i has no effect on $i+1$, as $(i+1)'s$ delivery just starts earlier. Also, if the earlier schedule was feasible then $t_i + t_{i+1} \leq d_{i+1}$ but since $d_i > d_{i+1}$ even for i the new delivery schedule is still feasible. In particular, an optimal schedule can be so arranged that the packages with earlier deadline are delivered before packages with later deadlines. In the rest of the solution, we assume the packages are numbered in order of increasing deadlines. That is, $d_1 \leq d_2 \leq ... \leq d_n$.

We now try to come up with a recurrence for the problem. Lets consider some package i and some start day s. Suppose we want to find a delivery schedule from package i onwards, and with a starting date of s, such that we maximize our profit. We have two choices for package i, we can either deliver it or leave it. Suppose we decide to deliver this package, we then gain a profit of p_i but we have spent t_i days to deliver it. In this case, when we are considering package $i+1$, we can only start deliveries at day $s + t_i$. If we decide to not deliver package i, then we have not gained any profit, but we can start delivering package $i+1$ and onwards from day s. The above can be used to come up with a recurrence equation containing i and s, where i represents the package under consideration and s represents the day at which the next delivery can start. The value of i goes from 1 to n and the value of s goes from 0 to T (where T is the max deadline of any package). We address one final issue in our recurrence relation: Depending on the deliveries made before i we might be at a start date s, from which there is no way to deliver package i before its deadline. In such a case, there is no profit in delivering this package and we will simply choose not to deliver it. This leads to the following recurrence below.

$$\mathrm{OPT}[i,s] = \max \begin{cases} \mathrm{OPT}[i+1,s] \\ \mathrm{OPT}[i+1,s+t_i] + p_i, \text{only if } s + t_i \leq d_i \end{cases} \quad (1)$$

We now give an iterative solution using the recurrence relation above.

Algorithm:

1. Sort packages by increasing order of deadline, break ties by sorting on increasing order of time taken to deliver.
2. Base Case : For s in 0 to T, set $\mathrm{OPT}[n+1,s] \leftarrow 0$
3. For i in n to 0, For s in T to 1
   (a) If ($s + t_i > d_i$)
   (b) $\mathrm{OPT}[i,s] = \mathrm{OPT}[i+1,s]$
   (c) Else
   (d) $IncludingDelivery = \mathrm{OPT}[i+1,s+t_i] + p_i$

(e) $ExcludingDelivery = \text{OPT}[i+1, s]$

(f) Set $\text{OPT}[i, s] \leftarrow \max(IncludingDelivery, ExcludingDelivery)$

4. Return $\text{OPT}[1, 0]$

Run time of our algorithm is equal to the time taken to fill in the array. Since the size of the array is $(n+1) \times T$ and we spend constant time filling each position, the runtime is $O(nT)$. Here n is the number of packages and T is the latest deadline among all packages.

Problem 2

The dynamic programming solution to this problem is based on the following subproblem specification:

$\text{WIN}[h_1, a, h_2, \ell] = true$ iff player 1 is guaranteed to win when player 1's character starts combat with h_1 hitpoints and a attack power, and player 2 starts combat with h_2 hitpoints and ℓ uses of heal left.

These values are defined for $0 \leq h_1 \leq H_1$, $1 \leq a \leq 5$, $0 \leq h_2 \leq H_2$ and $0 \leq \ell \leq 2$. Given this subproblem specification, the answer we wish to compute is given by $\text{WIN}[H_1, 1, H_2, 2]$.

Say we want to compute the value of $\text{WIN}[h_1, a, h_2, \ell]$. Note that player 1 has two choices: either to attack or to flex (though flexing is only an option when their character's attack power is at most 4). In case they choose to attack, then player 2 may choose to either attack back or heal (if they have enough uses of heal left). In case player 2 chooses to attack, we are left with the following subproblem:

$$\text{WIN}[h_1 - A_2, a, h_2 - a \cdot A_1, \ell].$$

But if player 2 chooses to heal, then we are left with the following subproblem:

$$\text{WIN}[h_1, a, \min\{h_2 - a \cdot A_1 + 20, H_2\}, \ell - 1].$$

Alternatively, if player 1 chooses to flex, then again player 2 has the same two options. If player 2 chooses to attack, we are left with the subproblem:

$$\text{WIN}[h_1 - A_2, a + 1, h_2, \ell].$$

But if player 2 chooses to heal, then we have the subproblem:

$$\text{WIN}[h_1, a + 1, \min\{h_2 + 20, H_2\}, \ell - 1].$$

Now, remember we are under the assumption that both players are playing optimally. This means that if player 1 has a choice that leads to a win (returns $true$), then they are going to pick that option and win, and the same holds for player 2, though in that case they are looking for an option that returns false. In this case, it makes sense to take the Boolean OR of the two options player 1 has and then the Boolean AND of the two options player 2 has once player 1 has made a decision. Putting all of this together, the following recurrence correctly computes the value of $\text{WIN}[h_1, a, h_2, \ell]$ for all relevant values:

