Ch 15. Dynamic Programming

Dynamic Programming

- 표(table)를 만들어 채워가면서 답을 구하는 방법
- Divide and Conquer 와의 차이점 : overlaps in subproblems
- meaning of “programming” here : tabular method
- used in solving optimization problem
  - find an optimal solution, as opposed to the optimal solution

Dynamic Programming

1. 최적해의 구조적 특징을 찾는다.
2. 최적해의 값을 재귀적으로 정의한다.
3. 최적해의 값을 일반적으로 상향식 방법으로 계산한다.
4. 계산된 정보들로부터 최적해를 구성한다.

examples of dynamic programming

15.1 rod cutting

15.2 matrix-chain multiplication

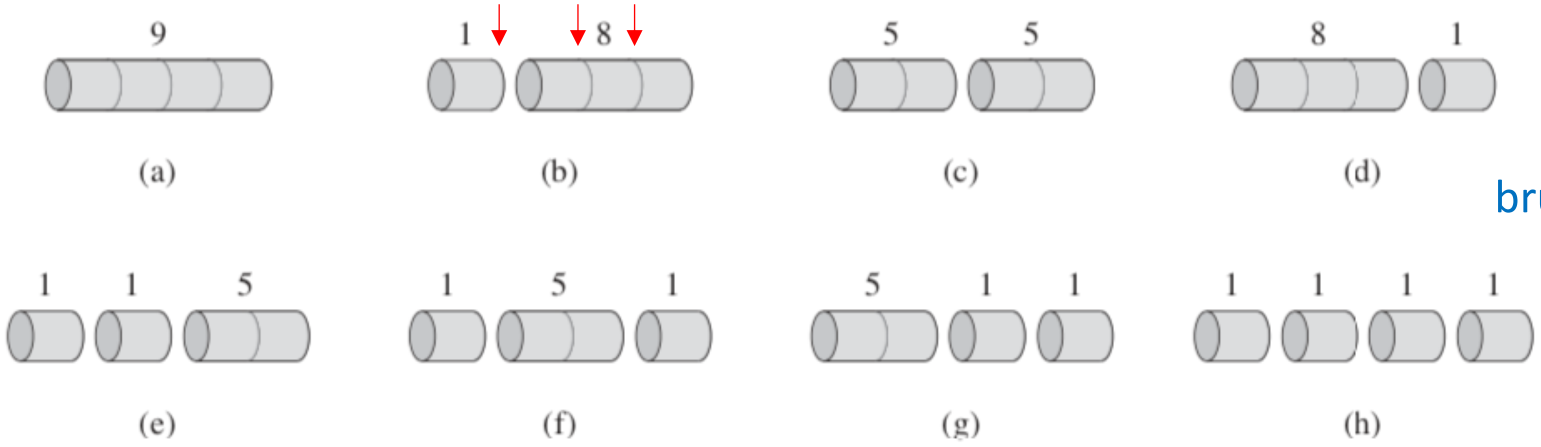
15.4 longest common subsequence

15.1 : rod cutting

- n 인치 막대를 잘라서 판매하여 얻을 수 있는 최대 수익 r_n 을 찾아라.
- 막대를 자르는 비용은 0
- sample 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

- 예를 들어, 4 inch rod 를 자르는 방법은 8=2^3 가지가 있다.



brute-force algorithm : $\Theta(2^n)$

n-inch rod cutting

- 자르는 방법은 2^{n-1} 가지가 있다. → brute-force algorithm : $\Theta(2^n)$
- 7 = 2+2+3 로 자르면 수익은 $r_7 = 5 + 5 + 8 = 18$

$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \text{ (no cuts)}, \\
r_2 &= 5 && \text{from solution } 2 = 2 \text{ (no cuts)}, \\
r_3 &= 8 && \text{from solution } 3 = 3 \text{ (no cuts)}, \\
r_4 &= 10 && \text{from solution } 4 = 2 + 2, \\
r_5 &= 13 && \text{from solution } 5 = 2 + 3, \\
r_6 &= 17 && \text{from solution } 6 = 6 \text{ (no cuts)}, \\
r_7 &= 18 && \text{from solution } 7 = 1 + 6 \text{ or } 7 = 2 + 2 + 3, \\
r_8 &= 22 && \text{from solution } 8 = 2 + 6, \\
r_9 &= 25 && \text{from solution } 9 = 3 + 6, \\
r_{10} &= 30 && \text{from solution } 10 = 10 \text{ (no cuts)}.
\end{aligned}
$$

best among 1+1 and $\boxed{2}$

best among 1+1+1, 1+2 and $\boxed{3}$

best among 1+1+1+1, 1+1+2, 1+3, $\boxed{2+2}$, 4

r_n 은 $r_{n-1}, r_{n-2} \ldots r_1$ 로부터 구할 수 있다.

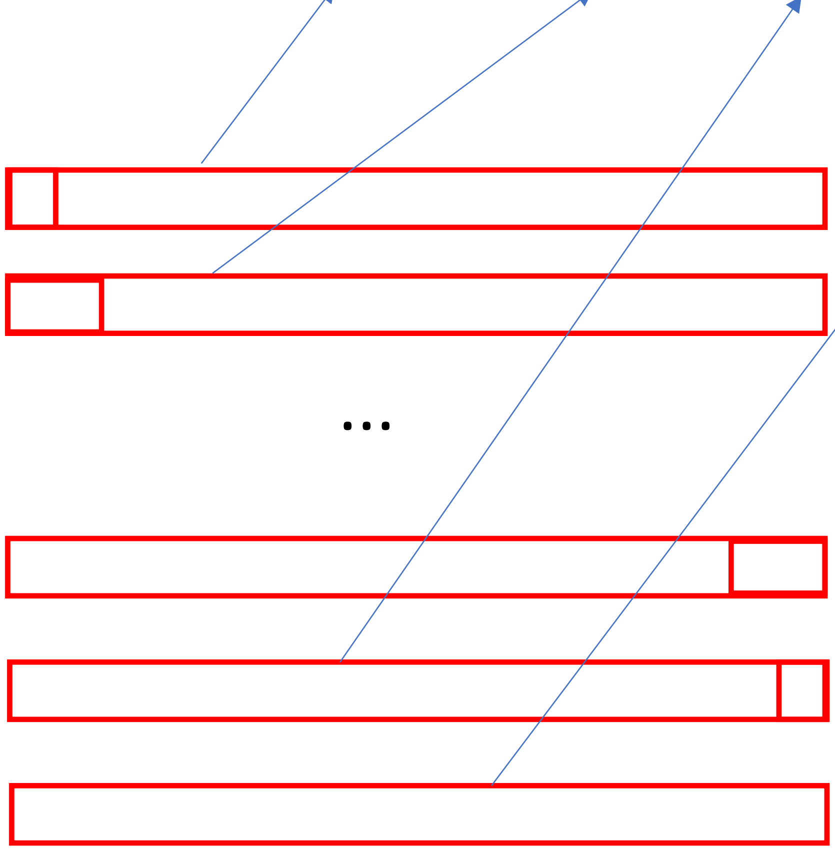
$n = i_1 + i_2 + \cdots + i_k$ 일 때 $r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$

p_j 는

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

r_i for $i<n$ 으로부터 r_n 을 구할 수 있다.
→ optimal substructure 를 가졌다.

$r_n = \max(p_1 + r_{n-1}, p_2 + r_{n-2}, \ldots, p_{n-1} + r_1, p_n)$ => $r_n = \max_{1 \le i \le n} (p_i + r_{n-i})$



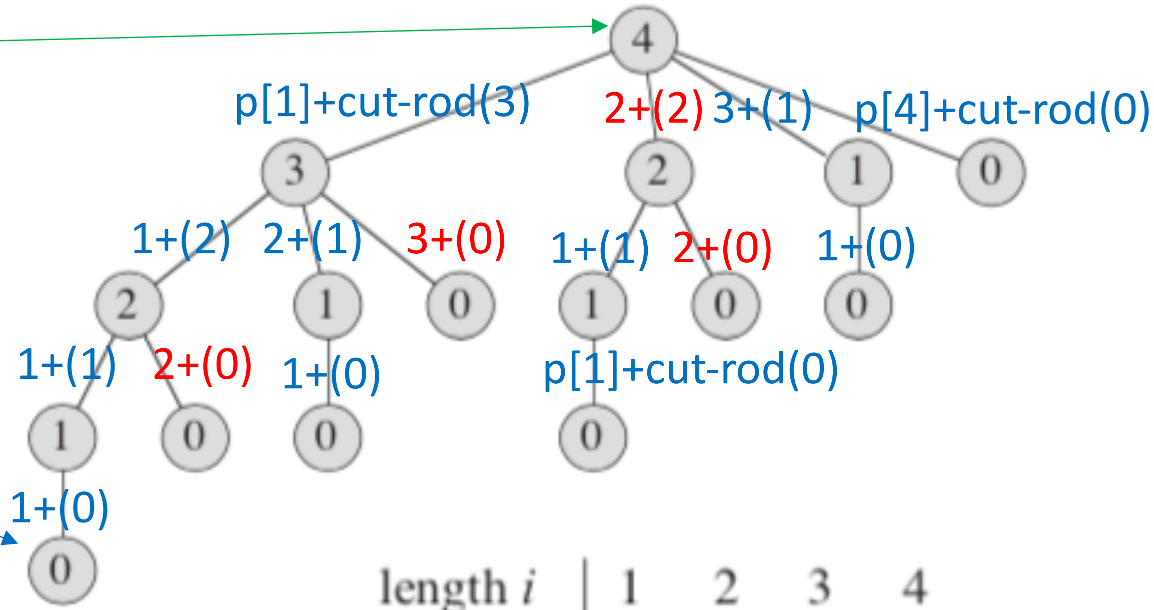
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Recursive top-down implementation

```
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
```

자르지 않은 길이 i의 막대 가격

나머지 부분의 최적 분할 가격



length i	1	2	3	4
price p_i	1	5	8	9

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) .$$

→ $T(n) = 2^n$ (exercise 15.1-1)

Dynamic Programming – top-down $\Theta(n^2)$

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)
```

r

-1	-1	-1	-1	-1

length i	1	2	3	4
price p_i	1	5	8	9

r

0	1	5	8	10
	1+0	2+0	3+0	2+2

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
```

vs.

