CSE 421: Introduction to Algorithms

Dynamic Programming

1

"Dynamic Programming"

Program — A plan or procedure for dealing with some matter – Webster's New World Dictionary

Dynamic Programming

- Examples: 5.10, 6.8
- Today:
  - Example 1 – Licking Stamps
  - General Principles
  - Example 2 – Knapsack
- Tomorrow
  - Example 3 – Sequence Comparison

Licking Stamps

- Given:
  - Large supply of 5¢, 4¢, and 1¢ stamps
  - An amount N
- Problem: choose fewest stamps totaling N

How to Lick 27¢

# of 5¢ Stamps	# of 4¢ Stamps	# of 1¢ Stamps	Total Number
5	0	2	7
4	1	3	8
3	3	0	6

Moral: Greed doesn't pay

A Simple Algorithm

- At most N stamps needed, etc.

```
for a = 0, …, N {
  for b = 0, …, N {
    for c = 0, …, N {
      if (5a+4b+c == N && a+b+c is new min)
        {retain (a,b,c);}}}
output retained triple;
```

- Time: $O(N^3)$
  (Not too hard to see some optimizations, but we're after bigger fish…)

Better Idea

Theorem: If last stamp licked in an optimal solution has value v, then previous stamps form an optimal solution for N-v.

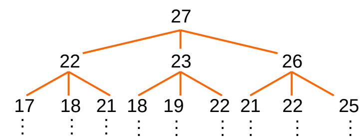
Proof: if not, we could improve the solution for N by using opt for N-v.

$$M(i) = \min\begin{Bmatrix} 0 & i=0 \\ 1+M(i-5) & i\geq 5 \\ 1+M(i-4) & i\geq 4 \\ 1+M(i-1) & i\geq 1 \end{Bmatrix}$$

where $M(i)$ = min number of stamps totaling i¢

New Idea: Recursion

$$M(i) = \min\begin{Bmatrix} 0 & i=0 \\ 1+M(i-5) & i\geq 5 \\ 1+M(i-4) & i\geq 4 \\ 1+M(i-1) & i\geq 1 \end{Bmatrix}$$



Time: > $3^{N/5}$

Another New Idea: Avoid Recomputation

- Tabulate values of solved subproblems
  - Top-down: "memoization"
  - Bottom up:

    for i = 0, …, N do $M(i) = \min\begin{Bmatrix} 0 & i=0 \\ 1+M(i-5) & i\geq 5 \\ 1+M(i-4) & i\geq 4 \\ 1+M(i-1) & i\geq 1 \end{Bmatrix}$;
- Time: O(N)

Finding How Many Stamps

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
M(i)	0	1	2	3	1	1	2	3	2						

1+Min(3,1,3) = 2

Finding Which Stamps: Trace-Back

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
M(i)	0	1	2	3	1	1	2	3	2						

4¢

<u>1</u>+Min(3,<u>1</u>,3) = <u>2</u>

Complexity Note

- O(N) is better than O(N^3) or O($3^{N/5}$)
- But still *exponential* in input size (log N bits)

  (E.g., miserably slow if N is 64 bits.)
- Note: can do in O(1) for 5¢, 4¢, and 1¢ but not in general. See "NP-Completeness" later

Elements of Dynamic Programming

- What feature did we use?
- What should we look for to use again?

- "Optimal Substructure"
  Optimal solution contains optimal subproblems
- "Repeated Subproblems"
  The same subproblems arise in various ways

The Knapsack Problem (§ 5.10)

Given positive integers W, w_1, w_2, …, w_n,
Find a subset of the w_i's totaling exactly W.

(Like stamp problem, but limited supply of each.)

Motivation: simple 1-d abstraction of packing boxes, trucks, VLSI chips, …