Knapsack problem - formulation, examples of a problem and algorithms

Formulation

- Given a set of items, each with a weight and a value, determine which items to include in the collection so that the total weight is less than or equal to a given limit and the total value is as large as possible.
- There are a few versions of that problem:
  - 0/1 Knapsack problem

    Each item can be either included or excluded, you cannot take fractional parts of an item or include an item twice.
  - Fractional Knapsack

    Items can be divided into fractions, but still cannot take an item twice.
  - Bounded Knapsack problem

    The same as 0/1 knapsack, but now you get the number of copies of each item, and you can take multiple instances of that item.
  - Unbounded Knapsack

    The same as bounded knapsack, but now there is no limit on number of instances of an item.

Examples of a problem

- A project manager needs to allocate limited resources (like budget, manpower, or time) to various project tasks.
- A shipping company needs to load a cargo ship with goods, where each type of good has a different weight and value.

- An individual or organization has a fixed budget to spend on various items or investments.
- Least wasteful way to cut raw materials

Algorithms

Brute force

- Produce every possible combination and calculate the sum of values for each to find the best.
- It has $O(n!)$ complexity

Greedy

- It doesn't always produce the best solution for the 0/1 knapsack problem, it is a heuristic. But for the fractional it always gives the best solution.
- Algorithms:
  1. For each item, compute the value-to-weight ratio
  2. Sort the items based on that ratio
  3. Insert the item with the greatest ratio into the stack
  4. For the fractional knapsack problem if the next item doesn't fit fully, take a fraction of that item.

Dynamic programming for the 0/1 knapsack

- Assume weights are positive integers.
- This method has time complexity $O(n \times W)$ where n is the number of items and W is the value of the weight limit.
- Define $dp[i, w]$ to be the maximum value that can be attained with first i items that have the total weight less than or equal to w.
- We can define dp recursively as follows:

- $dp[0, w] = 0$
- $dp[i, w] = dp[i-1, w]$ if the new item doesn't fit into the knapsack, or:
- $dp[i, w] = \max(dp[i-1, w],\ dp[i-1, w-w_i] + v_i)$ if the new item does fit into the knapsack

- We can then solve the problem by filling the table from the top to bottom and taking the value $dp[n, W]$ where n is the number of items and W is the weight limit.
- Example of such a table:

	-	0	1	2	3	4	5	6	7	8	9	10
	0 items	0	0	0	0	0	0	0	0	0	0	0
5	Item 1 10	0	0	0	0	0	10	10	10	10	10	10
4	Item 2 40	0	0	0	0	**40**	40	40	40	40	50	50
6	Item 3 30	0	0	0	0	40	40	30/40 → 40	40	40	50	40/30 → 70
3	Item 4 50	0	0	0	50	50	50	50	90	90	90	**90**

```
def knapsack(weights, values, limit):
    dp = [[0 for _ in range(limit+1)] for _ in range(len(weights
    for i in range(1, len(weights)+1):
        for j in range(1, limit+1):
            if weights[i-1] > j:
                dp[i][j] = dp[i-1][j]
            else:
                dp[i][j] = max(
                    dp[i-1][j],
                    dp[i-1][j-weights[i-1]] + values[i-1]
                )
    return dp[len(weights)][limit]
```