

0/1 Knapsack

The grandfather of all DP — take it or leave it ■

● MEDIUM

■ Hook

You have a knapsack of capacity W and n items, each with a weight and value. Maximize total value without exceeding capacity. Each item: use it or don't (0 or 1). This is the **template problem** — master it and all knapsack variants follow.

■ Core Intuition

$dp[i][w]$ = max value using first i items with capacity w . For each item i : either **skip it** ($dp[i-1][w]$) or **take it** ($dp[i-1][w-wt[i]] + val[i]$). Take the max. 'i-1' because item i can only be taken once.

■ DP Grid

```
Items: wt=[1,3,4,5]  val=[1,4,5,7]  W=7
      w→  0  1  2  3  4  5  6  7
i=0 [ ]  [0  0  0  0  0  0  0  0]
i=1 [1]  [0  1  1  1  1  1  1  1]
i=2 [3]  [0  1  1  4  5  5  5  5]
i=3 [4]  [0  1  1  4  5  6  6  9]
i=4 [5]  [0  1  1  4  5  7  8  9]
★ ANSWER = dp[4][7] = 9  (items with wt=3,4: val=4+5=9)
```

■ Code

```
public int knapsack(int W, int[] wt, int[] val, int n) {
    int[][] dp = new int[n+1][W+1];
    for (int i=1; i<=n; i++) {
        for (int w=0; w<=W; w++) {
            dp[i][w] = dp[i-1][w]; // skip item i
            if (wt[i-1] <= w) // can we take item i?
                dp[i][w] = Math.max(dp[i][w],
                                     dp[i-1][w-wt[i-1]] + val[i-1]);
        }
    }
    return dp[n][W];
}
```

■ TIME

$O(n \times W)$

■ SPACE

$O(n \times W) \rightarrow O(W)$ with 1D

■ 1D Space Optimization

Iterate w from W down to $wt[i]$ (backward!) to prevent reusing item in same row. $for(w=W; w \geq wt[i]; w--): dp[w] = \max(dp[w], dp[w-wt[i]] + val[i])$

■ Practice

#	Problem	Diff	Pattern
1	0/1 Knapsack	Medium	Classic 2D DP
2	Partition Equal Subset Sum (LC 416)	Medium	Knapsack variant
3	Target Sum (LC 494)	Medium	Knapsack / DFS
4	Last Stone Weight II (LC 1049)	Medium	Knapsack
5	Ones and Zeroes (LC 474)	Medium	2D Knapsack

- ✓ $dp[i][w] = \max$ value with i items and capacity w
- ✓ Skip: $dp[i][w] = dp[i-1][w]$
- ✓ Take: $dp[i][w] = dp[i-1][w-wt[i]] + val[i]$ (if $wt[i] \leq w$)
- ✓ Answer = $dp[n][W]$
- ✓ 1D optimization: iterate w backward to avoid reuse

■ MENTAL MODEL — Remember This Forever

The knapsack table is a 'budget planner'. Rows = items considered. Columns = budget available. Each cell asks: 'Given I can only use the first i items and have w budget, what's the best I can do?' The answer comes from the best of two worlds: excluding the current item, or including it.