Dynamic Programming Recipe

- **Step 1**: Devise simple recursive algorithm
  - Flavor: make “first choice”, then recursively solve remaining part of the problem
- **Step 2**: Write recurrence for optimal value
- **Step 3**: Design bottom-up iterative algorithm
  - Weighted interval scheduling: first-choice is binary
  - Rod-cutting: first choice has \( n \) options
  - Subset Sum: “add a variable” (one more dimension)

Subset Sum: Problem Formulation

- **Input**
  - Items 1, 2, \ldots , \( n \)
  - Weights \( w_i \) for all items (integers)
  - Capacity \( W \)
- **Goal**: select a subset \( S \) whose total weight is as large as possible without exceeding \( W \).
- **Example**: \( W = 6 \), items: \( w_1 = 2, w_2 = 2, w_3 = 3 \).

Clicker Question

\[
\text{SubsetSum}(j)
\]

- if \( j = 0 \) then return 0
- \( v_{\text{max}} = \text{SubsetSum}(j-1) \) \hspace{1cm} ▶ Case 1: \( j \notin O \)
- if \( w_j \leq W \) then \hspace{1cm} ▶ Case 2: \( j \in O \)
  \[ v_{\text{max}} = \max(v_{\text{max}}, w_j + \text{SubsetSum}(j-1)) \]
  return \( v_{\text{max}} \)

Is there a problem in Case 2?

- A. No. It is correct
- B. Yes, must consider selecting \( j^{\text{th}} \) item multiple times
- C. Yes, if we take item \( j \), the remaining capacity changes

Second call to SubsetSum(j-1) no longer has capacity \( W \).
Solution: must add extra parameter (problem dimension)

Step 1: Recursive Algorithm, Binary Choice

- Let \( O \) be optimal solution on items 1 to \( j \). Is \( j \in O \) or not?
- \( \text{SubsetSum}(j) \)
  - if \( j = 0 \) then return 0
  - ▶ Case 1: \( j \notin O \)
    \[ v_{\text{max}} = \text{SubsetSum}(j-1) \]
  - ▶ Case 2: \( j \in O \)
    if \( w_j \leq W \) then
    \[ v_{\text{max}} = \max(v_{\text{max}}, w_j + \text{SubsetSum}(j-1)) \]
  return \( v_{\text{max}} \)

Clicker Question

\[
\text{SubsetSum}(j, w)
\]

- if \( j = 0 \) then return 0
  ▶ Case 1: \( j \notin O \)
  \[ v_{\text{max}} = \text{SubsetSum}(j-1, w) \]
  ▶ Case 2: \( j \in O \)
  if \( w_j \leq w \) then
  \[ v_{\text{max}} = \max(v_{\text{max}}, w_j + \text{SubsetSum}(j-1, w-w_j)) \]
  return \( v_{\text{max}} \)
Recurrence

- Let $\text{OPT}(j, w)$ be the maximum weight of any subset of items $\{1, \ldots, j\}$ that does not exceed $w$.

$$\text{OPT}(j, w) = \begin{cases} 
\text{OPT}(j - 1, w) & w_j > w \\
\max \left\{ \text{OPT}(j - 1, w), w_j + \text{OPT}(j - 1, w - w_j) \right\} & w_j \leq w 
\end{cases}$$

- Base case: $\text{OPT}(0, w) = 0$ for all $w = 0, 1, \ldots, W$.

Questions

- Do we need a base case for $\text{OPT}(j, 0)$? No.
- What is overall optimum to original problem? $\text{OPT}(n, W)$

Step 3: Iterative Algorithm

SubsetSum($n, W$)

Initialize array $M[0..n, 0..W]$.

Set $M[0, w] = 0$ for $w = 0, \ldots, W$.

For $j = 1$ to $n$ do

  For $w = 1$ to $W$ do

    If $w_j > w$ then $M[j, w] = M[j - 1, w]$;

    Else $M[j, w] = \max(M[j - 1, w], w_j + M[j - 1, w - w_j])$.

Return $M[n, W]$. Can we switch inner loop ($j$) and outer loop ($w$)? Yes.

- Running Time? $\Theta(nW)$.

Clicker Question

- For $j = 1$ to $n$ do

  For $w = 1$ to $W$ do

    If $w_j > w$ then $M[j, w] = M[j - 1, w]$;

    Else $M[j, w] = \max(M[j - 1, w], w_j + M[j - 1, w - w_j])$.

Suppose we have $n$ items, and the total capacity has $m$ decimal digits. Then the complexity is:

- A. $\Theta(nm)$
- B. $\Theta(n \log_{10} m)$
- C. $\Theta(10^m)$
- D. $\Theta(10^{nm})$

The capacity is $W \approx 10^m$. The complexity is $\Theta(n10^m)$.

Polynomial vs. Pseudo-polynomial

- If numbers have $m$ digits, input size is $\Theta(nm)$, runtime is $\Theta(n10^m)$.
- Polynomial time means polynomial in input size ($nm$).

For numeric problems, input size is log of magnitude of the numbers. Poly-time algorithm should be polynomial in $n$ and log $W$.

- Subset sum is pseudo-polynomial: polynomial in number of items ($n$) and magnitude of numbers ($W \approx 10^m$)

- Thus exponential in input size of numbers (digit count $m$).
- No polynomial-time algorithm is known. Discuss again with NP-completeness.

Subset Sum

Source: https://www.xkcd.com/287/

0–1 Knapsack Problem

Introduce an additional parameter, value

- Input

  - Items $1, 2, \ldots, n$

  - Weights $w_i$ for all items (integers)

  - Values $v_i$ for all items (integers)

  - Capacity $W$

- Goal: select a subset $S$ whose total value is as large as possible without exceeding $W$.

- Does the solution change?
Clicker Question

Recall recurrence for subset sum: \( \text{OPT}(j, w) \)
\[
= \begin{cases} 
\text{OPT}(j - 1, w) & w_j > w \\
\max(\text{OPT}(j - 1, w), w_j + \text{OPT}(j - 1, w - w_j)) & w_j \leq w 
\end{cases}
\]

What is the correct change for the blue term in the recurrence?
A. \( w_j + \text{OPT}(j - 1, w - w_j) \)
B. \( w_j + \text{OPT}(j - 1, w - v_j) \)
C. \( v_j + \text{OPT}(j - 1, w - v_j) \)
D. \( v_j + \text{OPT}(j - 1, w - w_j) \)

Optimum adds values \( v_j \), subtracts weights \( w_j \) from capacity (D)

Clicker Question

Does our knapsack solution still work if the weights and/or values are real numbers instead of integers?
A. It still works if both the values and weights are real numbers.
B. It works if weights are real numbers but values are integers.
C. It works if values are real numbers but weights are integers.
D. It does not work if either weights or values are real numbers.

Weights are the second dimension: can index on ints, not reals. (C)

Real weights \( \Rightarrow \) consider all \( 2^n \) choices.

Fractional knapsack problem: allows partial objects
(think: grains, sand, …)

Simple greedy solution: choose highest value per weight