Announcements:

- Midterm: Tuesday, Oct 27, in class.
  - one handwritten cheat sheet.
  - through dynamic programming.
  - midterm reviews:
    - Taggart, Sunday,
    - Hartline, Monday, 6-8pm,

Reading: 6.4, 6.8

Last time:

- Dynamic Programming
- Weighted Interval scheduling

Today:

- D.P. (cont.)
- Integer Knapsack
- Interval Pricing.

Suggested Approach

I. identify subproblem in english

   OPT(i) = "optimal schedule of $\{i, \dots, n\}$ (sorted by start time)"

II. specify subproblem recurrence

   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 $S = \{1, \dots, n\}$
- T days.
- i's ok days: $I_i = \{s_i, \dots, f_i\}$
- i's value: $v_i \in \{1, \dots, V\}$

output:

- prices $p[t]$ for day t.
- consumer i buys on day $t_i = \text{argmin}_{t \in I_i} p[t]$ 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:

  $\mathbf{v} = (C/2 + 2, C/2, C/2)$.

  $\mathbf{s} = (C/2 + 1, C/2, C/2)$.

- smallest value/size:

  $\mathbf{v} = (1, C/2, C/2)$.

  $\mathbf{s} = (2, C/2, C/2)$.

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

Answer: Is item 1 in the knapsack or not?

- if 1 in knapsack:

  value of knapsack is v_i + optimal knapsack value on $S \setminus \{1\}$ with capacity $C - s_1$.

- if 1 not in knapsack:

  value of knapsack is optimal knapsack on $S \setminus \{1\}$ with capacity C.

Succinct description:

- remaining objects $\{j, \ldots, n\}$ represented by "j"
- remaining capacity represented by $D \in \{0, \ldots, C\}$.

Step I: identify subproblem in English

$\mathrm{OPT}(j, D)$

$= $ "value of optimal size D knapsack on $\{j, \ldots, n\}$"

Step II: write recurrence

$\mathrm{OPT}(j, D)$

$$= \max(\underbrace{v_j + \mathrm{OPT}(j+1, D - s_j)}_{\text{if } s_j \leq D}, \mathrm{OPT}(j+1, D))$$

Step III: base case

$\mathrm{OPT}(n+1, D) = 0$ (for all D)

Step IV: iterative DP

Algorithm: knapsack

1. $\forall D$, $\mathrm{memo}[n+1, D] = 0$.
2. for $i = n$ down to 1,

   for $D = C$ down to 0,

   (a) if i fits (i.e., $s_i \leq D$)

   $\mathrm{memo}[j, D] = \max[\mathrm{memo}[j+1, D],$

   $v_j + \mathrm{OPT}(j+1, D - s_j)]$

   (b) else,

   $\mathrm{memo}[j, D] = \mathrm{memo}[j+1, D]$
3. return $\mathrm{memo}[1, C]$

Correctness

induction

Runtime

$$\begin{aligned} T(n, C) &= O(\# \text{ of subprobs} \times \text{cost per subprob}) \\ &= O(nC). \end{aligned}$$

Note: not polynomial time.

Example: Interval Pricing

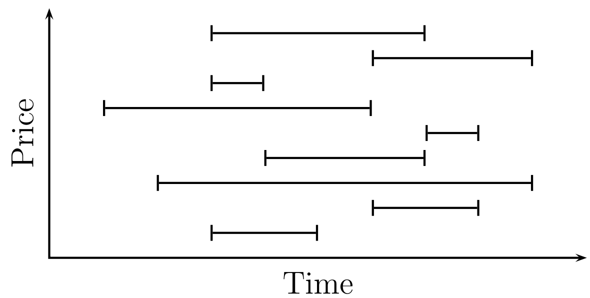
input:

- 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 $t_i = \text{argmin}_{t \in I_i} p[t]$ if $p[t_i] \leq v_i$.
- revenue $= \sum_{i \text{ that buys}} p[t_i]$.
- goal: maximize revenue.

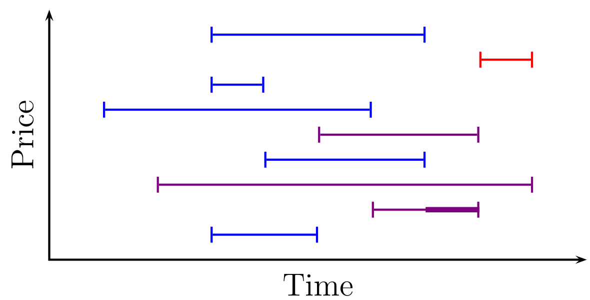
Example:



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

Answer: day and price of smallest price.

Example:



Step I: identify subproblem in English

OPT(s, f, p)

= "optimal revenue from intervals strictly between s and f with minimum price at least p"

Step II: write recurrence

OPT(s, f, p)

$$= \max_{s<t<f, q \geq p} \text{Rev}(t, p) + \text{OPT}(s, t, q) + \text{OPT}(t, f, q).$$

Step III: base case

- OPT$(s, s+1, p) = 0$.
- OPT$(s, t, P+1) = 0$.

Step IV: iterative DP

(exercise)

Correctness

induction

Runtime

- precompute 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.