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
```

Dynamic Programming – top-down $\Theta(n^2)$

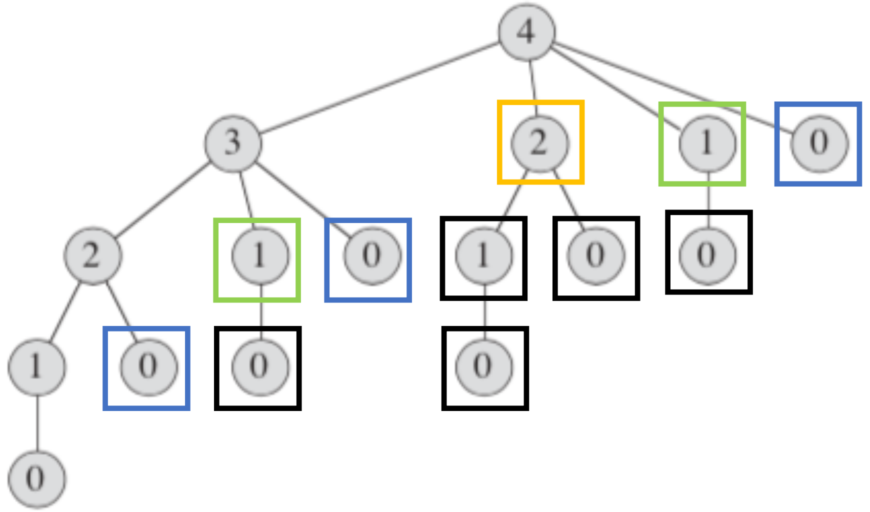
```
MEMOIZED-CUT-ROD(p, n)
    let r[0..n] be a new array
    for i = 0 to n
        r[i] = −∞
    return MEMOIZED-CUT-ROD-AUX(p, n, r)

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
```

r	-1	-1	-1	-1	-1

length i	1	2	3	4
price p_i	1	5	8	9

r	0	1	5	8	10

1+0 2+0 3+0 2+2



Dynamic Programming – bottom-up

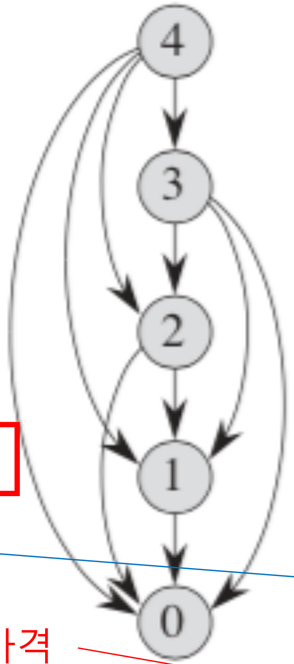
BOTTOM-UP-CUT-ROD(p, n)

```
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]
```

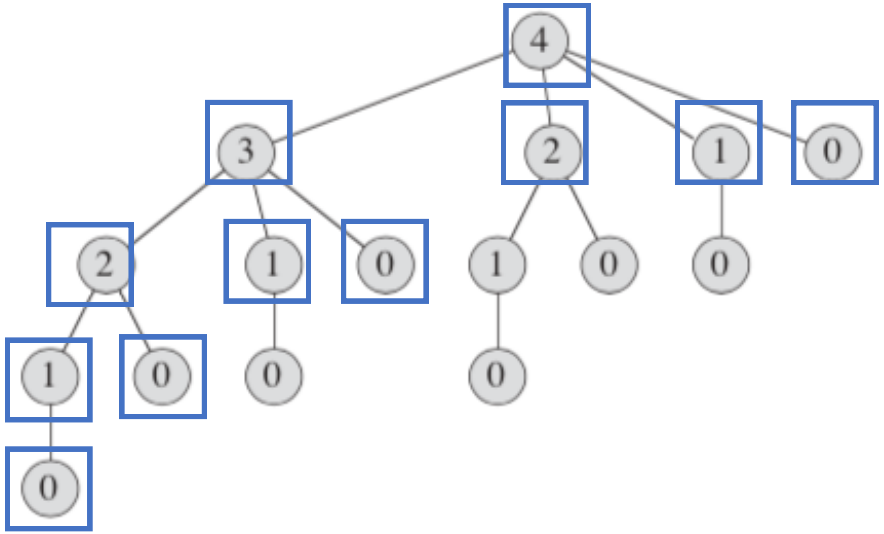
자르지 않은 길이 i의 막대 가격

나머지 부분의 최적 분할 가격

$$\sum_{j=1}^{n} j = \Theta(n^2)$$



vs.



subproblem graph G=(V,E)

Time Complexity of the algorithm = O(|v|+|E|)

length i	1	2	3	4
price p_i	1	5	8	9

r	0	1	5	8	10
		1+0	1+1	1+2	1+3
			2+0	2+1	2+2
				3+0	3+1
					4+0

j

i

Reconstructing a solution : 어떻게 잘라야 하나?

```
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
```

자르지 않은 길이 i의 막대 가격

나머지 부분의 최적 분할 가격

```
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]
```

$\Theta(n^2)$

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

p[1]+r[0]

p[1]+r[1] = 1+1
p[2]+r[0] = 5+0

p[1]+r[3]=1+8
p[2]+r[2]=5+5
p[3]+r[1]=8+1
p[4]+r[0]=9+0

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

n-inch rod cutting

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

- 자르는 방법은 2^{n-1} 가지가 있다.
- 7 = 2+2+3 로 자르면 수익은 $r_7 = 5 + 5 + 8 = 18$

$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)}, \\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)}, \\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)}, \\
r_4 &= 10 && \text{from solution } 4 = 2 + 2, \\
r_5 &= 13 && \text{from solution } 5 = 2 + 3, \\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)}, \\
r_7 &= 18 && \text{from solution } 7 = \boxed{1 + 6} \text{ or } 7 = \boxed{2 + 2 + 3}, \\
r_8 &= 22 && \text{from solution } 8 = 2 + 6, \\
r_9 &= 25 && \text{from solution } 9 = 3 + 6, \\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}.
\end{aligned}
$$

best among 1+1 and 2

best among 1+1+1, 1+2 and 3

best among 1+1+1+1, 1+1+2, 1+3, 2+2, 4

$n = i_1 + i_2 + \cdots + i_k$ 일 때 $r_n = p_{i_1} + p_{i_2} + \cdots + p_{i_k}$

p_j 는

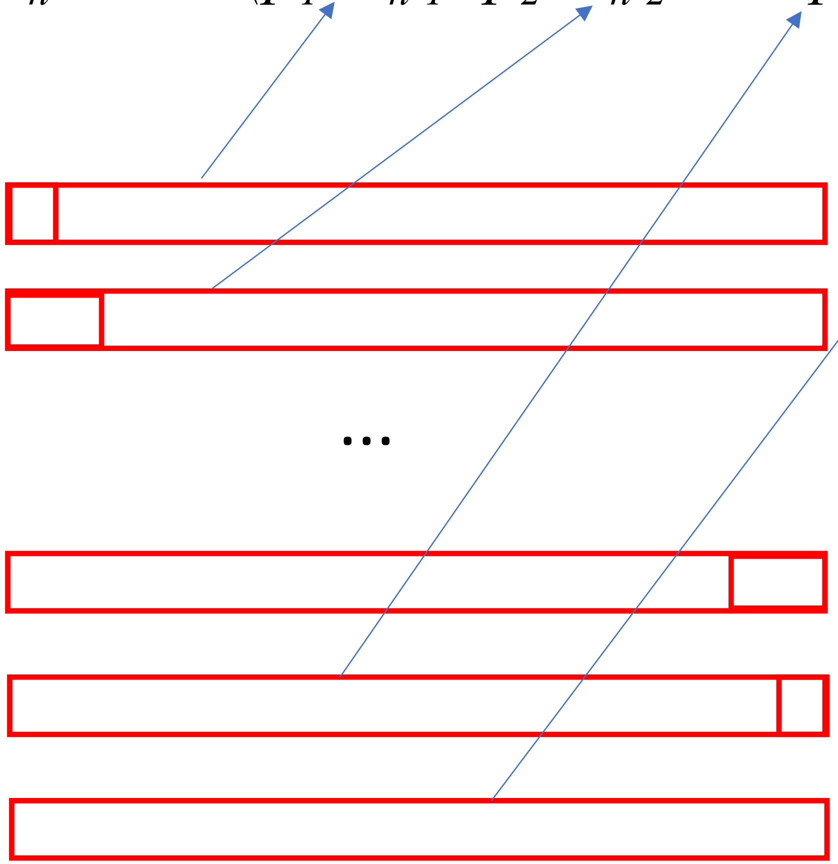
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Dynamic Programming

- 표(table)를 만들어 채워가면서 답을 구하는 방법
- Divide and Conquer 와의 차이점 : overlaps in subproblems
- meaning of “programming” here : tabular method
- used in solving optimization problem
  - find an optimal solution, as opposed to the optimal solution

r_i for $i<n$ 으로부터 r_n 을 구할 수 있다.
→ optimal substructure 를 가졌다.

$$r_n = \max\,(p_1+r_{n-1},\ p_2+r_{n-2},\ \dots,\ p_{n-1}+r_1,\ p_n) \quad \Rightarrow \quad r_n = \max_{1\le i\le n}(p_i + r_{n-i})$$



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

행렬 곱셈 $C = AB$

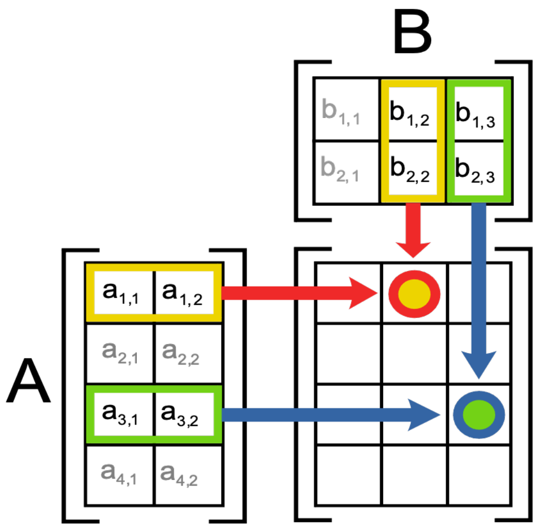
MATRIX-MULTIPLY(A, B)

