Australian National University

COMP3600/6466 Algorithms
Lecture 11

S2 2016

Dr. Hassan Hijazi
Prof. Weifa Liang

Dynamic Programming: Summary

1. Characterise the optimal substructure or show that none exists

2. Recursively define the value of an optimal solution

3. Compute the value of an optimal solution

   Use memoization or bottom-up DP!

4. Construct an optimal solution from computed information

Problem statement

Given the chain $A_1 \cdot A_2 \ldots \cdot A_n$, fully parenthesise/split it in a way that minimises the number of scalar multiplications.

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

The size of matrix A_i will be denoted $p_{i-1} \times p_i$

Brute force enumeration

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

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \, , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 \, . \end{cases}$$

The number of possible solutions is $\Omega(2^n)$

Complete the proof by hand

Dynamic Programming: How To

1. Characterise the optimal substructure or show that none exists

Let $A_{i\ldots j} = A_i \cdot A_{i+1} \ldots \cdot A_j$

Example: $A_{234} = A_2 \cdot A_3 \cdot A_4$.

Any submatrix $A_{i\ldots j}$ is the result of a product of two submatrices splitted at position k, that is $A_{i\ldots j} = A_{i\ldots k} \cdot A_{k+1\ldots j}$

The problem has an optimal substructure:
If k is the optimal split for $A_{i\ldots j}$ then both $A_{i\ldots k}$ and $A_{k+1\ldots j}$ are the result of an optimal splitting of each sub-chain.

Optimal Substructure Proof: How To

Use a proof by contradiction:
1. Assume that the solution is composed of a non-optimal sub-solution
2. Show that this contradicts the optimality assumption of the global solution
(Hint: Use an optimal sub-solution to improve the overall solution)

The Cut and Paste method

The problem has an optimal substructure:
If k is the optimal split for $A_{i\ldots j}$ then both $A_{i\ldots k}$ and $A_{k+1\ldots j}$ are the result of an optimal splitting of each sub-chain.

Proof.

1. Assume that the splitting of the left subchain does not lead to an optimal number of multiplication
2. If I replace this splitting with an optimal splitting, I can improve the cost of the overall solution, which contradicts the optimality assumption on the global chain.

Dynamic Programming: How To

2. Recursively define the value of an optimal solution

Let $m[i, j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i \ldots j}$.

We are thus looking for $m[1, n]$

For a given split after matrix k, based on the optimal substructure, we have:

$$m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$$

Dynamic Programming: How To

2. Recursively define the value of an optimal solution

Let $m[i, j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i \dots j}$.

We are thus looking for $m[1, n]$

For a given split after matrix k, based on the optimal substructure, we have:

$$m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$$

k is not known in advance, can be any value in $i, \dots j - 1$

Dynamic Programming: How To

2. Recursively define the value of an optimal solution

Find the k that minimises the total number of multiplications

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

Dynamic Programming: How To

3. Compute the value of an optimal solution

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
if i == j
    return 0
m[i, j] = ∞
for k = i to j − 1
    q = RECURSIVE-MATRIX-CHAIN(p, i, k)
        + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
        + p_{i−1} p_k p_j
    if q < m[i, j]
        m[i, j] = q
return m[i, j]
```

Exercise 11.1

Express the running time of RECURSIVE-MATRIX-CHAIN as a recursive function

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
if i == j
    return 0
m[i, j] = ∞
for k = i to j − 1
    q = RECURSIVE-MATRIX-CHAIN(p, i, k)
        + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
        + p_{i−1} p_k p_j
    if q < m[i, j]
        m[i, j] = q
return m[i, j]
```

$$
\begin{aligned}
T(1) &\geq 1 , \\
T(n) &\geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \qquad \text{for } n > 1 .
\end{aligned}
$$

Exercise 11.2

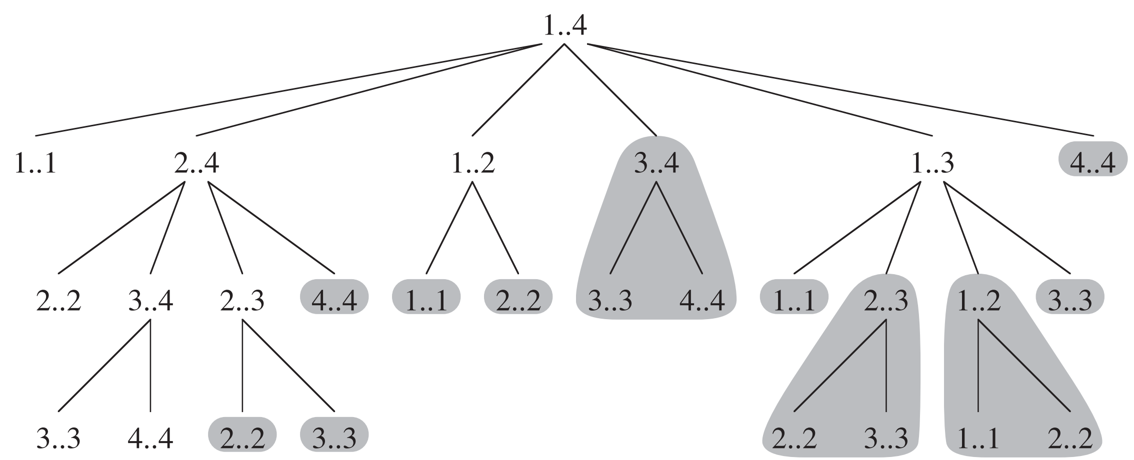
$$\begin{aligned} T(1) &\geq 1 \,, \\ T(n) &\geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \qquad \text{for } n > 1 \,. \end{aligned}$$

