ADA 2021 Tutorial 5

Jatin, Syamantak

1. Define the subproblems clearly
2. Write a recursion using the above definition
3. Argue properly about why the recursion is correct (this is the optimal substructure property)
4. Implement using tables and argue runtime.

1 Jatin turns DJ Again.

(From Midsem ADA 2021) Fed up with the accidental question paper leak, Jatin decides to turn DJ once again. Now, he has n club nights he can possibly perform at, for simplicity the ith club night is on night i. If he performs on night i, he earns money v_i and this costs him energy w_i. Now, every day he eats food (regardless of whether he has a performance that night) which gives him exactly 1 unit of energy. Jatin starts with an initial energy of 0, so just before the first night, his energy is exactly 1 (from the food that day).

Jatin needs to make sure that his energy never goes below zero else he won't be able to teach ADA ever again. In other words, if Jatin performs at a subset S of nights then S should satisfy

$$i - \sum_{j \in S : j \leq i} w_j \geq 0 \text{ for all } 1 \leq i \leq n$$

Give a polynomial-time (in n) algorithm to find the subset of club nights Jatin can perform at to make the most money, while ensuring that his energy always stays non-negative. Assume that w_i, v_i are integers.

Solution.

Subproblems. Define $\mathsf{saveJatin}(i, j)$ to be the optimal solution to the subproblem with the nights under consideration being $i, i+1, \cdots n$ and the energy before the performance on night i equal to j, $\forall 1 \leq i \leq n+1, j = 0, 1, 2, \cdots n$. (Note that since Jatin is getting 1 unit of energy every night, the total energy cannot exceed n). [+2 for subproblems]

Recurrence.

$$\mathsf{saveJatin}(i,j) = \max \left\{ \begin{array}{l} \mathsf{saveJatin}(i+1, j+1) \\ \mathsf{saveJatin}(i+1, j-w_i+1) + v_i \end{array} \right\}, \forall 1 \leq i \leq n, w_i \leq j \leq n$$

$$\mathsf{saveJatin}(i,j) = \mathsf{saveJatin}(i+1, j+1), \forall 1 \leq i \leq n, j < w_i$$

$$\mathsf{saveJatin}(n+1, j) = 0 \forall\, j$$

Final Solution. is given by $\mathsf{OPT}[1][1]$

Algorithm 1 DP for Save Jatin

```
procedure FindOptBalls(v[1 : n], w[1 : n])
    saveJatin[1 : n + 1][0 : n] : Array of size n + 1 × n + 1
    for j = 0 to n do
        saveJatin[n + 1][j] ← 0
    end for
    for i = n down to 1 do
        for j = 1 to n do
            if j < w_i then
                saveJatin[i][j] ← saveJatin[i + 1][j + 1]
            else
                saveJatin[i][j] ← max{saveJatin[i + 1][j − w_i + 1] + v_i, saveJatin[i + 1][j + 1]}
            end if
        end for
    end for
    Return saveJatin[1][1]
end procedure
```

Runtime. We are filling n^2 table entries and each of them takes constant time. Hence the complexity is $\mathcal{O}(n^2)$.

2 Inventory Management

Consider the following inventory problem. You are running a store that sells some large product (let us assume you sell trucks), and predictions tell you the quantity of sales to expect over the next n months. Let d_i denote the number of sales you expect in month i. We will assume that all sales happen at the beginning of the month, and trucks that are not sold are stored until the beginning of the next month. You can store at most S trucks, and it costs C to store a single truck for a month. You receive shipments of trucks by placing orders for them, and there is a fixed ordering fee of K each time you place an order (regardless of the number of trucks you order). You start out with no trucks. The problem is to design an algorithm that decides how to place orders so that you satisfy all the demands d_i, and minimize the costs. In summary: There are two parts to the cost.

- First, storage: it costs C for every truck on hand that is not needed that month. Second, ordering fees: it costs K for every order placed.
- Each month, you need enough trucks to satisfy the demand d_i, but the amount left over after satisfying the demand for the month should not exceed the inventory limit S.

Solution.

Subproblems. The subproblems are a bit subtle here. We define $\mathsf{OPT}(i, s)$ to be the optimal solution for months i to n given that there are s trucks at the beginning of month i (before placing any order).