```
if A.columns ≠ B.rows
    error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
    for i = 1 to A.rows
        for j = 1 to B.columns
            c_ij = 0
            for k = 1 to A.columns
                c_ij = c_ij + a_ik · b_kj
    return C
```

$=\Theta(A.rows \times B.columns \times A.columns)$

행렬 곱셈의 교환법칙은 성립하지 않음 $AB \neq BA$

행렬 곱셈의 결합법칙은 성립 $A(BC) = (AB)C$



A B A * B

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \begin{pmatrix} 6 & 3 \\ 5 & 2 \\ 4 & 1 \end{pmatrix} = \begin{pmatrix} 1*6+2*5+3*4 & 1*3+2*2+3*1 \\ 4*6+5*5+6*4 & 4*3+5*2+6*1 \end{pmatrix}$$

15.2 Matrix-chain multiplication

- 여러 개의 행렬을 곱할 때 곱셈 순서에 따라 연산 갯수가 달라진다.

MATRIX-MULTIPLY(A, B)

```
if A.columns ≠ B.rows
      error "incompatible dimensions"
else let C be a new A.rows × B.columns matrix
      for i = 1 to A.rows
            for j = 1 to B.columns
                  c_ij = 0
                  for k = 1 to A.columns
                        c_ij = c_ij + a_ik · b_kj
      return C
```

$A_1 : 2\times3 \quad A_2 : 3\times5 \quad A_3 : 5\times6 \rightarrow A_1A_2A_3 : 2\times6$

$(A_1 A_2) A_3 : 2\times3\times5 + 2\times5\times6 = 90$

$A_1 (A_2 A_3) : 3\times5\times6 + 2\times3\times6 = 126$

행렬 곱셈의 순서를 정하는 문제 (곱셈을 하는 게 아님)

$$(A_1(A_2(A_3A_4)))$$
$$(A_1((A_2A_3)A_4))$$
$$((A_1A_2)(A_3A_4))$$
$$((A_1(A_2A_3))A_4)$$
$$(((A_1A_2)A_3)A_4)$$

중에서 어떤 순서로 연산하는 것이 scalar 곱셈 횟수가 최소화될까?

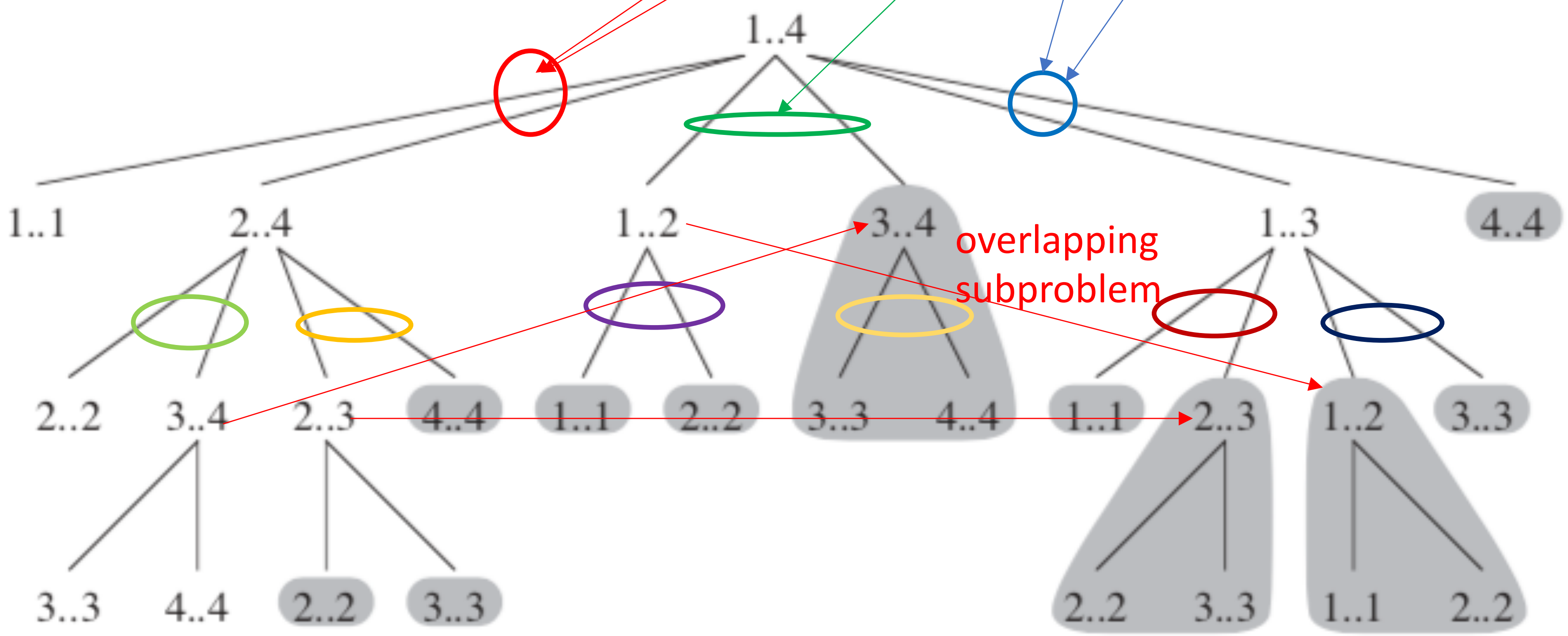
brute-force approach

- Exhaustive search when n = 4

$P(n)$: n 개의 행렬을 괄호로 묶는 서로 다른 방법의 수

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

$(A_1(A_2(A_3A_4)))$
$(A_1((A_2A_3)A_4))$
$((A_1A_2)(A_3A_4))$
$((A_1(A_2A_3))A_4)$
$(((A_1A_2)A_3)A_4)$



Dynamic Programming for solving matrix chain multiplication

1. 최적해의 구조적 특징을 찾는다.
2. 최적해의 값을 재귀적으로 정의한다.

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

$m[i,j]$: A_i x… x A_j 의 곱을 optimal 순서로 곱했을 때 연산의 횟수

p_k : A_k 의 column 의 갯수 = A_{k+1} 의 row 의 갯수

단 p_0는 A_1의 row의 갯수

따라서→ A_k의 크기는 p_{k-1} x p_k

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

→

P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

3. 최적해의 값을 일반적으로 상향식 방법으로 계산한다.

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

을 recursive call 로 구현하면 $\Omega(2^n)$

- optimal substructure 를 가지고 subproblem 들이 overlapped 되어있다.

→ dynamic programming 의 조건

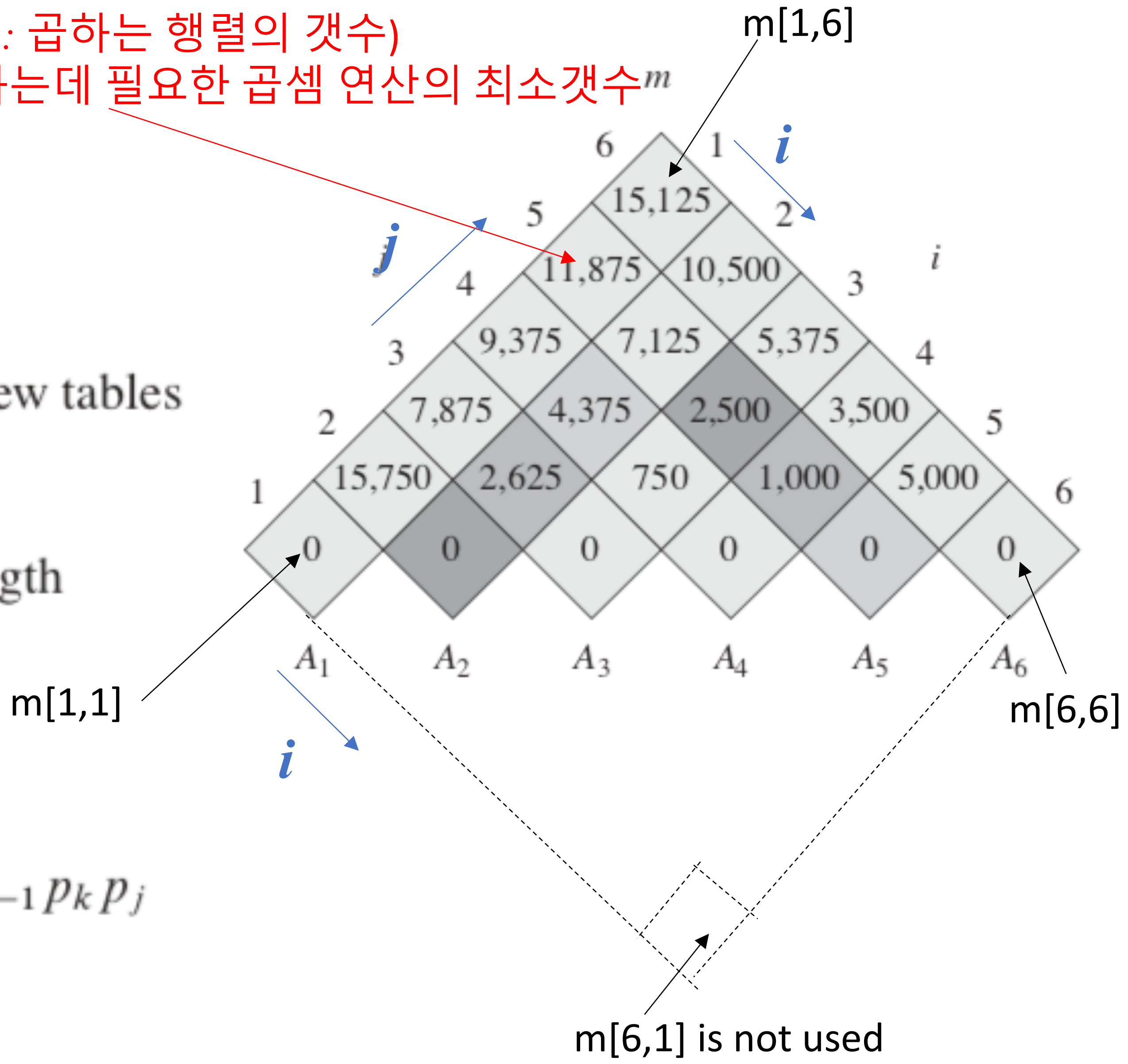
r_i for $i<n$ 으로부터 r_n 을 구할 수 있다.
→ optimal substructure 를 가졌다.

P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

m[i,j] : ($l=j-i+1$: 곱하는 행렬의 갯수)
$A_i \ldots A_j$ 를 계산하는데 필요한 곱셈 연산의 최소갯수

MATRIX-CHAIN-ORDER(p)

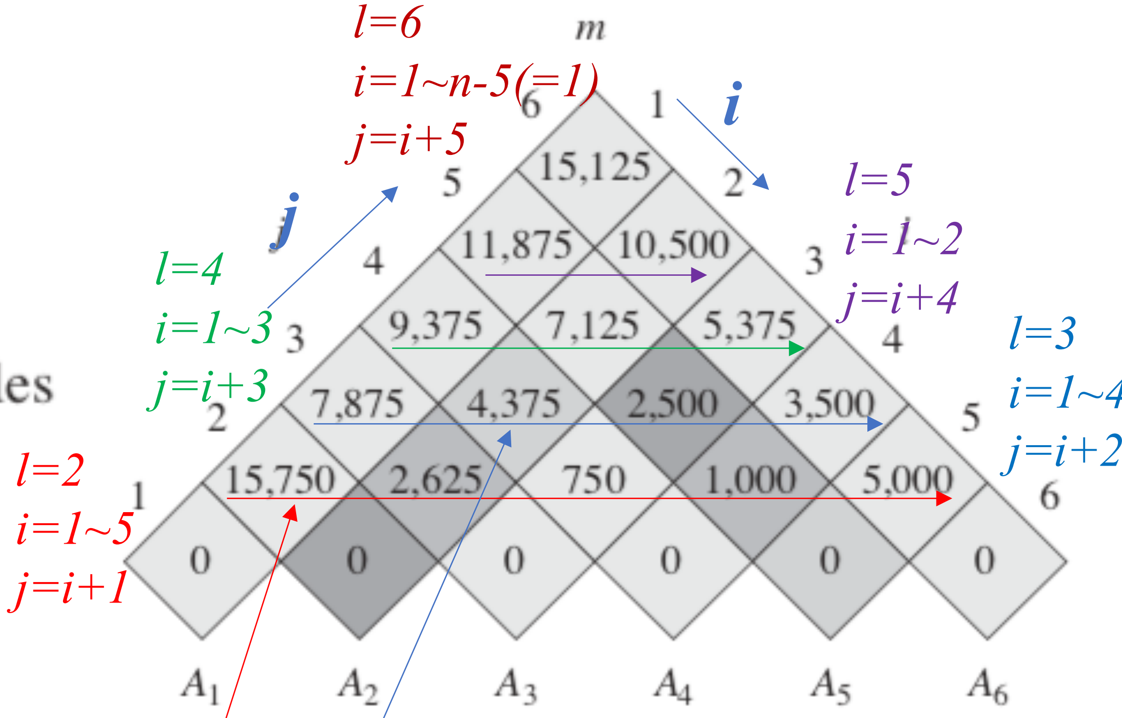
```
n = p.length − 1   (n=6)
let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
for i = 1 to n
    m[i, i] = 0
for l = 2 to n                    // l is the chain length
    for i = 1 to n − l + 1
        j = i + l − 1
        m[i, j] = ∞
        for k = i to j − 1
            q = m[i, k] + m[k + 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
```



P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

MATRIX-CHAIN-ORDER(p)