Show that $T(n) = \Omega(2^n)$

Complete the proof by hand

Dynamic Programming: How To

3. Compute the value of an optimal solution



How many unique subproblems?

$$n(i..i) + \binom{n}{2}(i..j) = n + \frac{n!}{(n-2)!\ 2!} = n + \frac{n(n-1)}{2} = \frac{n^2+n}{2} = \frac{n(n+1)}{2} = \sum_{k=1}^{n} k$$

Memoized Dynamic Programming

3. Compute the value of an optimal solution

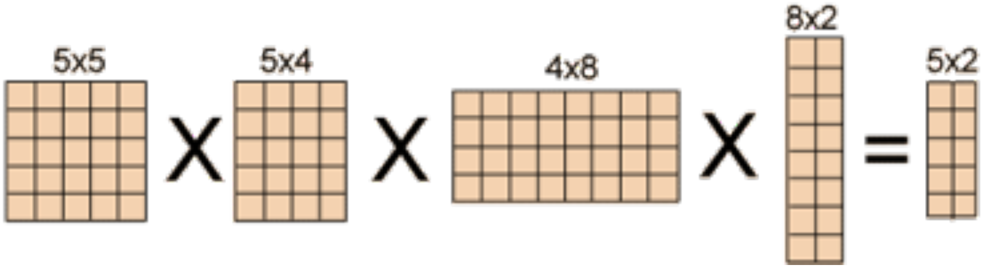
```
MEMOIZED-MATRIX-CHAIN(p)
n = p.length − 1
let m[1..n, 1..n] be a new table
for i = 1 to n
    for j = i to n
        m[i, j] = ∞
return LOOKUP-CHAIN(m, p, 1, n)
```

```
LOOKUP-CHAIN(m, p, i, j)
if m[i, j] < ∞
    return m[i, j]
if i == j
    m[i, j] = 0
else for k = i to j − 1
        q = LOOKUP-CHAIN(m, p, i, k)
              + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
        if q < m[i, j]
            m[i, j] = q
return m[i, j]
```

Australian National University

Bottom-Up in Action

Derive a bottom up algorithm for this instance:



Iterative Dynamic Programming

3. Compute the value of an optimal solution

```
MATRIX-CHAIN-ORDER(p)
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

Iterative Dynamic Programming

3. Compute the value of an optimal solution

MATRIX-CHAIN-ORDER(p)

```
n = p.length − 1
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                          // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
            if q < m[i, j]
                m[i, j] = q
                s[i, j] = k
return m and s
```

number of subproblems per level

k is the split index

Computing the min cost solution

storing the optimal cost and the optimal split

Chain of length n

Chain of length 2

Chain of length 1

Exercise 11.3

Give an asymptotic upper bound on the running time of MATRIX-CHAIN-ORDER

```
MATRIX-CHAIN-ORDER(p)
n = p.length - 1
let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
for i = 1 to n
    m[i,i] = 0
for l = 2 to n                // l is the chain length
    for i = 1 to n - l + 1
        j = i + l - 1
        m[i,j] = ∞
        for k = i to j - 1
            q = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j
            if q < m[i,j]
                m[i,j] = q
                s[i,j] = k
return m and s
```

1

$n^2 + (n-1)^2$

$n+1$

n

$n \times t(l) + 1$

$t(l) \leq (n-1) \times t(i) + 1$

$t(i) \leq 3 + n \times t(k)$

$t(k) = c$

$$T(n) \leq \Theta(n^2) + n((n-1)(3+nc)+1) = \Theta(n^3)$$

$$\implies T(n) = O(n^3)$$

Iterative Dynamic Programming

4. Construct an optimal solution from computed information

```
PRINT-OPTIMAL-PARENS(s, i, j)
if i == j
    print "A"_i
else print "("
    PRINT-OPTIMAL-PARENS(s, i, s[i, j])
    PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
    print ")"
```