

Lec19-subset-sum-knapsack

Thursday, October 22, 2026 1:20 PM

Recall: Dynamic programming is like divide-and-conquer in that you build up big solutions from smaller ones, but is more general/powerful in that small solutions might be used many times, generally applied in problems where brute-force would be exponential.

Subset Sum K.T. 6.4 (special easy case of knapsack problem)

Given integer bound W , and n items with integer weight $w_i \in \mathbb{N}$

find a subset S of items that maximizes $\sum_{i \in S} w_i$ while keeping $\sum_{i \in S} w_i \leq W$.

Motivation fully schedule a piece of equipment so least downtime
e.g. CPU or mass-spectrometer

Aside: integer weights very important here

Let S^* be an optimal choice of items.

Let $OPT(n, W)$ be the value of the optimal solution $OPT(n, W) = \sum_{i \in S^*} w_i$

compute this first, then keep info needed to reconstruct S^* .

Subproblems: Compute $OPT(j, w)$ for optimal value with a knapsack of size $w \leq W$ and the first j items.

Recurrence:

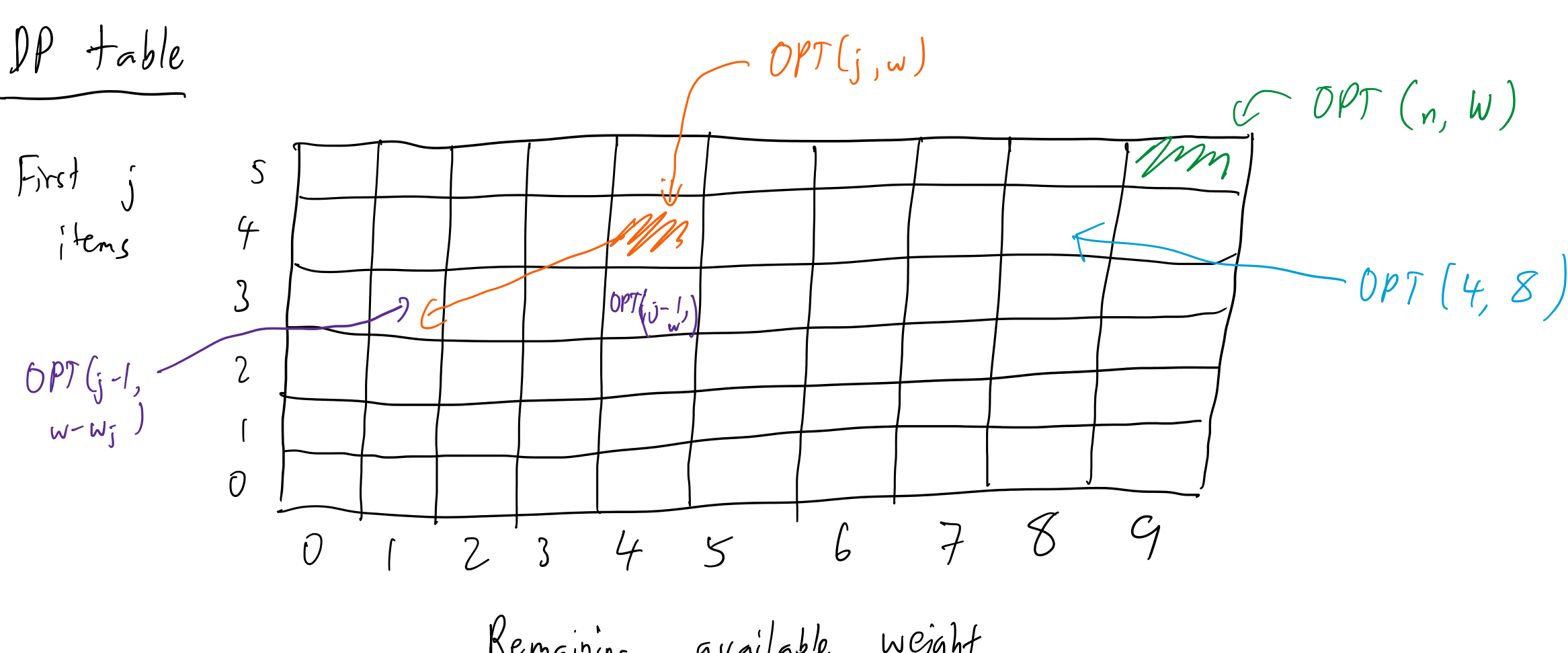
$$OPT(j, w) = \max \begin{cases} OPT(j-1, w) & \text{if } j \notin S^* \\ v_j + OPT(j-1, w-w_j) & \text{if } j \in S^* \end{cases}$$

unknown at computation time, so try both using max.