```
n = p.length − 1   (n=6)
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
```



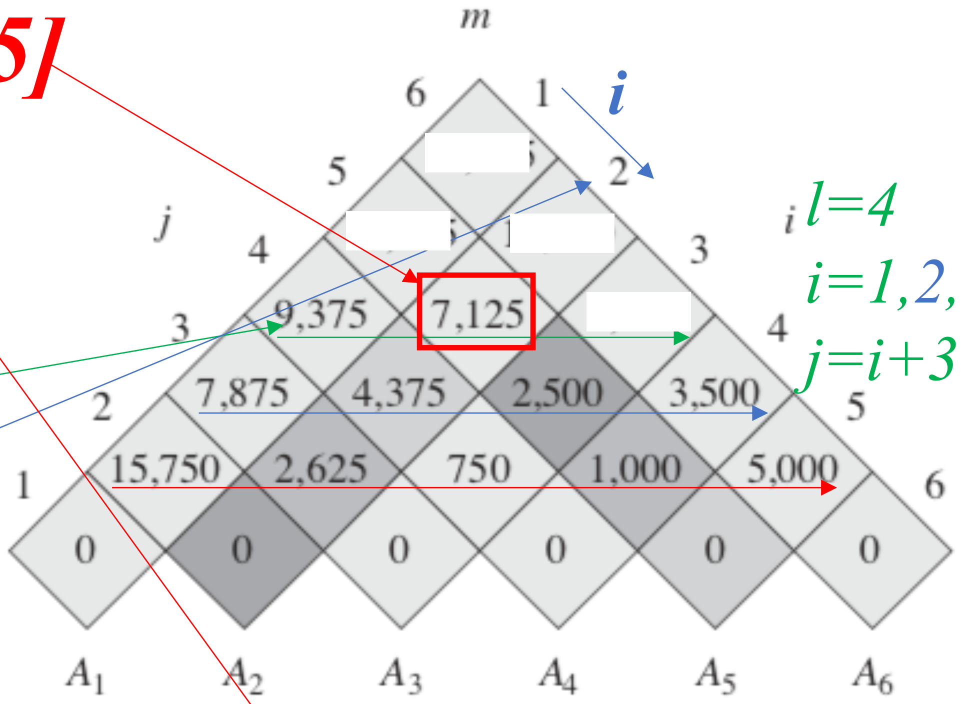
m[1,2] : (l=2 일 때 계산됨)
A_1A_2 를 계산하는데 필요한 곱셈 연산의 갯수
= $p_0 x p_1 x p_2$ = 30x35x15 = 15750

m[2,4] : (l=3 일 때 계산됨)
$A_2A_3A_4$ 를 계산하는데 필요한 곱셈 연산의 최소갯수
$A_2(A_3A_4)$ or $(A_2A_3)A_4$

computing *m[2,5]* and *s[2,5]*

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          when l=4  // l is the chain length
    for i = 1 to n − l + 1          when i=2
        j = i + l − 1                j=5
        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
```

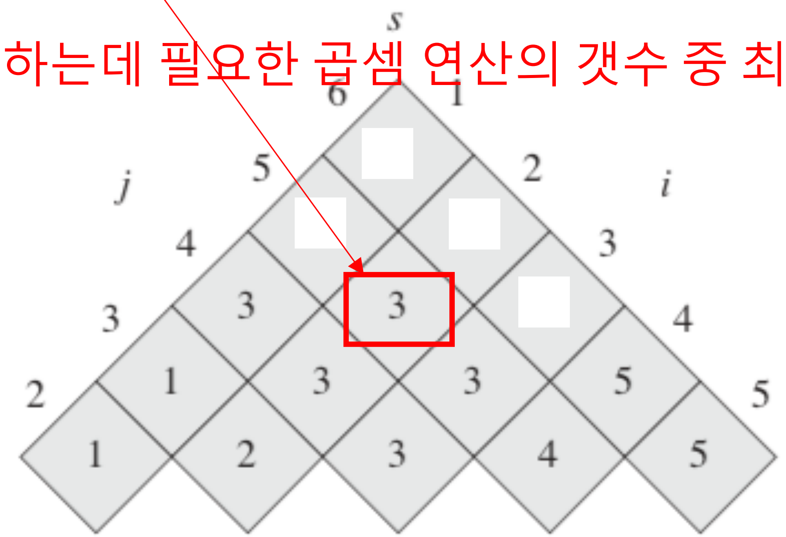


$l=4$

$i=1,2,3$

$j=i+3$

m[2,5] :
$A_2A_3A_4A_5$ 를 계산하는데 필요한 곱셈 연산의 갯수 중 최소값



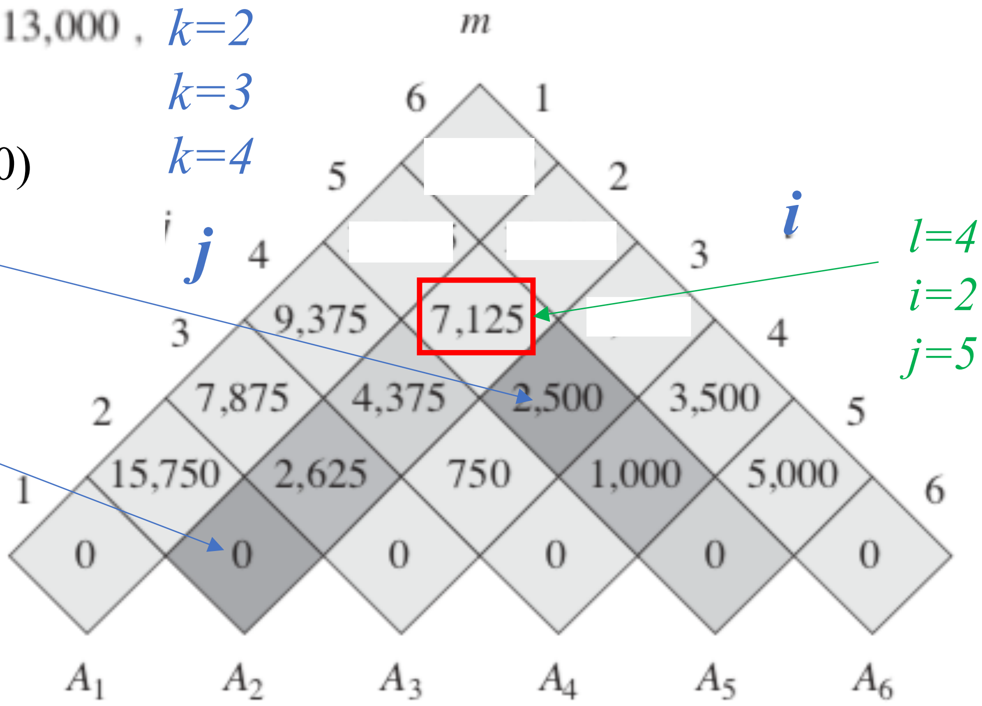
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = \end{cases}$$
$$= 7125.$$

