EECS 336: Introduction to Algorithms — **Lecture 6**
Dynamic Programming (cont.) — interval pricing

Reading: (5.5)

Last time:

- Shortest-paths (Bellman-Ford Alg)
- sequence alignment

Today:

- interval pricing
- summary of dynamic programming
- comparison to divide and conquer
- (integer multiply)

Example: Interval Pricing

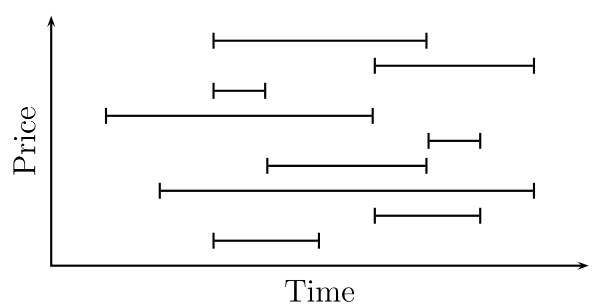
input:

- 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.

Example:

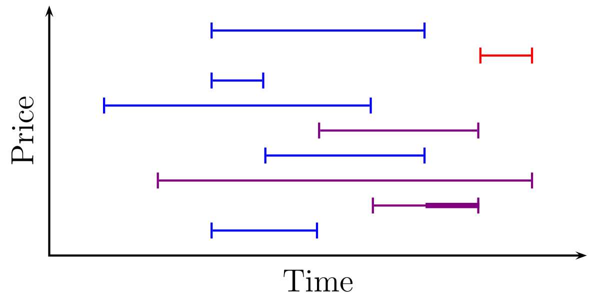


let's use dynamic programming. subproblem?

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 customers i with intervals $\{s_i, \ldots, f_i\}$ containd within interval $\{s+1, \ldots, f-1\}$ with minimum price at least p"

Step II: write recurrence

OPT(s, f, p)

$$= \max_{t \in \{s+1,\ldots,f-1\}; q \in \{p,\ldots,V\}} \mathrm{Rev}(s, t, f, p) + \mathrm{OPT}(s, t, q) + \mathrm{OPT}(t, f, q).$$

Rev(s, t, f, p) = "the revenue from customers i with intervals $\{s_i, \ldots, f_i\}$ containd within interval $\{s+1, \ldots, f-1\}$ with price p"

Step III: value of optimal solution

- optimal interval pricing = OPT$(1, T, 0)$

Step IV: base case

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

Step V: iterative DP

(exercise)

Correctness

induction

Step VI: Runtime

- precompute Rev(s, t, f, p) in $O(T^3 V n)$ time.
- size of table: $O(T^2 V)$
- cost of combine: $O(TV)$.
- total: $O(T^3 V (V + n))$

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.

Step VII: implementation

(exercise)

Summary of Dynamic Programming

"divide problem into small number of subproblems and **memoize** solution to avoid redundant computation"

Finding Subproblems

- identify a first decision, subproblems for each outcome of decision.
- partition problem, sumarize information from one part needed to solve other part.

Subproblem Properties

1. succinct
   (only a polynomial number of them)
2. efficiently combinable.
3. depend on "smaller" subproblems (avoid infinite loops), e.g.,
   - process elements "once and for all" [[*today*]]
   - "measure of progress/size". [[*coming soon*]]

Runtime Analysis

runtime $=$ initialization $+$ size of table $\times$ cost to combine

Finding Solution

- write DP to identify value of optimal solution.
- traverse memoization table to determine actual solution.

Divide and Conquer

- divide problem into subproblems
- solve subproblems
- merge solutions to solve original.

Example: repeated squaring, sorting, many data structures

Note: subproblem dependency graph vs dynamic programming

- DP: dependencies are directed acyclic graph.
- D&C: dependencies are tree.

Integer multiplication

input: n bit integers x, y.

output: $2n$ bit integer $z = x \cdot y$.

Algorithm: elementary school multiply

```
        101101
      x 010110
    ----------
        000000
       101101
      101101
     000000
    101101
 + 000000
--------------
whatever
```

Runtime: $T(n) = O(n^2)$.

can we do better?

Idea:

1. separate high order from low order bids
   - $k = n/2$ [[*assume n even*]]
   - x_H = high k bits of x
   - x_L = low k bits of x

   $\Rightarrow x = x_H 2^k + x_L$.
2. $x \cdot y = (x_H 2^k + x_L)(y_H 2^k + y_L)$

   $= x_H y_H 2^n + (x_L y_H + x_H y_L) 2^k + x_L y_L$

$\Rightarrow$ one n bit mult requires 4 $n/2$ bit mults

mult by 2^k is bit shift (easy)

$\Rightarrow T(n) = 4T(n/2) + cn$

additions require cn time

$= O(n^2)$.

need a better idea!

- let $H = x_H y_H$; $L = x_L y_L$; and $Z = x_H y_L + x_L y_H$

Q: compute H, L, and Z in < 4 mults?

Idea:

- $P = (x_H + x_L)(y_H + y_L)$

  $= x_H y_H + x_H y_L + x_L y_H + x_L y_L$

  $= H + Z + L$

3. Rearrange: $Z = P - H - L$

$\Rightarrow xy = H 2^n + (P - H - L) 2^k + L$

$\Rightarrow$ 3 size $n/2$ mults needed.

Runtime: $T(n) = 3T(n/2) + cn$

$= O(n^{\log_2 3}) = O(n^{1.59})$.

■ THIS SHOULD BE SURPRISING!

(Google: Arthur Benjamin does "Mathemagic")

```
35 x 51
= 15x100 + (8 * 6 - 15 - 5)x10 + 5
=           \_____ 28 _____/
= 1785
```