Lecture 10.1

- Anurag Sharma & Shymal Chandra

Greedy Algorithms vs Dynamic Programming

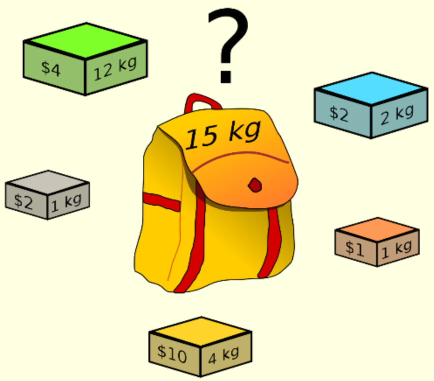
Greedy vs Dynamic Programming

- The greedy approach and dynamic programming are two ways to solve optimization problems
- When a greedy approach solves a problem, the result may be a simpler
- However, it can be difficult to determine whether a greedy algorithm always produces an optimal solution
- We will now look at The Knapsack Problem and try to solve it using both the algorithms!

Smart thief?

The Knapsack Problem

- The Knapsack Problem can be described as follows:
  - A thief breaks into a jewellery store carrying a knapsack wanting to steal items and pack it into his knapsack
  - Given n items S = {item1, item2, …, item n}, each item having a weight w_i and providing a profit p_i , which items should the thief put into his knapsack that has a maximum capacity W in order to obtain the maximum profit?



- The Knapsack problem has two variations: 0-1 Knapsack and Fractional Knapsack

Greedy Approach: The 0-1 Knapsack Problem

- This problem requires a subset A of S to be determined such that
- $maximize \sum_{i \in A} p_i \mid \sum_{i \in A} w_i \leq W$
- Greedy Strategies: Steal (select) items with the largest profit or steal items with the lightest weight etc.
- These however may not be optimal
- Another greedy strategy would be to steal items with the largest profit per unit weight

Example

- 3 items with w = [5, 10, 20], p = [50, 60,140] and W = 30
- Profits per unit = [10, 6, 7]
- Greedy Approach: Select items 1 and 3: Profit is $190, although optimal profit would be $200 (items 2 and 3)
- The problem is that even if items 1 and 3 are selected, there is wastage of space (5 units) in the knapsack (since knapsack is not filled to capacity)
- However, in the 0-1 Knapsack problem, you can either select the whole of an item or none of it – no fractions allowed, so the above problem is expected

Greedy Approach: The Fractional Knapsack Problem

- In a slight variation, the Fractional Knapsack Problem is where the thief does not have to steal all of an item, but rather can take any fraction of the item
- Greedy approach to the fractional knapsack problem yields the optimal solution
- Example: With the same strategy in the previous example (i.e. select items with the highest profit per weight value): Profit = 50 + 140 + (5/10) * 60 = $220 (where you select items 1 and 3 first and take 5/10 of item 2 to avoid any wastage of space in the knapsack) This gives you the optimal profit!

Dynamic Programming Approach: The 0-1 Knapsack Problem

- For a dynamic programming algorithm, the principle of optimality should apply
- Let A be an optimal subset of n items. There are two cases:

1. If A contains item i, the total profit of items in A is equal to $p_i +$ the optimal profit obtained from the first $i - 1$ items, where the total weight cannot exceed $W - w_i$
2. If A does not contain item i, the total profit of items in A is equal to the optimal subset of the first $i - 1$ items

Cont.

- You can create a 2-D array P (whose rows are indexed from 0 to n and columns indexed from 0 to W)
- In general, the two cases discussed on the previous slide can be represented by the following formula:
- $P[i][w] = \begin{cases} \max(P[i-1][w], p_i + P[i-1][w-w_i]), if\ w_i \leq w \\ P[i-1][w] \qquad\qquad\qquad\qquad\qquad\qquad\quad , if\ w_i > w \end{cases}$
- The maximum profit is given by the value at $P[n][w]$

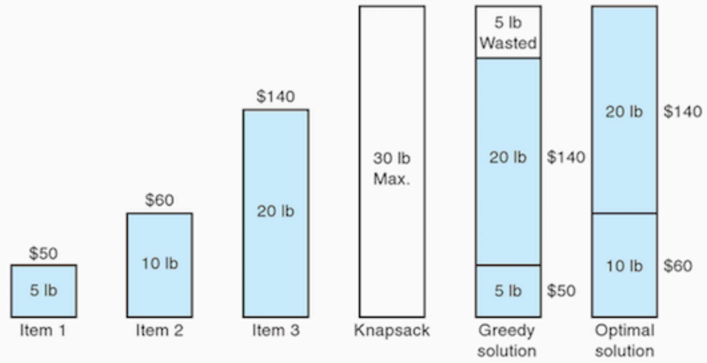


Figure 4.13 ● A greedy solution and an optimal solution to the 0-1 Knapsack problem.

• Example 4.9

Suppose we have the items in Figure 4.13 and $W = 30$. First we determine which entries are needed in each row.

Determine entries needed in row 3:
We need

$$P[3][W] = P[3][30].$$

Determine entries needed in row 2:
To compute $P[3][30]$, we need

$$P[3-1][30] = P[2][30] \qquad \text{and} \qquad P[3-1][30-w_3] = P[2][10].$$

Determine entries needed in row 1:
To compute $P[2][30]$, we need

$$P[2-1][30] = P[1][30] \qquad \text{and} \qquad P[2-1][30-w_2] = P[1][20].$$

To compute $P[2][10]$, we need

$$P[2-1][10] = P[1][10] \qquad \text{and} \qquad P[2-1][10-w_2] = P[1][0].$$

Next we do the computations.

Compute row 1:

$$P[1][w] = \begin{cases} maximum\left(P[0][w], \$50 + P[0][w-5]\right) & \text{if } w_1 = 5 \le w \\ P[0][w] & \text{if } w_1 = 5 > w, \end{cases}$$
$$= \begin{cases} \$50 & \text{if } w_1 = 5 \le w \\ \$50 & \text{if } w_1 = 5 > w. \end{cases}$$

Therefore,

$$P[1][0] = \$0$$
$$P[1][10] = \$50$$
$$P[1][20] = \$50$$
$$P[1][30] = \$50.$$

Compute row 2:

$$P[2][10] = \begin{cases} maximum(P[1][10], \$60 + P[1][0]) & \text{if } w_2 = 10 \le 10 \\ P[1][10] & \text{if } w_2 = 10 > 10 \end{cases}$$
$$= \$60.$$

$$P[2][30] = \begin{cases} maximum(P[1][30], \$60 + P[1][20]) & \text{if } w_2 = 10 \le 30 \\ P[1][30] & \text{if } w_2 = 10 > 30 \end{cases}$$
$$= \$60 + \$50 = \$110.$$

Compute row 3:

$$P[3][30] = \begin{cases} maximum(P[2][30], \$140 + P[2][10]) & \text{if } w_3 = 20 \le 30 \\ P[2][30] & \text{if } w_3 = 20 > 30 \end{cases}$$
$$= \$140 + \$60 = \$200.$$