$k=2$
$k=3$
$k=4$

$A_2 (A_3 A_4 A_5)$
(35x15)x(15x20)

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

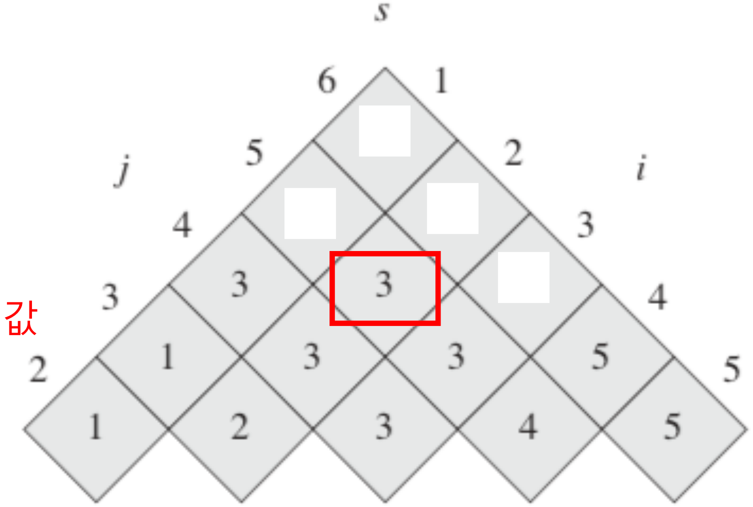


$l=4$
$i=2$
$j=5$

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      when l=4 // l is the chain length
    for i = 1 to n − l + 1      when i=2
        j = i + l − 1      j=5
        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
```

m[2,5] :
$A_2A_3A_4A_5$ 를 계산하는데 필요한 곱셈 연산의 갯수 중 최소값



P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

$$m[2,5] = \min \begin{cases} m[2,\boxed{2}] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, & k=2 \\ m[2,\boxed{3}] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, & k=3 \\ m[2,\boxed{4}] + m[5,5] + & k=4 \end{cases}$$

$$= 7125\,.$$

$(A_2 A_3)(A_4 A_5)$

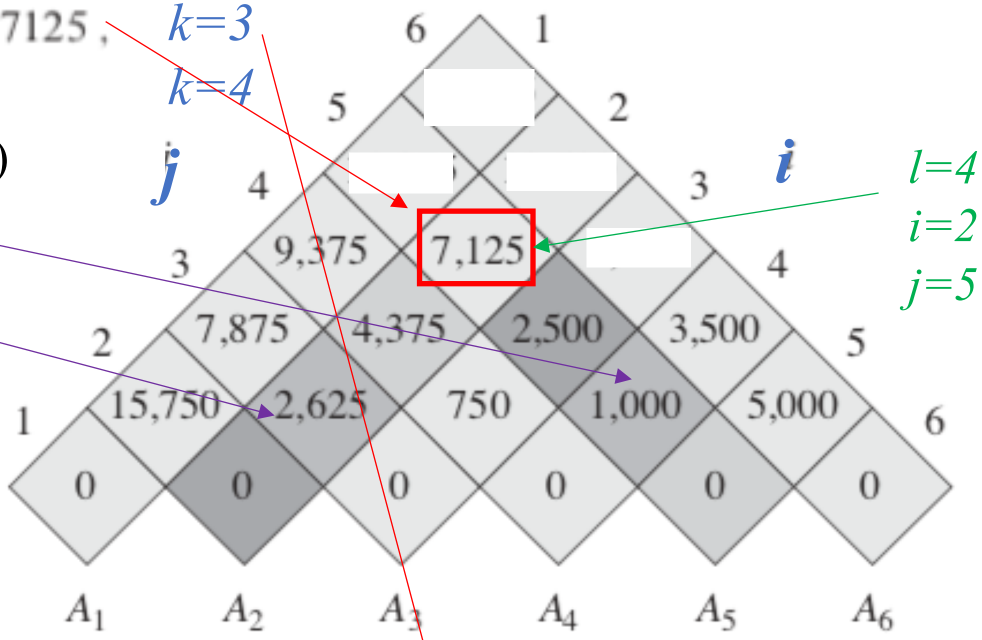
(35x5)x(5x20)

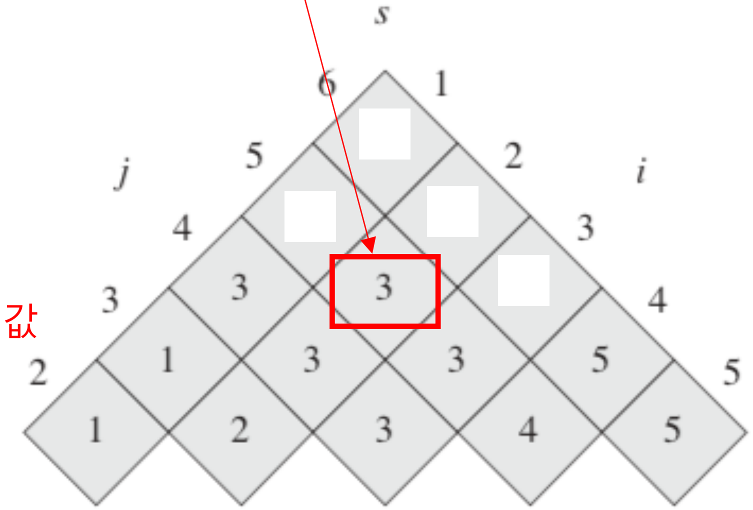
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

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        when l=4 // l is the chain length
    for i = 1 to n - l + 1        when i=2
        j = i + l - 1        j=5
        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
```

m[2,5] :
$A_2A_3A_4A_5$ 를 계산하는데 필요한 곱셈 연산의 갯수 중 최소값





P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

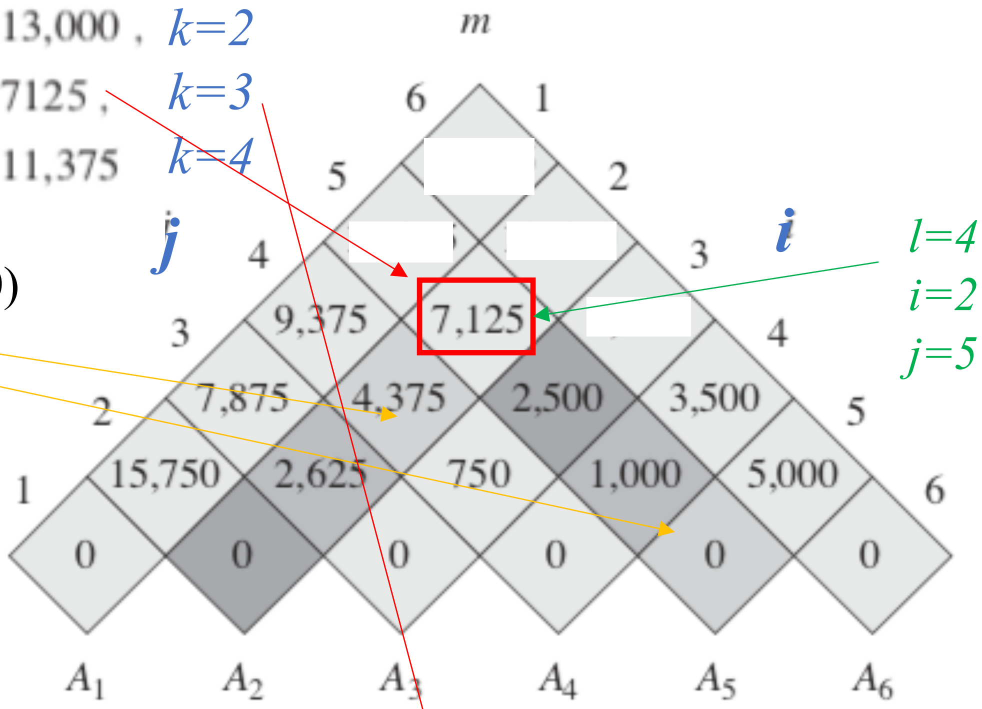
$$m[2,5] = \min \begin{cases} m[2,\boxed{2}] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, & k=2 \\ m[2,\boxed{3}] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, & k=3 \\ m[2,\boxed{4}] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 & k=4 \end{cases}$$

$$= 7125\,.$$

$(A_2A_3A_4)A_5$

(35x10)x(10x20)

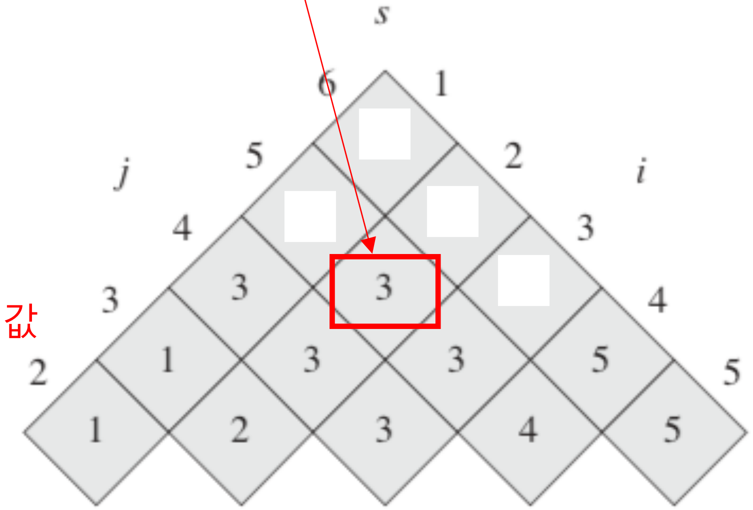
matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25