Base case: $OPT(0, W) = 0$ if no items, 0

$OPT(j, 0) = 0$ if no space, 0

DP table



Have to look at two entries in the row below

Pseudocode:

SubsetSum(n, W):

$M[0, w] = 0 \quad \forall w \in 0, \dots, W$

$M[j, 0] = 0 \quad \forall j \in 1, \dots, n$

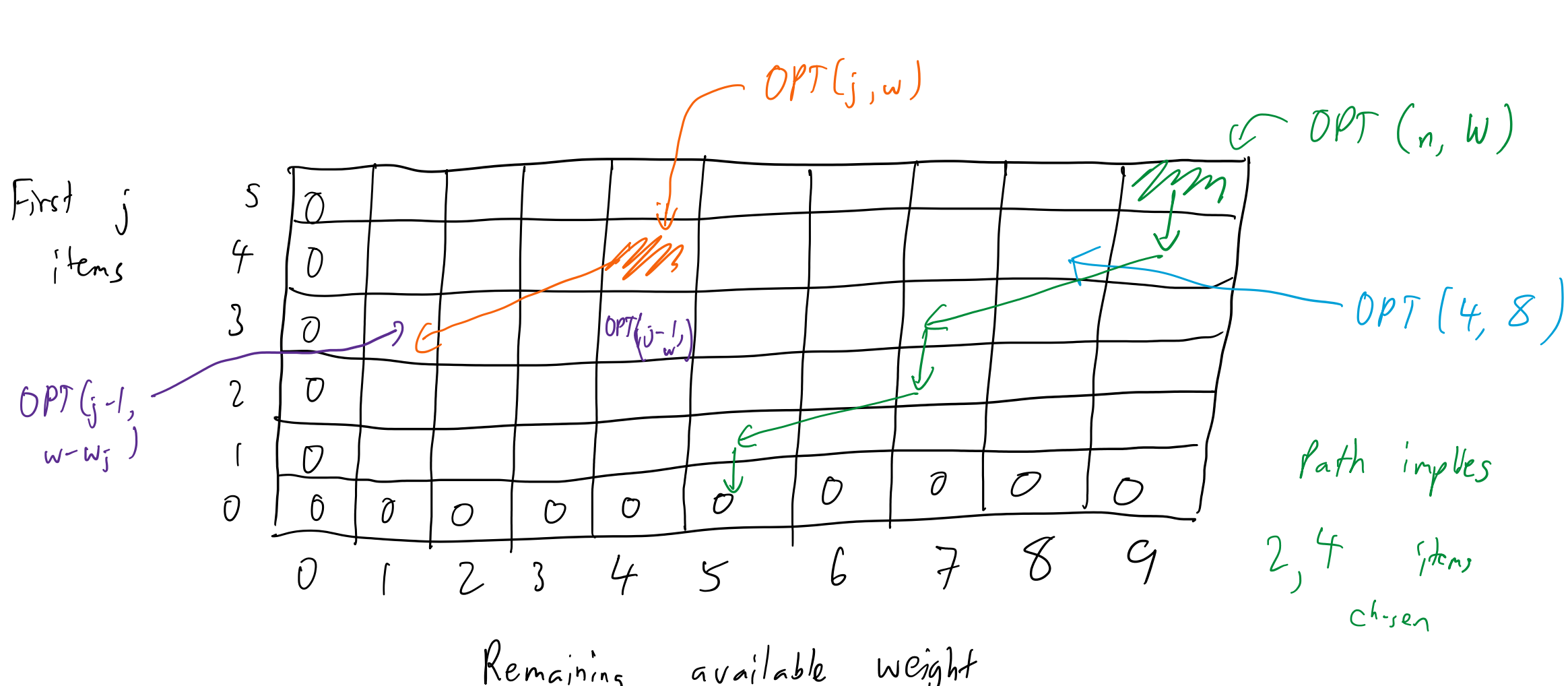
Define $M[j, r] = 0 \quad \forall r < 0$

For $j \in 1, \dots, n$:

For $w \in 0, \dots, W$:

$$M[j, w] = \max \begin{cases} M[j-1, w] \\ w_j + M[j-1, w-w_j] \end{cases}$$

also keep track of which entry was the max so we can reconstruct S^* at the end.



Path implies 2, 4 items chosen

nW cells in matrix

Each cell takes $O(1)$ time to fill.

$O(n)$ time to follow path backwards

$\Rightarrow O(nW + n) = O(nW)$ total running time.

General principles:

- Opt can be easily computed from subproblems
- Only poly number of subproblems.
- "Natural" ordering of problems from small to large, such that to answer large problem, need only small sols

Knapsack problem

• A bound W

• collection of n items, each with weight w_i , and a value v_i .

Find a subset S that

$$\text{maximizes } \sum_{i \in S} v_i \text{ while keeping } \sum_{i \in S} w_i \leq W.$$

want to maximize value instead of weight, as in subset sum.

Greedy doesn't work

Idea Sort by value per weight $p_i = \frac{v_i}{w_i}$, and pick highest value density.

1 lb bottle of ambrosia, \$30 $p_1 = \$30 / 1b$

2 lb jade paperweight, \$40 $p_2 = \$20 / 1b$

3 lb salt, \$45 $p_3 = \$15 / 1b$

4 lb iron crowbar, \$100 $p_4 = \$25 / 1b$

6 lb carrying capacity knapsack.

Greedy ambrosia, crowbar, jade, salt
1b 4lb 2lb 3lb
Total value: \$130
Total value: \$140
problem is can't divide objects.

Better crowbar, jade

$$\text{DP solution: } OPT(j, W) = \max \begin{cases} OPT(j-1, W) & \text{if } j \notin S^* \\ v_j + OPT(j-1, W-w_j) & \text{if } j \in S^* \end{cases}$$