CSE 421: Introduction to Algorithms

Dynamic Programming

“Dynamic Programming”

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

Dynamic Programming

- Outline:
  - Example 1 – Licking Stamps
  - General Principles
  - Example 2 – Knapsack (§ 5.10)
  - Example 3 – Sequence Comparison (§ 6.8)

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.

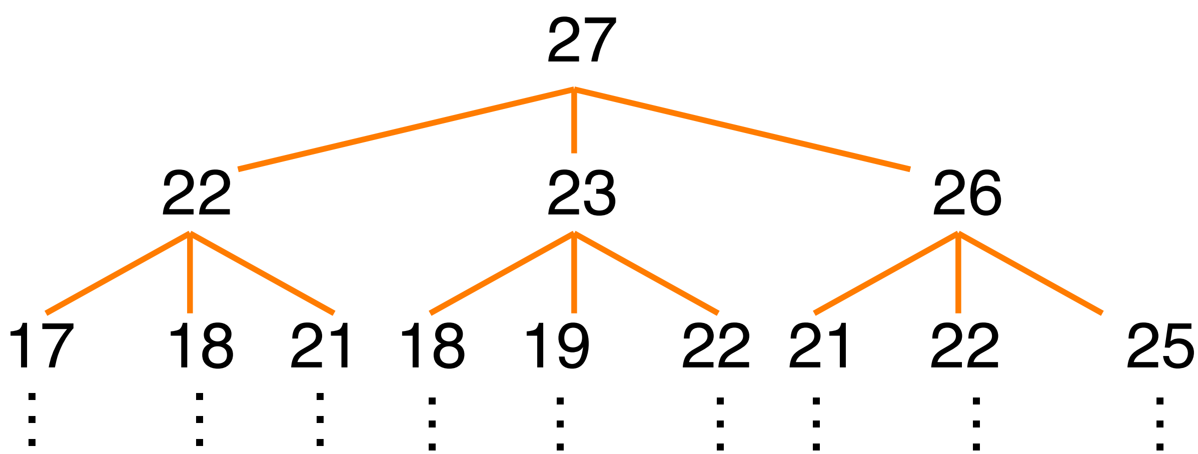
Proof: if not, we could improve the solution for N by using opt for N-v.

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

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

New Idea: Recursion

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



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 – $c \cdot 2^{64}$ steps for 64 bit input.)
- 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
  A non-example: min (number of stamps mod 2)

- "Repeated Subproblems"

  The same subproblems arise in various ways