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          when l=4 // l is the chain length
    for i = 1 to n - l + 1      when i=2
        j = i + l - 1       j=5
        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
```

m[2,5] :
$A_2A_3A_4A_5$ 를 계산하는데 필요한 곱셈 연산의 갯수 중 최소값

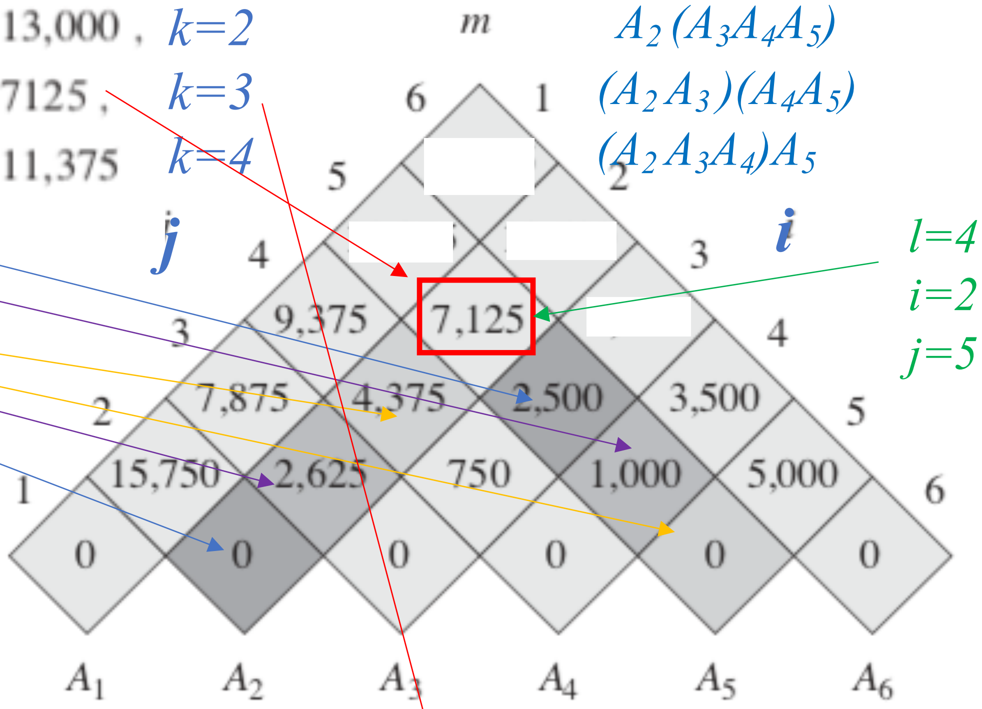


P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375 \end{cases}$$

$$= 7125\,.$$

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25



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
```

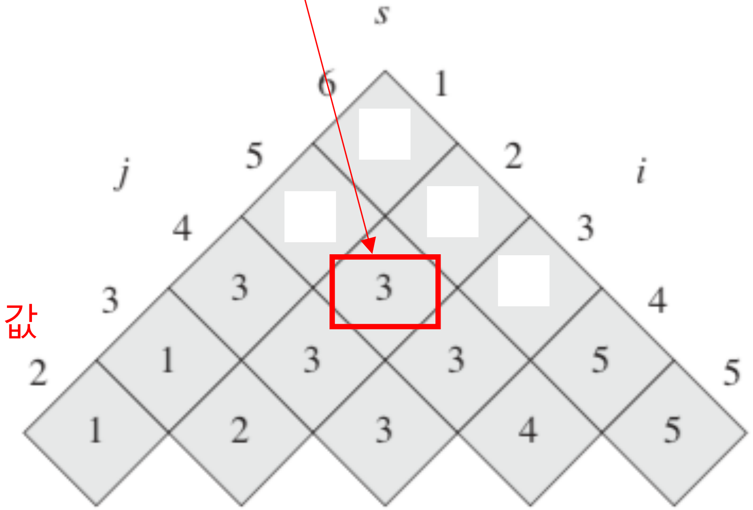
when l=4

when i=2

j=5

m[2,5] :

$A_2A_3A_4A_5$ 를 계산하는데 필요한 곱셈 연산의 갯수 중 최소값



P_0	P_1	P_2	P_3	P_4	P_5	P_6
30	35	15	5	10	20	25

Time complexity $= O(n^3)$

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
```

MATRIX-CHAIN-ORDER(p)

$\Theta(n^3)$

```
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
```

$$\sum_{i=0}^{n} c = (n+1)c$$

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1) \qquad \sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}$$

Time Complexity

$$\sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{j-1} c$$

$$= \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} \sum_{k=i}^{i+l-2} c$$

$$= \sum_{l=2}^{n} \sum_{i=1}^{n-l+1} (l-1)c$$

$$= \sum_{l=2}^{n} (n-l+1)(l-1)c$$

$$= c\sum_{i=1}^{n-1} (n-i)i$$

$$= c\sum_{i=1}^{n-1} ni - c\sum_{i=1}^{n-1} i^2$$

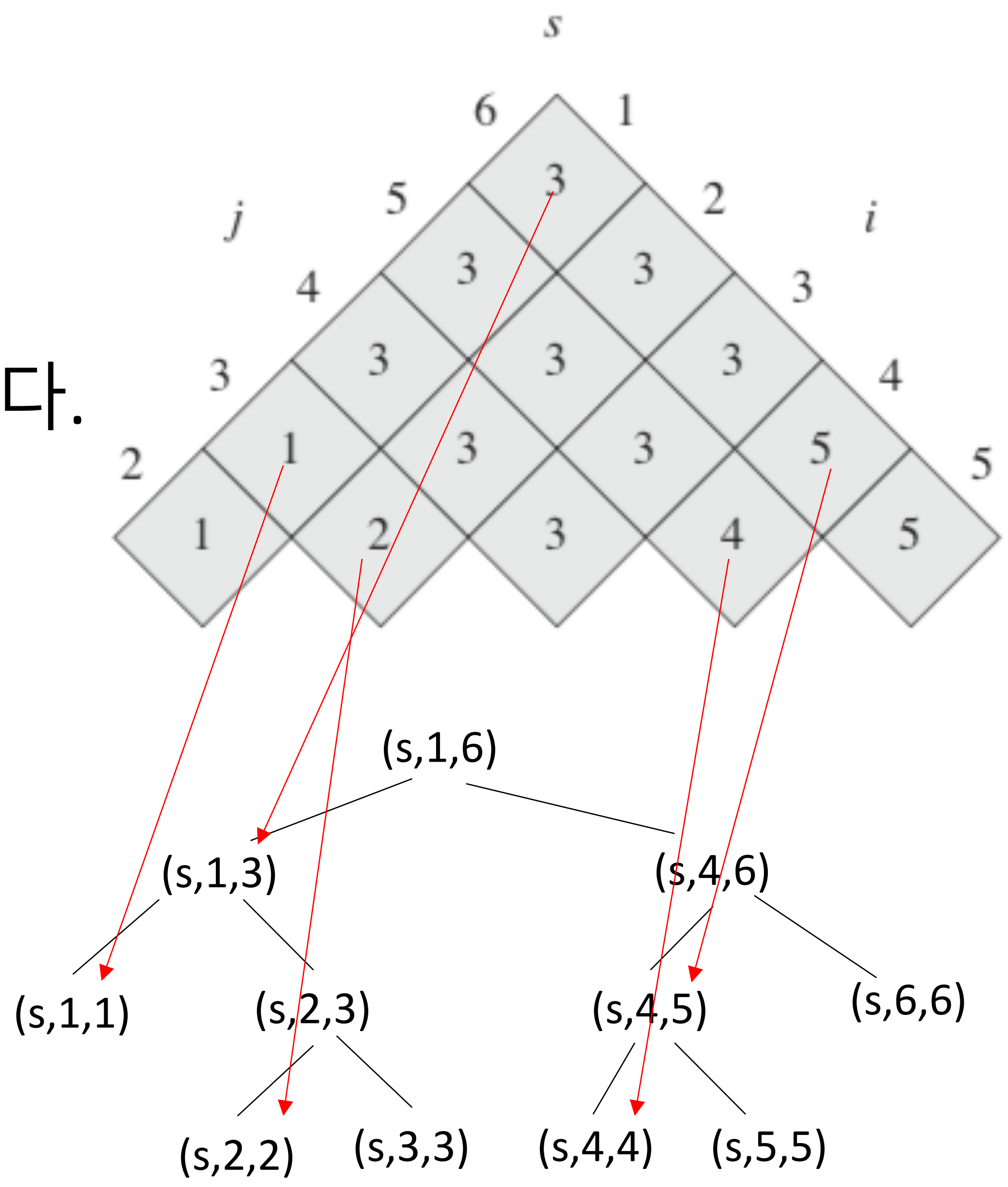
$$= cn\sum_{i=1}^{n-1} i - c\sum_{i=1}^{n-1} i^2$$

$$= c\left(n\frac{n(n-1)}{2} - \frac{(n-1)n(2n-1)}{6}\right)$$

$$= cn(n-1)\frac{3n-2n+1}{6} = \Theta(n^3)$$

4. 계산된 정보들로부터 최적해를 구성한다.

