Design and Analysis of Algorithms CS218M

Dynamic Programming

Paritosh Pandya

Indian Institute of Technology, Bombay

Autumn, 2022

Rod Cutting

Given a rod of length n and price table

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

$p[2]$

- $p[i]$ is the cost of rod piece of length i.
- Objective: Cut rod of length n into k pieces of lengths $i_1, i_2, \ldots, i_k$ such that

$$n = i_1 + i_2 + \ldots + i_k,$$

$$\text{total price } p[i_1] + p[i_2] + \ldots + p[i_k] \text{ is maximized.}$$

This maximal price is denoted r_n.

r_3

3 $\rightarrow$ 8

2, 1 $\rightarrow$ 6

1, 1, 1 $\rightarrow$ 3

$\langle 3 \rangle$

$r_4 \rightarrow 10$

$\langle 2, 2 \rangle$

Rod Cutting

Given a rod of length n and price table

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- $p[i]$ is the cost of rod piece of length i.
- Objective: Cut rod of length n into k pieces of lengths $i_1, i_2, \ldots, i_k$ such that

  $n = i_1 + i_2 + \ldots + i_k$,

  total price $p[i_1] + p[i_2] + \ldots + p[i_k]$ is maximized.

  This maximal price is denoted r_n.

Example: $r_4 = 10$ due to two pieces of size $2, 2$.

$r_n \quad \max_{i=0}^{n-1} \; r_i + r_{n-i}$

$r_1 = p[1] \qquad r_0 = 0$

Rod Cutting

Given a rod of length n and price table

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- $p[i]$ is the cost of rod piece of length i.
- Objective: Cut rod of length n into k pieces of lengths $i_1, i_2, \ldots, i_k$ such that

  $n = i_1 + i_2 + \ldots + i_k$,

  total price $p[i_1] + p[i_2] + \ldots + p[i_k]$ is maximized.

  This maximal price is denoted r_n.

Example: $r_4 = 10$ due to two pieces of size $2, 2$.

Optimal Substructure

Let r_n denote weight of the optimal solution for rod of length n.
Then, $r_0 = 0$, and

$$r_n = max_{1 \leq i \leq n} \; (p[i] + r_{n-i})$$

Topdown Recursive Solution

```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

n

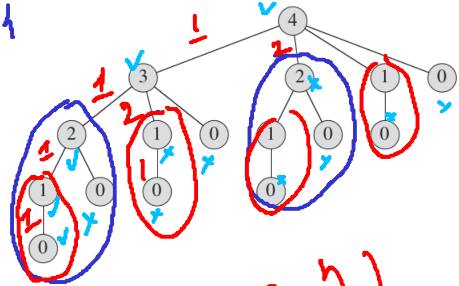
Topdown Recursive Solution

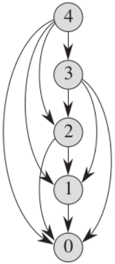
```
CUT-ROD(p, n)
if n == 0
    return 0
q = −∞
for i = 1 to n
    q = max(q, p[i] + CUT-ROD(p, n − i))
return q
```

Recursion Tree and Subproblem Graph

$$r_n = max_{1 \leq i \leq n} \; (p[i] + r_{n-i})$$

$n = 4$





$\Omega(2^n)$

$C(n,k) = C(n-1,k) + C(n-1,k-1)$

Topdown Memoized Recursive Procedure

- Estimate number of distinct recursive calls possible.
- A table $r[0..n]$ stores result for call with parameter n.
- On subsequent call with same parameter, the result is returned from the table.

```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = -∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

Topdown Memoized Recursive

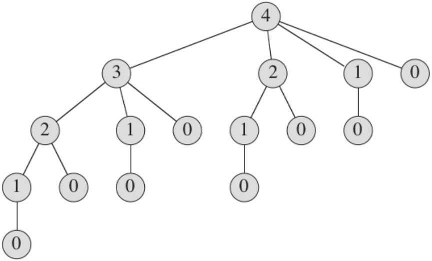
MEMOIZED-CUT-ROD-AUX(p, n, r)

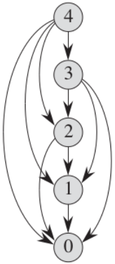
```
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
r[n] = q
return q
```

$\Theta(n^2)$

Recursion Tree and Subproblem Graph

$$r_n = max_{1 \le i \le n} \; (p[i] + r_{n-i})$$





Bottom Up Procedure

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max(q, p[i] + r[j - i])
    r[j] = q
return r[n]
```

$\Theta(n)$

$\sum_{i=1}^{n} i$

$\Theta(n^2)$

Reconstructing Solution

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        if q < p[i] + r[j - i]
            q = p[i] + r[j - i]
            s[j] = i
    r[j] = q
return r and s
```

Reconstructing Solution

```
PRINT-CUT-ROD-SOLUTION(p, n)
(r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
while n > 0
    print s[n]
    n = n - s[n]
```

In our rod-cutting example, the call EXTENDED-BOTTOM-UP-CUT-ROD$(p, 10)$ would return the following arrays:

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

2, 2, 0

Dynamic Programming Outline

- Characterize the optimal substructure of the problem.
- Characterize the substructure recursively.
- Embody recursive formulation of optimal choice in memoized topdown or bottomup computation.
- Output optimal solution from computed information.

