EECS 336: Introduction to Algorithms	Lecture 10
Dynamic Programming (cont)	Integer Knapsack, Interval pricing

Reading: 6.4, 6.8,
"guide to dynamic programming" (Canvas)

Last time:

- Dynamic Programming (a derivation)
- Weighted interval scheduling

Today:

- Dynamic Programming (a framework)
- Integer Knapsack
- Interval Pricing.

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

Suggested Approach

I. identify subproblem in english

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

II. specify subproblem recurrence

   $\mathrm{OPT}(i) = \max(\mathrm{OPT}(i+1), v_i + \mathrm{OPT}(\mathrm{next}(i)))$

III. solve original problem (from subproblems)

   Optimal Interval Schedule = OPT(1)

IV. identify base case

   $\mathrm{OPT}(n+1) = 0$

V. write iterative DP.

   (see last thurs)

VI. analyze runtime.

   $O(n \log n)$

VII. (for homework) implement iterative DP.

   (any language most students can read. e.g., Python)

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., $\sum_{i \in K} v_i$)

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

OPT(j, D)

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

Step II: write recurrence

OPT(j, D)

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

Justification: either i is in or not (exhaustive).

Step III: solve original problem

Value of Optimal Knapsack $= \mathrm{OPT}(1, C)$

Step IV: base case

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

Step V: iterative DP

Algorithm: knapsack

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

   for $D = C$ down to 0,

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

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

   (b) else,

   memo$[j, D] =$ memo$[j+1, D]$
3. return memo$[1, C]$

VI: Runtime

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

Note: not polynomial time.

VII: implementation

(see "guide")

Alternative Approach

"isolate previously made decisions"

Suppose:

- already processed jobs $\{1, \ldots, i\}$, and
- used capacity D.

Note: previous decisions succinctly summarized by i and D

Part I: subproblem in english

OPT(i, D) = "value from remaining knapsack if

- already processed jobs $\{1, \ldots, i\}$
- used capacity D."

Part II: recurrence

$\text{OPT}(i, D) = max(v_i + \text{OPT}(i+1, D+s_i), OPT(i+1, D))$

(assuming $D + s_i \leq C$)

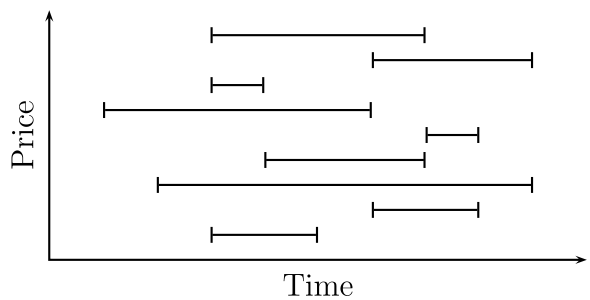
...

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

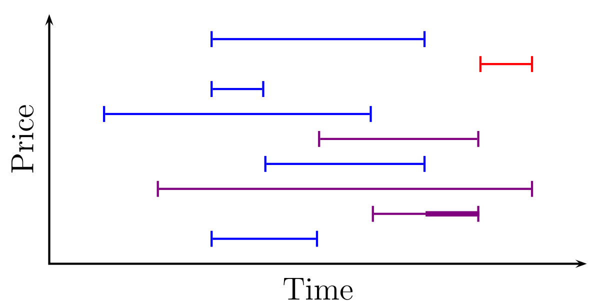
Example:



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)

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

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 $O(TVn)$ time.
- size of table: $O(T^2V)$
- cost of combine: $O(TV)$.
- total: $O(T^3V^2)$ (assuming $n < T^2V$).

Note: without loss of generality T, V are $O(n)$ so runtime is $O(n^5)$

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