Dynamic Programming

LP Final Projetc

Raina & Noura

University of Colorado Denver

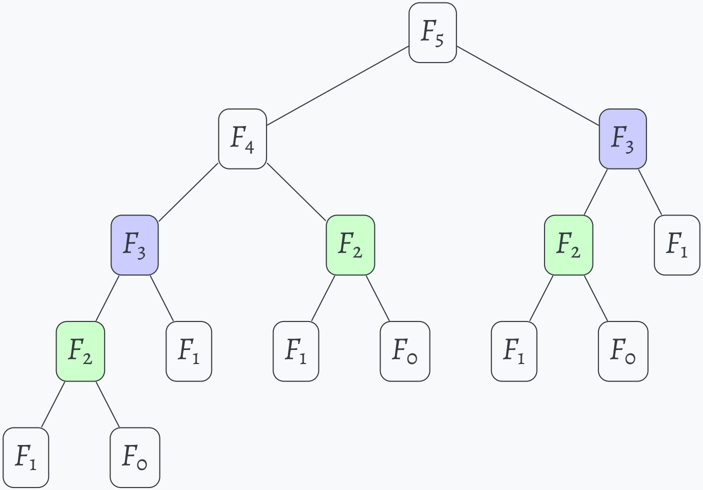
Reduce complexity of ***NP-HARD*** problems.
Problems must have

1. ***OPTIMAL SUBSTRUCTURE***
2. ***OVERLAPPING SUB-PROBLEMS***

Optimal Substructure

- How many subproblems can an optimal solution to the original problem use
- how many choices we have in determining which subproblem(s) to use in an optimal solution

Overlapping Sub-Problems



NP-Hard/Complexity Classes

Save Time by Using More Space

- **Memoization:** Top-Down
- **Tabulation:** Bottom-Up

A Framework for Solving DP Problems

1. Define the Subproblems
2. Define the Recurrence Relation
3. Identify the Base Case
4. Build the Algorithm
5. Analyze the Time and Space Complexity

The ***Knapsack*** problem (NP-Hard)

$$\max \quad \sum_{i=1}^{n} v_i x_i$$

$$\text{s.t.} \quad \sum_{i=1}^{n} w_i x_i \leqslant W$$

$$x \in \{0, 1\}$$

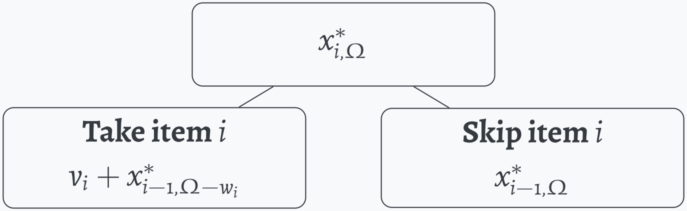
The Knapsack Problem

1/5: Define the Subproblems

	Weight capacity				
Item Number	0	1	2	...	W
0	$x^*_{0,0}$	$x^*_{0,1}$	$x^*_{0,2}$	$\dots$	$x^*_{0,W}$
1	$x^*_{1,0}$	$x^*_{1,1}$	$x^*_{1,2}$	$\dots$	$x^*_{1,W}$
2	$x^*_{2,0}$	$x^*_{2,1}$	$x^*_{2,2}$	$\dots$	$x^*_{2,W}$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
n	$x^*_{n,0}$	$x^*_{n,1}$	$x^*_{n,2}$	$\dots$	$x^*_{n,W}$

The Knapsack Problem

2/5: Define the Recurrence Relation



$$x^*_{i,\Omega} = \max\{(v_i + x^*_{i-1,\Omega-w_i}), (x^*_{i-1,\Omega})\}$$

The Knapsack Problem

3/5: Identify the Base Case

	Weight capacity				
Item Number	0	1	2	…	W
0	0	0	0	…	0
1	0	$x^*_{1,1}$	$x^*_{1,2}$	…	$x^*_{1,W}$
2	0	$x^*_{2,1}$	$x^*_{2,2}$	…	$x^*_{2,W}$
⋮	⋮	⋮	⋮	⋱	⋮
n	0	$x^*_{n,1}$	$x^*_{n,2}$	…	$x^*_{n,W}$

The Knapsack Problem

4/5: Build the algorithm (DP approach).

```
for $i = 1, \dots, n$
    for $\Omega = 1, \dots W$
```

$$DP[i][\Omega] = \begin{cases} DP[i-1][\Omega], & w_i < \Omega \\ \max(DP[i-1][\Omega],\ v_i + DP[i-1][\Omega - w_i]), & \text{otherwise} \end{cases}$$

return $DP[n][W]$

The Knapsack Problem

5/5: Analyze the time and space complexity.

Brute Force	Dynammic Programming
Time: $O(2^n)$	Time: $O(nW)$
Space: $O(n)$	Space: $O(nW)$