Matrix Chain Multiplication

- multiplying $A_1 A_2$ of dimensions $p \times q$ and $q \times r$ requires pqr scaler multiplications.

$C[2,j] = C_{2,j}$

Matrix Chain Multiplication

- multiplying $A_1 A_2$ of dimensions $p \times q$ and $q \times r$ requires pqr scaler multiplications.
- Consider chain $A_1 A_2 A_3$ of matrices with dimensions 10×100, 100×5 and 5×50 to be multiplied.

Matrix Chain Multiplication

- multiplying $A_1 A_2$ of dimensions $p \times q$ and $q \times r$ requires pqr scaler multiplications.
- Consider chain $A_1 A_2 A_3$ of matrices with dimensions 10×100, 100×5 and 5×50 to be multiplied.
- Full parenthesis $((A_1 A_2) A_3)$ requires $10 \times 100 \times 5 \; + \; 10 \times 5 \times 50$ scaler multiplication, i.e. $5000 + 2500$.   7500

Matrix Chain Multiplication

- multiplying A_1A_2 of dimensions $p \times q$ and $q \times r$ requires pqr scaler multiplications.
- Consider chain $A_1A_2A_3$ of matrices with dimensions 10×100, 100×5 and 5×50 to be multiplied.
- Full parenthesis $((A_1A_2)A_3)$ requires $10 \times 100 \times 5 \ + \ 10 \times 5 \times 50$ scaler multiplication, i.e. $5000 + 2500$. = 7500
- $(A_1(A_2A_3))$ requires $100 \times 5 \times 50 \ + \ 10 \times 100 \times 50$ scaler multiplication, i.e. $25000 + 50000$. = 75000

- multiplying A_1A_2 of dimensions $p \times q$ and $q \times r$ requires pqr scaler multiplications.
- Consider chain $A_1A_2A_3$ of matrices with dimensions 10×100, 100×5 and 5×50 to be multiplied.
- Full parenthesis $((A_1A_2)A_3)$ requires $10 \times 100 \times 5 \ + \ 10 \times 5 \times 50$ scaler multiplication, i.e. $5000 + 2500$.
- $(A_1(A_2A_3))$ requires $100 \times 5 \times 50 \ + \ 10 \times 100 \times 50$ scaler multiplication, i.e. $25000 + 50000$.
- Given chain $A_1A_2 \ldots A_n$ of n matrices, the number of full parentheses is $\Omega(2^n)$.

Matrix Chain Multiplication

- multiplying A_1A_2 of dimensions $p \times q$ and $q \times r$ requires pqr scaler multiplications.
- Consider chain $A_1A_2A_3$ of matrices with dimensions 10×100, 100×5 and 5×50 to be multiplied.
- Full parenthesis $((A_1A_2)A_3)$ requires $10 \times 100 \times 5 \ + \ 10 \times 5 \times 50$ scaler multiplication, i.e. $5000 + 2500$.
- $(A_1(A_2A_3))$ requires $100 \times 5 \times 50 \ + \ 10 \times 100 \times 50$ scaler multiplication, i.e. $25000 + 50000$.
- Given chain $A_1A_2 \ldots A_n$ of n matrices, the number of full parentheses is $\Omega(2^n)$.

Matrix Chain Multiplication Problem

Given a chain $A_1A_2 \ldots A_n$ of n matrices to be multiplied, where A_i has dimension $p_{i-1} \times p_i$, find a full parenthesis of the chain which requires least number scaler multiplications.

Development

- Let $A_{i..j} = A_i \cdot A_{i+1} \ldots A_j$.
- Let $m[i,j]$ denote the least number of scaler multiplications needed to compute $A_{i..j}$.
- Discovering sub-structure: If we parenthesize $A_{i..j}$ optimally $(A_i \ldots A_k)(A_{k+1} \ldots A_j)$ for $i \leq k < j$ then
$$m[i,j] \quad = \quad m[i,k] \; + \; m[k+1,j] \; + \; p_{i-1} \cdot p_k \cdot p_j.$$

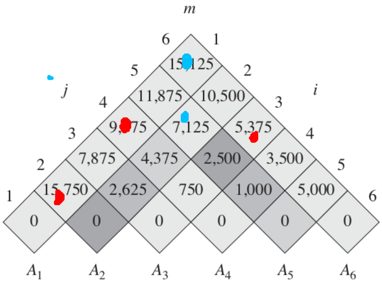
Recursive Substructure

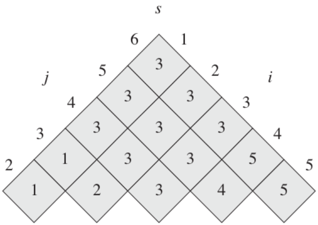
$$m[i,j] = \begin{cases} 0 & \text{if } i = j \ , \\ \min\limits_{i \leq k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j \ . \end{cases}$$

Subproblem Graph

$A_1 \ldots A_6$

$(i:j)$





m[2, 2]

Bottom Up Procedure

$m[2,j]$ $(j-i+1)$

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
```

$\Theta(n)$

$\Theta(n)$

$\Theta(n^3)$

Printing Solution

```
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 ")"
```

Topdown Memoized Solution

```
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]
```