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

21st October, 2014
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

\[ F_0 = 0, \]
\[ F_1 = 1, \]
\[ F_n = F_{n-1} + F_{n-2} \quad (\text{for } n \geq 2). \]

A recursive algorithm

**Algorithm** \texttt{Rec-Fib}(n)

1. \textbf{if } \texttt{n} = 0 \textbf{ then} \\
2. \hspace{1em} \textbf{return} 0 \\
3. \textbf{else if } \texttt{n} = 1 \textbf{ then} \\
4. \hspace{1em} \textbf{return} 1 \\
5. \textbf{else} \\
6. \hspace{1em} \textbf{return} \texttt{Rec-Fib}(n - 1)+\texttt{Rec-Fib}(n - 2)

Ridiculously slow: \textit{exponentially many} repeated computations of \texttt{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 \( n \geq 8 \).

*ADS: lect 9 – slide 4 – 21st October, 2014*
Fibonacci Example (cont’d)

Dynamic Programming Approach

Algorithm \textsc{Dyn-Fib}(n)

1. \( F[0] = 0 \)
2. \( F[1] = 1 \)
3. \textbf{for} \( i \leftarrow 2 \) \textbf{to} \( n \) \textbf{do}
4. \hspace{1em} \( F[i] \leftarrow F[i - 1] + F[i - 2] \)
5. \textbf{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 \( F(i) \) 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

\[
\begin{array}{cccc}
  A & \cdot & B & \cdot & C & \cdot & D \\
  30 \times 1 & 1 \times 40 & 40 \times 10 & 10 \times 25
\end{array}
\]

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 $n$. 

ADS: lect 9 – slide 8 – 21st October, 2014
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

$$A_1 \cdot A_2 \cdot A_3$$

$$3 \times 100 \quad 100 \times 2 \quad 2 \times 2$$

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.

*ADS: lect 9 – slide 9 – 21st October, 2014*
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

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

Then

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

Running time $T(n)$ satisfies the recurrence

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

This implies

$$T(n) = \Omega(2^n).$$
Dynamic Programming Solution

As before:

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

Moreover,

\[ s[i,j] = (\text{the smallest}) \ k \text{ such that } i \leq k < j \text{ and } \]
\[ m[i,j] = m[i,k] + m[k+1,j] + p_i p_k p_j. \]

\( s[i,j] \) 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 :-)

ADS: lect 9 – slide 12 – 21st October, 2014
Implementation

**Algorithm** Matrix-Chain-Order\((p)\)

1. \(n \leftarrow p.length - 1\)
2. **for** \(i \leftarrow 1\) **to** \(n\) **do**
3. \(m[i, i] \leftarrow 0\)
4. **for** \(\ell \leftarrow 2\) **to** \(n\) **do**
5. **for** \(i \leftarrow 1\) **to** \(n - \ell + 1\) **do**
6. \(j \leftarrow i + \ell - 1\)
7. \(m[i, j] \leftarrow \infty\)
8. **for** \(k \leftarrow i\) **to** \(j - 1\) **do**
9. \(q \leftarrow m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\)
10. **if** \(q < m[i, j]\) **then**
11. \(m[i, j] \leftarrow q\)
12. \(s[i, j] \leftarrow k\)
13. **return** \(s\)

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

ADS: lect 9 – slide 13 – 21st October, 2014
Example

\[ A_1 \cdot A_2 \cdot A_3 \cdot A_4 \]
\[ 30 \times 1 \quad 1 \times 40 \quad 40 \times 10 \quad 10 \times 25 \]

Solution for \( m \) and \( s \)

\[
\begin{array}{c|cccc}
   m & 1 & 2 & 3 & 4 \\
\hline
   1 & 0 & 1200 & 700 & 1400 \\
   2 & 0 & 400 & 650 & \\
   3 & & 0 & 10000 & \\
   4 & & & 0 & \\
\end{array}
\]

\[
\begin{array}{c|cccc}
   s & 1 & 2 & 3 & 4 \\
\hline
   1 & 1 & 1 & 1 \\
   2 & 2 & 3 \\
   3 & & 3 \\
   4 & & 4 \\
\end{array}
\]

Optimal Parenthesisation

\[ A_1 \cdot ((A_2 \cdot A_3) \cdot A_4) \]
Multiplying the Matrices

Algorithm $\text{Matrix-Chain-Multiply}(A, p)$
1. $n \leftarrow A.\text{length}$
2. $s \leftarrow \text{Matrix-Chain-Order}(p)$
3. return $\text{Rec-Mult}(A, s, 1, n)$

Algorithm $\text{Rec-Mult}(A, s, i, j)$
1. if $i < j$ then
2. $C \leftarrow \text{Rec-Mult}(A, s, i, s[i, j])$
3. $D \leftarrow \text{Rec-Mult}(A, s, s[i, j] + 1, j)$
4. return $(C) \cdot (D)$
5. else
6. return $A_i$
Problems

see Wikipedia:
[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].