Lec20-subset-sum-knapsack

Tuesday, October 24, 2023 9:29 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 to problems where brute-force would be exponential.

Subset Sum KT 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: the integer weights is 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:

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

unknown at computation time, so try both using max

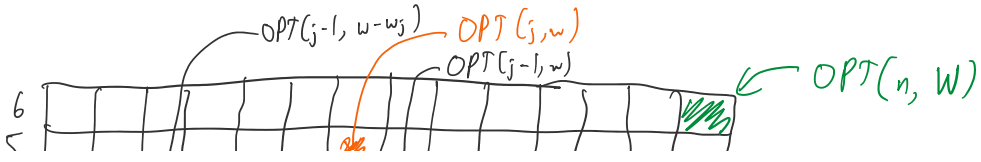
Base case:

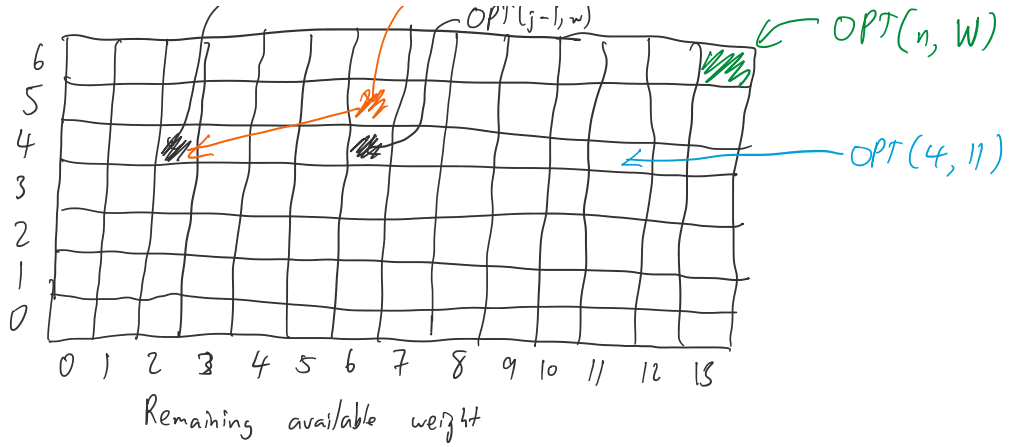
$OPT(0, W) = 0$ if no items, 0

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

Also, $OPT(j, [\text{negative number}]) = 0$

DP Table:





Have to look at two entries in the row below

Pseudocode:

SubsetSum (n, W):

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

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

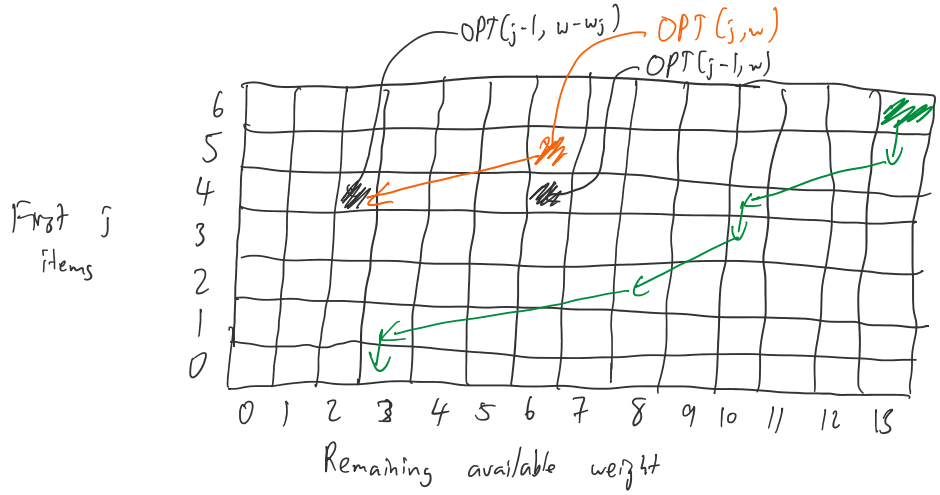
Define $M[j, r] = 0$ for all $r < 0$.

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

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

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

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



Path implies items 2, 3, 5

nW cells in matrix.

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

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

↑ pseudopolynomial because proportional to size of input numbers, and not just the number of items.

General principles:

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

---

Knapsack problem

- A bound W.
- Collection of n items, each with weight w_i, and a value v_i.

Find subset S that

maximizes $\sum_{i \in S} v_i$ 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 the highest value density.

1 lb	bottle of ambrosia,	\$30
2 lb	jade paperweight,	\$40
3 lb	salt,	\$45
4 lb	iron crowbar,	\$100

$p_1 = \$30/\text{lb}$
$p_2 = \$20/\text{lb}$
$p_3 = \$15/\text{lb}$
$p_4 = \$25/\text{lb}$

6 lb carrying capacity knapsack

Greedy: ambrosia, crowbar, ~~jade~~ Total value: \$130 ? problem is

6 lb carrying capacity knapsack

Greedy: ambrosia, 1lb | crowbar, 4lb | ~~jade~~ 2lb — Total value: $130

Better: crowbar, jade — Total value: $140

} problem is can't divide objects

DP solution:

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