$$\begin{aligned}
\text{WIN}[h_1, a, h_2, \ell] = \big(&\text{WIN}[h_1 - A_2, a, h_2 - a \cdot A_1, \ell] \\
&\text{AND WIN}[h_1, a, \min\{h_2 - a \cdot A_1 + 20, H_2\}, \ell - 1]\big) && \text{only if } \ell \geq 1 \\
&\text{OR } \big(\text{WIN}[h_1 - A_2, a + 1, h_2, \ell] && \text{only if } a \leq 4 \\
&\text{AND WIN}[h_1, a + 1, \min\{h_2 + 20, H_2\}, \ell - 1]\big) && \text{only if } a \leq 4 \text{ and } \ell \geq 1
\end{aligned}$$

We are still missing the base cases, though these are simple. If player 1 is able to deal enough damage in an attack to defeat player 2, then player 1 will surely choose to attack and win. Alternatively, if player 1 is not able to deal

enough damage to defeat player 2, but player 2 is able to do so, then player 2 wins. This motivates the following base cases:

$$\begin{aligned} &\mathrm{WIN}[h_1, a, h_2, \ell] = true && \text{for } h_2 \leq a \cdot A_1 \\ &\mathrm{WIN}[h_1, a, h_2, \ell] = false && \text{for } h_2 > a \cdot A_1 \text{ and } h_1 \leq A_2 \end{aligned}$$

While the discussion above already covers most of what is necessary to establish correctness, a formal proof of correctness would have to use an inductive argument to guarantee that the recursive calls to WIN are correct. Note, however, that it is not immediately clear what we would need to induct on. To deal with this, we view each entry in the 4-dimensional table WIN as a vertex in a graph, and add an edge from one entry to the other when the value of the later depends on the value of the former. For example, $\mathrm{WIN}[h_1, a, h_2, \ell]$ for $h_2 \leq a \cdot A_2$ (one of the base cases) has no incoming edges, and there is an edge from it to $\mathrm{WIN}[h_1 + A_2, a, h_2 + a \cdot A_1, \ell]$ by the first line of the recurrence relation. Note that the graph we obtain by this procedure is a DAG (Directed Acyclic Graph) and because of this it admits a topological ordering. Finally, we can induct on this topological ordering, such that we can always assume that the values of WIN required to compute one entry of the table are correct.

Finally, we analyze the running time. We have a total of $5 \cdot 3 \cdot H_1 \cdot H_2 = O(H_1 \cdot H_2)$ subproblems, and computing each of those takes time $O(1)$. This means that a recursive implementation for the recurrence using memoization takes time $O(H_1 \cdot H_2)$.

Alternative solutions/variations

Similarly to the card game of Problem 1 from the discussion, we could define subproblems WIN_1 and WIN_2 such that:

$\mathrm{WIN}_i[h_1, a, h_2, \ell]_i = true$ iff player 1 is guaranteed to win when player 1's character starts with h_1 hitpoints and a attack power, player 2 starts with h_2 hitpoints and ℓ uses of heal left, and the first player to select an action is player i.

The recurrence relation is then very similar to the one we have seen before, though now we only need to look at two subproblems in order to make a decision.

$$\begin{aligned} \mathrm{WIN}_1[h_1, a, h_2, \ell] =& \mathrm{WIN}_2[h_1, a, h_2 - a \cdot A_1, \ell] \\ & \text{OR } \mathrm{WIN}_2[h_1, a+1, h_2, \ell] && \text{only if } a \leq 4 \end{aligned}$$

$$\begin{aligned} \mathrm{WIN}_2[h_1, a, h_2, \ell] =& \mathrm{WIN}_1[h_1 - A_2, a, h_2, \ell] \\ & \text{AND } \mathrm{WIN}_1[h_1, a, \min\{h_2 + 20, H_2\}, \ell - 1] && \text{only if } \ell \geq 1 \end{aligned}$$

The base cases are also similar:

$$\begin{aligned} &\mathrm{WIN}_1[h_1, a, h_2, \ell] = true && \text{for } h_2 \leq a \cdot A_1 \\ &\mathrm{WIN}_2[h_1, a, h_2, \ell] = false && \text{for } h_1 \leq A_2 \end{aligned}$$

This subproblem specification at most doubles the number of subproblems we need to solve, and the time per subproblem is still constant, so this solution also runs in time $O(H_1 \cdot H_2)$.

Finally, instead of using Boolean values to indicate whether player 1 wins or loses, we could, for example, have used numeric values 1 and 0. In this case, we would need to replace the OR's by $\max$'s and the AND's by $\min$'s.