```
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 ")"
```



곱셈 횟수 = m[1,6] = 15125 회

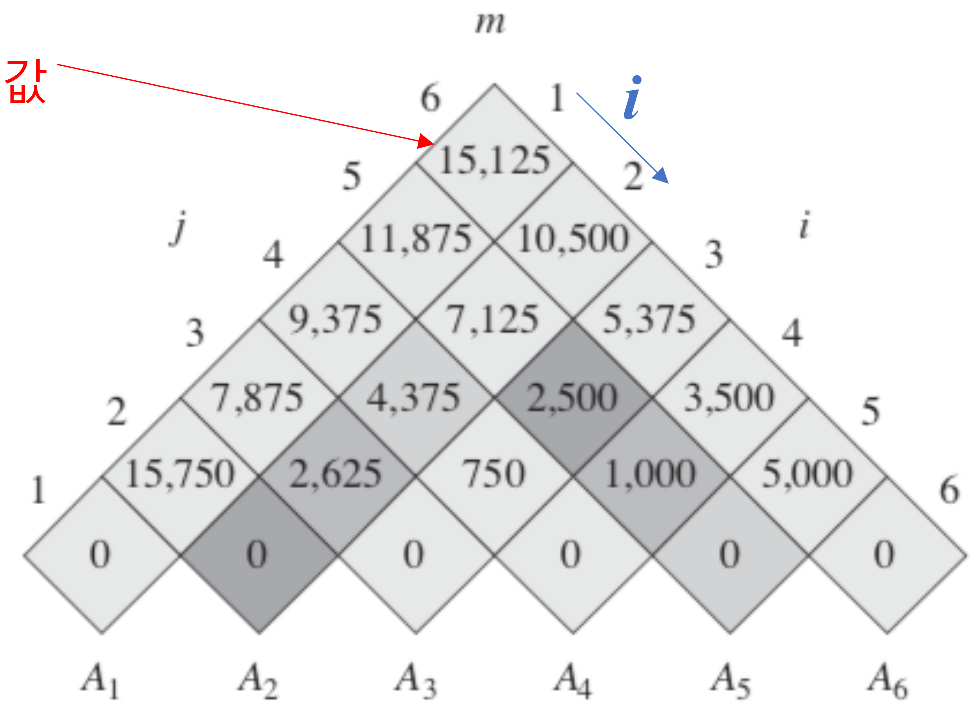
$$((A_1(A_2A_3))((A_4A_5)A_6))$$

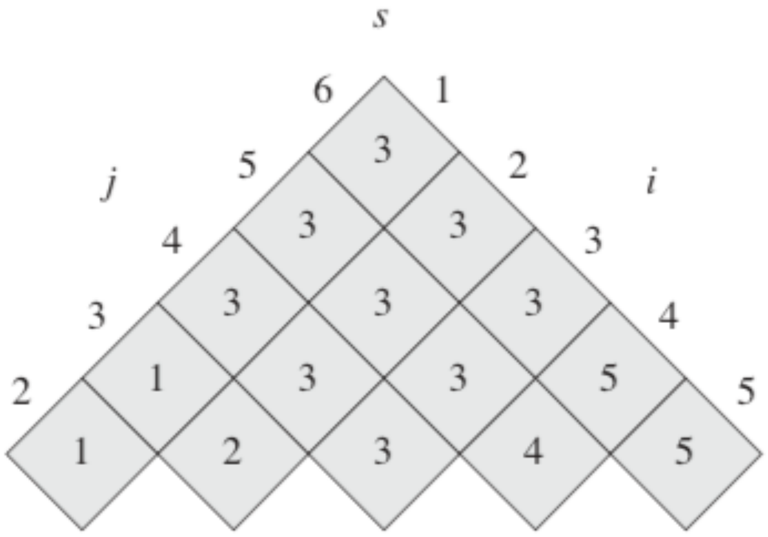
m[1,6] :
$A_1A_2A_3A_4A_5A_6$ 를 계산하는데 필요한 곱셈 연산의 갯수 중 최소값

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
```

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25





15.3 elements of dynamic programming

- matrix multiplication 을 dynamic programming 으로 풀 수 있는가?

