Algorithms and Data Structures: Dynamic Programming; Matrix-chain multiplication

Algorithmic Paradigms

Divide and Conquer

Idea: Divide problem instance into smaller sub-instances of the same problem, solve these recursively, and then put solutions together to a solution of the given instance.

Examples: Mergesort, Quicksort, Strassen's algorithm, FFT.

Greedy Algorithms

Idea: Find solution by always making the choice that looks optimal at the moment — don't look ahead, never go back.

Examples: Prim's algorithm, Kruskal's algorithm.

Dynamic Programming

Idea: **Turn recursion upside down.**

Example: Floyd-Warshall algorithm for the all pairs shortest path problem.

Dynamic Programming - A Toy Example

Fibonacci Numbers

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \qquad \text{(for } n \geq 2\text{)}. \end{aligned}$$

A recursive algorithm

Algorithm Rec-Fib(n)

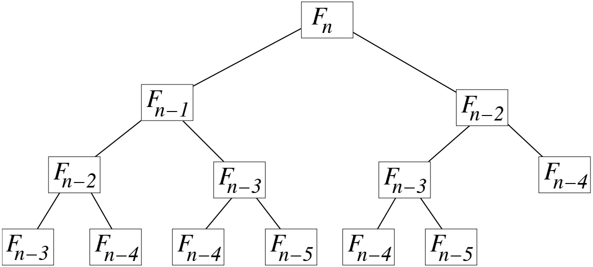
```
1. if $n = 0$ then
2.         return 0
3. else if $n = 1$ then
4.         return 1
5. else
6.         return Rec-Fib($n-1$)+Rec-Fib($n-2$)
```

Ridiculously slow: **exponentially many** repeated computations of Rec-Fib(j) for small values of j.

Fibonacci Example (cont'd)

Why is the recursive solution so slow?
Running time $T(n)$ satisfies

$$T(n) = T(n-1) + T(n-2) + \Theta(1) \geq F_n \sim (1.6)^n.$$



BOARD: We show $F_n \geq \frac{1}{2}(3/2)^n$ for $n \geq 8$.

Fibonacci Example (cont'd)

Dynamic Programming Approach

```
Algorithm Dyn-Fib($n$)
    $F[0] = 0$
    $F[1] = 1$
    for $i \leftarrow 2$ to $n$ do
        $F[i] \leftarrow F[i-1] + F[i-2]$
    return $F[n]$
```

Build "from the bottom up"

Running Time

$$\Theta(n)$$

Very fast in practice - just need an array (of linear size) to store the $\mathrm{F}(i)$ values.

Multiplying Sequences of Matrices

Recall

> Multiplying a $(p \times q)$ matrix with a $(q \times r)$ matrix (in the standard way) requires
>
> $$pqr$$
>
> multiplications.

We want to compute products of the form

$$A_1 \cdot A_2 \cdots A_n.$$

How do we set the parentheses?

Example

Compute

$$\begin{array}{ccccccc} A & \cdot & B & \cdot & C & \cdot & D \\ 30 \times 1 & & 1 \times 40 & & 40 \times 10 & & 10 \times 25 \end{array}$$

Multiplication order $(A \cdot B) \cdot (C \cdot D)$ requires

$$30 \cdot 1 \cdot 40 + 40 \cdot 10 \cdot 25 + 30 \cdot 40 \cdot 25 = 41,200$$

multiplications.
Multiplication order $A \cdot ((B \cdot C) \cdot D)$ requires

$$1 \cdot 40 \cdot 10 + 1 \cdot 10 \cdot 25 + 30 \cdot 1 \cdot 25 = 1,400$$

multiplications.

The Matrix Chain Multiplication Problem

Input:

Sequence of matrices $A_1, \ldots, A_n$, where A_i is a $p_{i-1} \times p_i$-matrix

Output:

Optimal number of multiplications needed to compute $A_1 \cdot A_2 \cdots A_n$, and an optimal parenthesisation to realise this

Running time of algorithms will be measured in terms of n.

