Suggested Approach

I. identify subproblem in english
   \[ \text{OPT}(i) = \text{“optimal schedule of} \{i, \ldots, n\} \text{ (sorted by start time)”} \]

II. specify subproblem recurrence
   \[ \text{OPT}(i) = \max(\text{OPT}(i + 1), v_i + \text{OPT(\text{next}(i))}) \]

III. identify base case
   \[ \text{OPT}(n + 1) = 0 \]

IV. write iterative DP.
   (see last thurs)

Interval Pricing

input:
- \( n \) customers \( S = \{1, \ldots, n\} \)
- \( T \) days.
- \( i \)'s ok days: \( I_i = \{s_i, \ldots, f_i\} \)
- \( i \)'s value: \( v_i \in \{1, \ldots, V\} \)

output:
- prices \( p[t] \) for day \( t \).
  - consumer \( i \) buys on day \( t_i = \arg\min_{t \in I_i} p[t] \) if \( p[t_i] \leq v_i \).
  - revenue = \( \sum_{i \text{ that buys } p[t_i]} \).
  - goal: maximize revenue.
Dynamic Programming: Finding Subproblems

“find a first decision you can make which breaks problem into pieces that
(a) do not interact (across subproblems)
(b) can be describe succinctly.”

Example: Integer Knapsack

input:  
- $n$ objects $S = \{1, \ldots, n\}$
  - $s_i =$ size of object $i$ (integer).
  - $v_i =$ value of object $i$.
  - capacity $C$ of knapsack (integer)

output:  
- subset $K \subseteq S$ of objects that
  (a) fit in knapsack together
     (i.e., $\sum_{i \in K} s_i \leq C$)
  (b) maximize total value
     (i.e., $\sum_{i \in K} v_i$)

Greedy fails, e.g.,

- largest value/size:
  $v = (C/2 + 2, C/2, C/2)$.
  $s = (C/2 + 1, C/2, C/2)$.

- smallest value/size:
  $v = (1, C/2, C/2)$.
  $s = (2, C/2, C/2)$.

Question: What is “first decision we can make” to separate into subproblems?

Answer: Is item 1 in the knapsack or not?
Step I: identify subproblem in Correctness English

\[ \text{OPT}(j, D) = \text{“value of optimal size } D \text{ knapsack on } \{j, \ldots, n\}” \]

Step II: write recurrence

\[ \text{OPT}(j, D) = \max(v_j + \text{OPT}(j + 1, D - s_j), \text{OPT}(j + 1, D)) \]

if \( s_j \leq D \)

Step III: base case

\[ \text{OPT}(n + 1, D) = 0 \text{ (for all } D) \]

Step IV: iterative DP

Algorithm: knapsack

1. \( \forall D, \text{memo}[n + 1, D] = 0. \)
2. for \( i = n \) down to 1,
   for \( D = C \) down to 0,
   (a) if \( i \) fits (i.e., \( s_i \leq D \))
   \[ \text{memo}[j, D] = \max[\text{memo}[j + 1, D], v_j + \text{OPT}(j + 1, D - s_j)] \]
   (b) else,
   \[ \text{memo}[j, D] = \text{memo}[j + 1, D] \]
3. return memo[1, C]
Example: Interval Pricing

input:
• \(n\) customers \(S = \{1, \ldots, n\}\)
• \(T\) days.
• \(i\)'s ok days: \(I_i = \{s_i, \ldots, f_i\}\)
• \(i\)'s value: \(v_i \in \{1, \ldots, V\}\)

output:
• prices \(p[t]\) for day \(t\).
• consumer \(i\) buys on day \(t_i = \arg\min_{t \in I_i} p[t]\) if \(p[t_i] \leq v_i\).
• revenue = \(\sum_i\) that buys \(p[t_i]\).
• goal: maximize revenue.

Example:

Step I: identify subproblem in English

\[
\text{OPT}(s, f, p)
\]

Correctness

induction

Runtime

• precompute \(\text{Rev}(t, p)\) in \(O(nV)\) time.
• size of table: \(O(n^2V)\)
• cost of combine: \(O(nV)\).
• total: \(O(n^3V^2)\).

Note: can be improved to \(O(n^4)\) with slightly better program.