CSE 421: Introduction to Algorithms

Dynamic Programming

1

Dynamic Programming

- Chapter 16
- Today:
  - Example 1 - Licking Stamps
  - General Principles
  - Example 2 - Matrix-chain products

2

Licking Stamps

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

3

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

4

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…)

5

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.

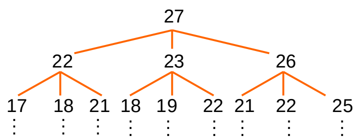
- $$M(i) = \min\begin{Bmatrix} 0 & i=0 \\ 1+M(i-5) & i\geq 5 \\ 1+M(i-4) & i\geq 4 \\ 1+M(i-1) & i\geq 1 \end{Bmatrix}$$

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

6

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: $> 3^{N/5}$

7

Another New Idea: Avoid Recomputation

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

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

8

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

9

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>

10

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.)
- See "NP-Completeness" later

11

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"

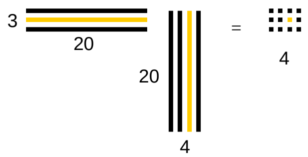
  The same subproblems arise in various ways

12

Matrix-chain Products

~~Strassen~~

- Given: $p_{i-1} \times p_i$ matrices A_i, $1 \le i \le n$
- Problem: Compute $A_1 \bullet A_2 \bullet \ldots \bullet A_n$



Work = 3 * 20 * 4

In general:
p × q times q × r
costs p*q*r

13

Matrix-chain Products

- Given: $p_{i-1} \times p_i$ matrices A_i, $1 \le i \le n$
- Problem: Compute $A_1 \bullet A_2 \bullet \ldots \bullet A_n$

- In What Order?
- Example: A•B•C, where:
  - A is 1 x 10
  - B is 10 x 2
  - C is 2 x 10

(A • B) • C	A • (B • C)
$\underbrace{1*10*2} + \underbrace{1*2*10}$	$\underbrace{1*10*10} + \underbrace{10*2*10}$
40	300

14

Simple Algorithm

- Just try all possible parenthesizations
- How many are there?
  - $P(1) = 1$

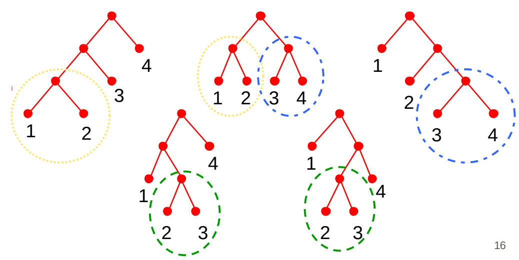
$$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k), n > 1$$

  - $P(n) = \frac{1}{n}\binom{2n-2}{n-1} = \Omega\left(\frac{4^n}{n^{3/2}}\right)$

15

Repeated Subproblems

- All 5 Parenthesizations of $A_1 \bullet A_2 \bullet A_3 \bullet A_4$:



16

Optimal Substructure:

- **Theorem:** if the last multiply is $(A_1 \ldots A_i) \bullet (A_{i+1} \ldots A_n)$, then $A_1 \ldots A_i$ is optimally parenthesized, as is $A_{i+1} \ldots A_n$.
  Proof: Could improve if not.

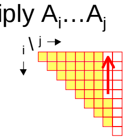
- Let M[i,j] = min ops to multiply $A_i \ldots A_j$

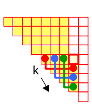
$$M[i,j] = \begin{cases} 0 & i = j \\ \min_{i \le k < j}(M[i,k] + M[k+1,j] + p_{i-1}p_k p_j) & i < j \end{cases}$$

17

An O(n^3) Algorithm

```
// Goal: M[i,j] = min ops to multiply Ai…Aj
for j := 1 to n do
  M[j,j] := 0;
  for i := (j - 1) downto 1 do
    M[i,j] := min_{i≤k<j}(p_{i-1}p_k p_j +
                 M[i,k]+M[k+1,j]);
```





18

Example:

	1	2	3	4
1				
2				
3				
4				

$p_0 = 2$
$p_1 = 3$
$p_2 = 1$
$p_3 = 5$
$p_4 = 1$

A_1: 2x3
A_2: 3x1
A_3: 1x5
A_4: 5x1

19