Solution "Attempts"

Approach 1: Exhaustive search (CORRECT but SLOW).
Try all possible parenthesisations and compare them. Correct, but extremely slow; running time is $\Omega(3^n)$. UGLY PROOF

Approach 2: Greedy algorithm (INCORRECT).
Always do the cheapest multiplication first. Does not work correctly — sometimes, it returns a parenthesisation that is not optimal:

Example: Consider

$$\begin{array}{ccccc} A_1 & \cdot & A_2 & \cdot & A_3 \\ 3 \times 100 & & 100 \times 2 & & 2 \times 2 \end{array}$$

Solution proposed by greedy algorithm: $A_1 \cdot (A_2 \cdot A_3)$ with $100 \cdot 2 \cdot 2 + 3 \cdot 100 \cdot 2 = 1000$ multiplications.

Optimal solution: $(A_1 \cdot A_2) \cdot A_3$ with $3 \cdot 100 \cdot 2 + 3 \cdot 2 \cdot 2 = 612$ multiplications.

Solution "Attempts" (cont'd)

Approach 3: Alternative greedy algorithm (INCORRECT).
Set outermost parentheses such that cheapest multiplication is done last.

Doesn't work correctly either (Exercise!).

Approach 4: Recursive (Divide and Conquer) - (SLOW - **see over**).
Divide:

$$(A_1 \cdots A_k) \cdot (A_{k+1} \cdots A_n)$$

For all k, recursively solve the two sub-problems and then take best overall solution.

For $1 \leq i \leq j \leq n$, let

$m[i,j] =$ least number of multiplications needed to compute $A_i \cdots A_j$

Then

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \leq k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & \text{if } i < j. \end{cases}$$

The Recursive Algorithm (SLOW)

Running time $T(n)$ satisfies the recurrence

$$T(n) = \sum_{k=1}^{n-1} \big(T(k) + T(n-k)\big) + \Theta(n).$$

This implies

$$T(n) = \Omega(2^n).$$

BOARD

Dynamic Programming Solution

As before:

$$m[i,j] = \text{least number of multiplications needed to compute } A_i \cdots A_j$$

Moreover,

$$s[i,j] = \text{(the smallest) } k \text{ such that } i \leq k < j \text{ and } m[i,j] = m[i,k] + m[k+1,j] + p_{i-1}p_kp_j.$$

$s[i,j]$ can be used to reconstruct the optimal parenthesisation.

Idea

Compute the $m[i,j]$ and $s[i,j]$ in a bottom-up fashion.

TURN RECURSION UPSIDE DOWN :-)

Implementation

Algorithm Matrix-Chain-Order(p)

```
$n \leftarrow p.length - 1$
for $i \leftarrow 1$ to $n$ do
    $m[i,i] \leftarrow 0$
for $\ell \leftarrow 2$ to $n$ do
    for $i \leftarrow 1$ to $n - \ell + 1$ do
        $j \leftarrow i + \ell - 1$
        $m[i,j] \leftarrow \infty$
        for $k \leftarrow i$ to $j - 1$ do
            $q \leftarrow m[i,k] + m[k+1,j] + p_{i-1}p_kp_j$
                if $q < m[i,j]$ then
                    $m[i,j] \leftarrow q$
                    $s[i,j] \leftarrow k$
return $s$
```

Running Time: $\Theta(n^3)$

Example

$$\begin{array}{ccccccc} A_1 & \cdot & A_2 & \cdot & A_3 & \cdot & A_4 \\ 30 \times 1 & & 1 \times 40 & & 40 \times 10 & & 10 \times 25 \end{array}$$

Solution for m and s

m	1	2	3	4
1	0	1200	700	1400
2		0	400	650
3			0	10 000
4				0

s	1	2	3	4
1		1	1	1
2			2	3
3				3
4				

Optimal Parenthesisation

$$A_1 \cdot \big((A_2 \cdot A_3) \cdot A_4\big)$$

Multiplying the Matrices

```
Algorithm Matrix-Chain-Multiply($A, p$)
    $n \leftarrow A.length$
    $s \leftarrow$ Matrix-Chain-Order($p$)
    return Rec-Mult($A, s, 1, n$)
```

```
Algorithm Rec-Mult($A, s, i, j$)
    if $i < j$ then
        $C \leftarrow$ Rec-Mult($A, s, i, s[i, j]$)
        $D \leftarrow$ Rec-Mult($A, s, s[i, j] + 1, j$)
        return $(C) \cdot (D)$
    else
        return $A_i$
```

Problems

see Wikipedia:
`http://en.wikipedia.org/wiki/Dynamic_programming`
[CLRS] Sections 15.2-15.3

1. Review the Edit-Distance Algorithm and try to understand why it is a dynamic programming algorithm.
2. Exercise 15.2-1 of [CLRS].