Recurrence. Idea : Look at $\mathsf{OPT}(i, s)$. Now there could be two cases regarding what happens in month i

1. Suppose $s > d_i$. This is the easier case, in this case we do not order any new truck in month i, pay a storage cost of $C(s - d_i)$. The residual solution would be (at least we hope) $\mathsf{OPT}(i + 1, s - d_i)$.
2. Suppose $s \leq d_i$. In this case we need to order new trucks - note that this cost is just K irrespective of the number of trucks ordered. Now suppose you have $0 \leq s' \leq S$ trucks remaining after meeting the demand. Then you pay Cs' for the storage. The remaining solution (we hope) would be $\mathsf{OPT}(i + 1, s')$. Hence, the recurrence is :

$$\mathsf{OPT}(i,s) = \min \begin{cases} \mathsf{OPT}(i+1, s-d_i) + C(s-d_i) & \text{if } s \leq d_i \\ \min_{0 \leq s' \leq S} K + Cs' + \mathsf{OPT}(i+1, s') & \text{if } s' > d_i \end{cases}$$

What is the base case ?

Correctness. We are going to prove the correctness of the recurrence in two different case.

1. $(s \leq d_i)$: We claim that in this case $\mathsf{OPT}(i+1, s-d_i) + C(s-d_i) = \mathsf{OPT}(i,s)$. To prove this, let us take an optimal solution for the months i to n and s number of trucks being available at the start of month i. Now let us consider this schedule and look at the beginning of month $i+1$. Clearly we remain with $s-d_i$ trucks at the beginning of month $i+1$. Hence, the residual schedule is clearly a feasible solution for $i+1$ to n months when there are $s-d_i$ trucks at the beginning of month $i+1$ and the cost of this solution is $\mathsf{OPT}(i,s) - C(s-d_i)$. Now suppose for contradiction this residual solution is not $\mathsf{OPT}(i+1, s-d_i)$. Hence $\mathsf{OPT}(i+1, s-d_i) < \mathsf{OPT}(i,s) - C(s-d_i)$. Now consider a schedule where you sell d_i trucks in month i and the rest of the schedule is $\mathsf{OPT}(i+1, s-d_i)$. This schedule is a feasible schedule for months i through n when we start with s trucks. But it has cost $\mathsf{OPT}(i+1, s-d_i) + C(s-d_i) < \mathsf{OPT}(i,s)$ contradicting the optimality of $\mathsf{OPT}(i,s)$.
2. $(s > d_i)$: The proof is very similar. **Please figure it out yourself. You need to practice writing proofs**

Algorithm. Quite straightforward.

3 Knapsack cover

You are given items i with sizes w_i and values v_i, and a target value V. Find a subset of items of minimum total size with total value at least V (or decide that there is no such subset).

Solution.

Subproblem.

For $1 \leq i \leq n, 0 \leq P \leq V$,

$B(i, P)$: minimum total size of items $1, 2, \ldots, i$ whose value is at least P

Recurrence.

$$B(i, P) = \min \begin{cases} B(i-1, P) \\ B(i-1, P - v_i) + w_i \quad \text{if } P \geq v_i \end{cases}$$

Base case:

$$B(0, P) = \begin{cases} 0 \quad \text{for } P = 0 \\ \infty \quad \text{otherwise} \end{cases}$$

Correctness.

Consider two cases about the optimum solution OPT for $B(i, P)$.

1. $i \in OPT$**:** We show that $OPT \setminus i$ is optimal for the subproblem for $(i - 1, P - v_i)$. It has value $P - v_i$ and uses items $1, 2, \ldots, i-1$ and hence is feasible. To show that it has the optimal size, suppose for contradiction that, there is a solution S of smaller total size than $OPT \setminus i$. Then, $S \cup i$ is a solution of value at least $P - v_i + v_i = P$ using items $1, 2, 3, \ldots i$ and hence is feasible for the subproblem for (i, P). But then the total size of $S \cup i$ is size$(S) + w_i <$ size$(OPT \setminus i) + w_i =$ size(OPT); which contradictions the definition of OPT.
   Hence, $OPT \setminus i$ has size $B(i-1, P - v_i)$ and OPT has size $B(i-1, P - v_i) + w_i$.

2. $i \notin OPT$**:** Write yourself, similar to the above.

Pseudocode.

Write yourself.

Time Complexity.

There are nV subproblems and each takes $O(1)$ time. Hence, the time complexity is $O(nV)$.