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
n = A.rows
let C be a new n × n matrix
if n == 1
    c11 = a11 · b11
else partition A, B, and C as in equations (4.9)
    C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
    C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
    C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
    C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
          + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
return C
```

elements of dynamic programming 1

- Optimal substructure : 문제의 최적해가 subproblem 의 최적해를 포함한다.

elements of dynamic programming 2

- Overlapping subproblems : 최적화 문제의 부분 문제를 풀기 위한 재귀 알고리즘이 같은 문제를 반복해서 푼다.

15.4 Longest common subsequence (LCS)

- subsequence $Z = \langle z_1, z_2, \ldots, z_k \rangle$ of sequence $X = \langle x_1, x_2, \ldots, x_m \rangle$

단조 증가하는 X 의 인덱스 시퀀스 $\langle i_1, i_2, \ldots, i_k \rangle$ such that $x_{i_j} = z_j$ 가 있다.

i.e. X = <A,B,C,B,D,A,B>

의 subsequence Z = <B,C,D,B> for <2,3,5,7>

- common subsequence Z of X and Y : Z is subsequence of X, and of Y.

brute-force approach for finding LCS Z of X and Y

- LCS: Longest Common Subsequence
- X의 모든 subsequence X'를 찾음 (2^m 개, m=X의 길이)
- X'이 Y 의 subsequence 인지 확인하고 가장 긴 것을 찾음

= $O(n2^m)$ n=Y의 길이

Theorem 15.1 (Optimal substructure of an LCS)
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

$X_{m-1} = \langle x_1, x_2, \ldots x_{m-1} \rangle$

concatenation of X and a: $X|a = \langle x_1, x_2, \ldots x_m , a \rangle$

Theorem 15.1 (Optimal substructure of an LCS)
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

X = ABACAB
Y = BCACCB
Z = BACB

1. X 와 Y 의 마지막 글자(x_m)가 같으면 (1) LCS Z의 마지막 글자 (z_k) 도 x_m 이고 (2) Z_{k-1} 은 X_{m-1} 과 Y_{n-1} 의 LCS 이다.

proof by contradiction)

$z_k \neq x_m$ 라면 (1) $Z|x_m$ 는 X와 Y의 common subseq. 이다. $Z|x_m$ 의 길이가 k+1 이므로 Z는 LCS가 아니다. 모순. 따라서 $z_k = x_m = y_n$.

(2) 그러면 Z_{k-1} 은 X_{m-1} 의 subseq. 이면서 Y_{n-1} 의 subseq.이다. 즉, X_{m-1} 과 Y_{n-1} 의 common subseq.이다. 그런데 Z_{k-1} 이 X_{m-1} 과 Y_{n-1} 의 LCS 가 아니라면 k-1 보다 길이가 긴 LCS W가 있을텐데 $W|x_m$ 은 X와 Y의 common subsequence 이고 길이는 k보다 크다. 그러면 Z는 LCS가 아니다. 모순. 따라서 Z_{k-1} 은 X_{m-1} 과 Y_{n-1} 의 LCS 이다.

Theorem 15.1 (Optimal substructure of an LCS)

Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

X = ABACABA
Y = BCACBBC
Z = BACB

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

2. X 와 Y 의 마지막 글자가 다른 경우 LCS Z의 마지막 글자가 x_m 이 아니면 Z는 X_{m-1} 과 Y 의 LCS 이다.

proof by contradiction)

$z_k \neq x_m$라면 Z는 X_{m-1} 의 subsequence 이고 Y의 subsequence 이다. 즉, Z는 X_{m-1} 과 Y의 common subsequence 이다. X_{m-1} 과 Y의 common subsequence W가 있고 그 길이가 k보다 크다면 그 W 가 X와 Y의 LCS 가 될 것이므로 Z는 X와 Y의 LCS가 될 수 없다. 모순 → 따라서 Z는 X_{m-1} 과 Y 의 LCS 이다.

Theorem 15.1 (Optimal substructure of an LCS)
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

X = ABACAB
Y = BCACCBC
Z = BACB

3. X 와 Y 의 마지막 글자가 다른 경우 LCS Z의 마지막 글자가 y_n 이 아니면 Z는 X 과 Y_{n-1} 의 LCS 이다.

proof by contradiction)

$z_k \neq y_n$ 라면 Z는 X의 subsequence 이고 Y_{n-1} 의 subsequence 이다. 즉, Z는 X 과 Y_{n-1}의 common subsequence 이다. X 과 Y_{n-1}의 common subsequence W가 있고 그 길이가 k보다 크다면 그 W 가 X와 Y의 LCS 가 될 것이므로 Z는 X와 Y의 LCS가 될 수 없다. 모순 → 따라서 Z는 X 과 Y_{n-1} 의 LCS 이다.

Theorem 15.1 (Optimal substructure of an LCS)
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

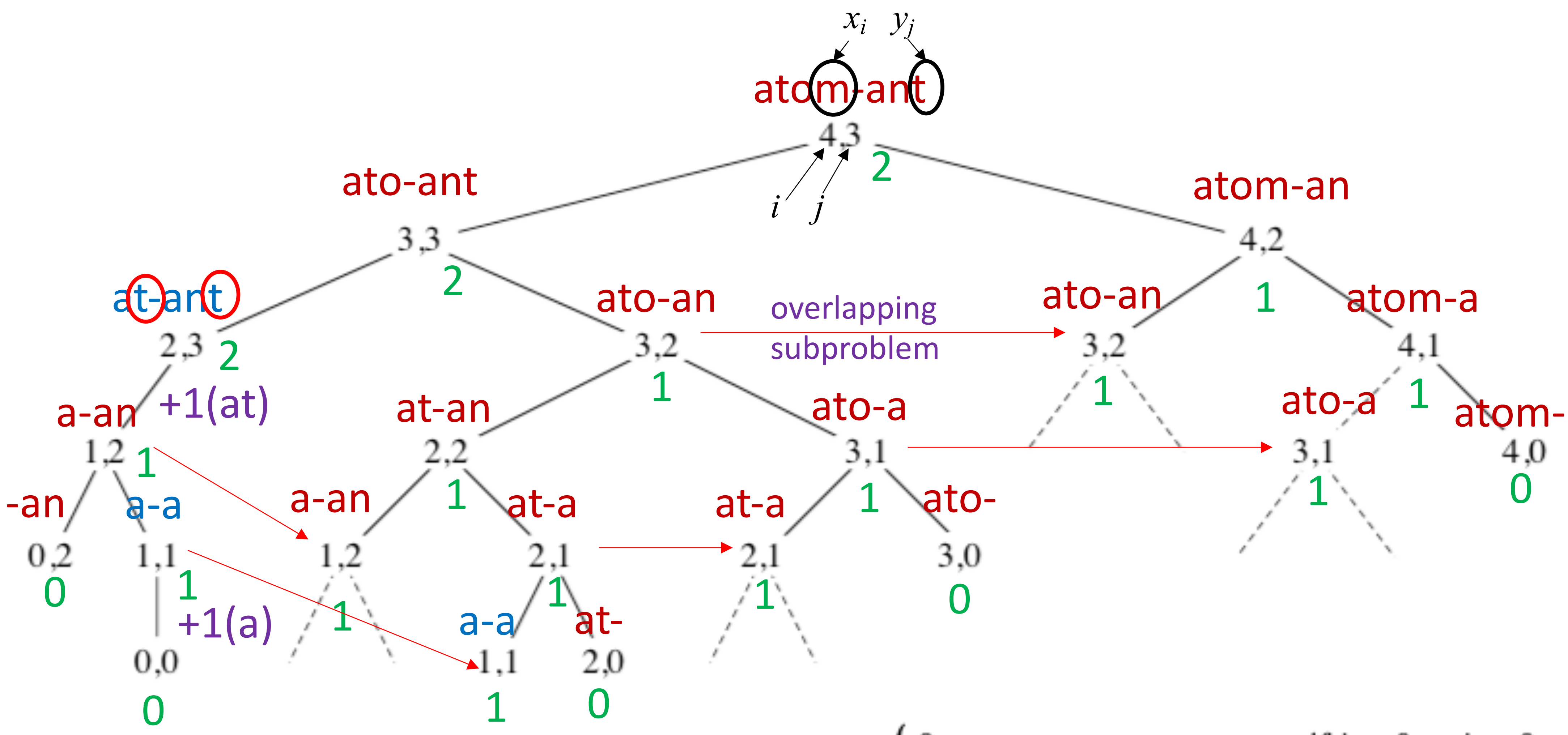
1. If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.
2. If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y.
3. If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}.

→

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j\,, \\ \max(c[i-1, j], c[i, j-1]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

$c[i,j] =$ length of LCS of X_i and Y_j.

$$X = \langle a, t, o, m \rangle \text{ and } Y = \langle a, n, t \rangle$$



$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 , \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j , \\ \max(c[i-1, j], c[i, j-1]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j . \end{cases}$$

LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

X_i와 Y_j의 LCS를 찾음

case 1

case 2 or 3

X_1(A)와 Y_4(BDCA)의 LCS 길이

	j	0	1	2	3	4	5	6	
i		y_j	B	D	C	A	B	A	
0	x_i	0	0	0	0	0	0	0	
1	A	0				?			b c
2	B	0							
3	C	0							
4	B	0							
5	D	0							
6	A	0							
7	B	0							

b행렬과 c 행렬을 같이 표시한 것

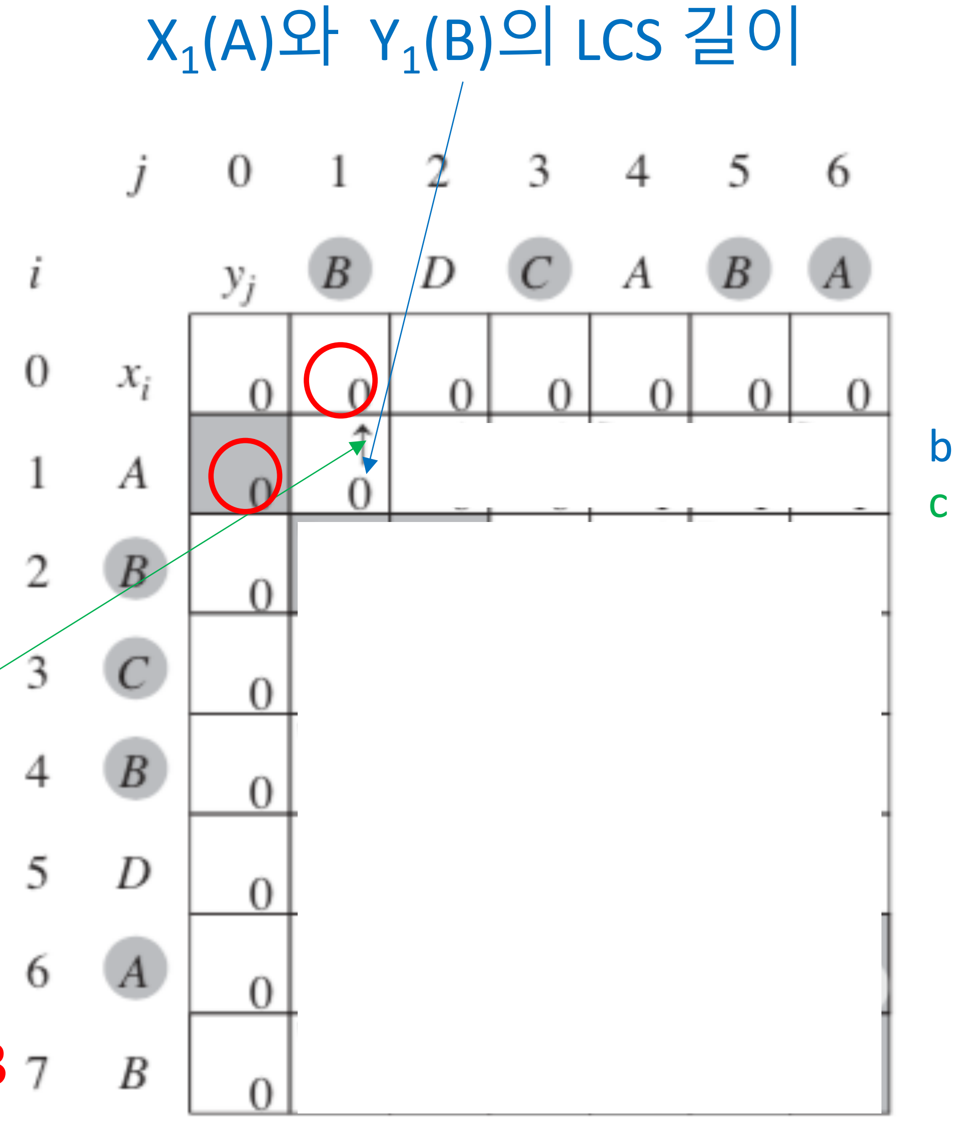
LCS-LENGTH(X, Y)

```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

X_i와 Y_j의 LCS를 찾음

case 1

case 2or3



LCS-LENGTH(X, Y)

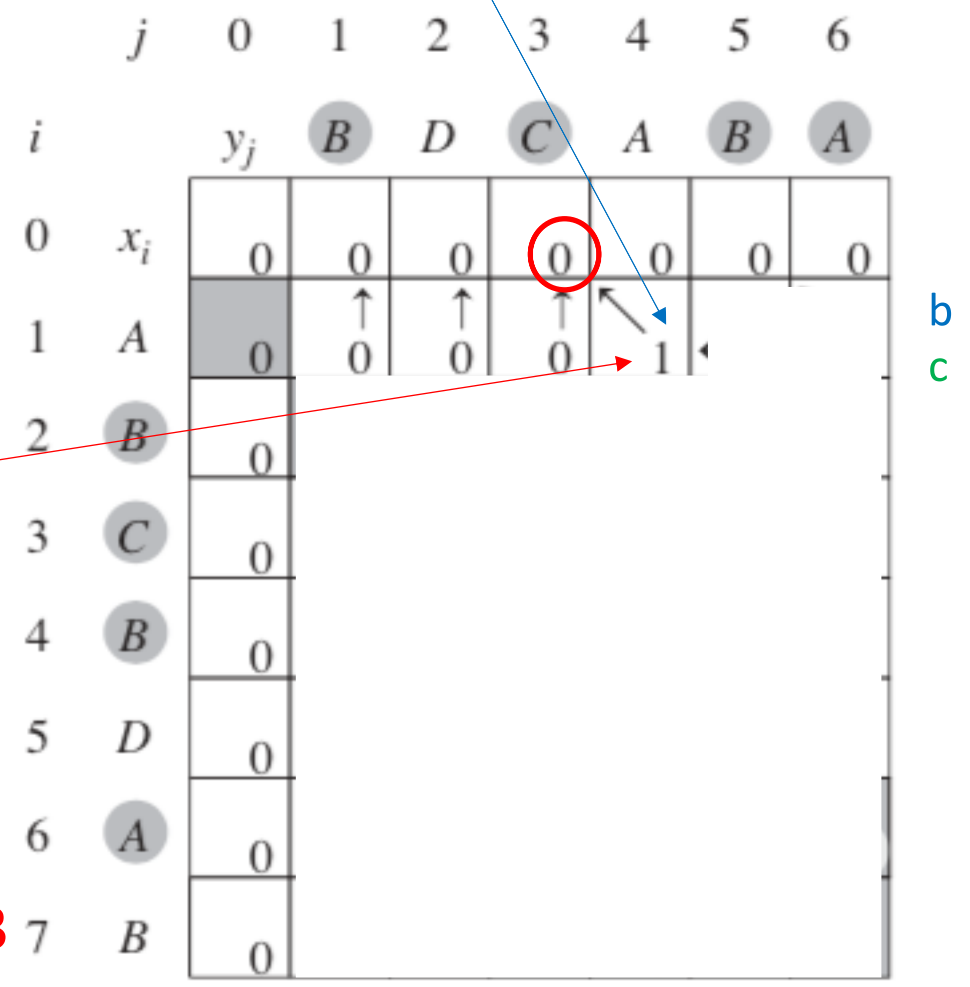
```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

X_i와 Y_j의 LCS를 찾음

case 1

case 2or3

X_1(A)와 Y_4(BDCA)의 LCS 길이



LCS-LENGTH(X, Y)

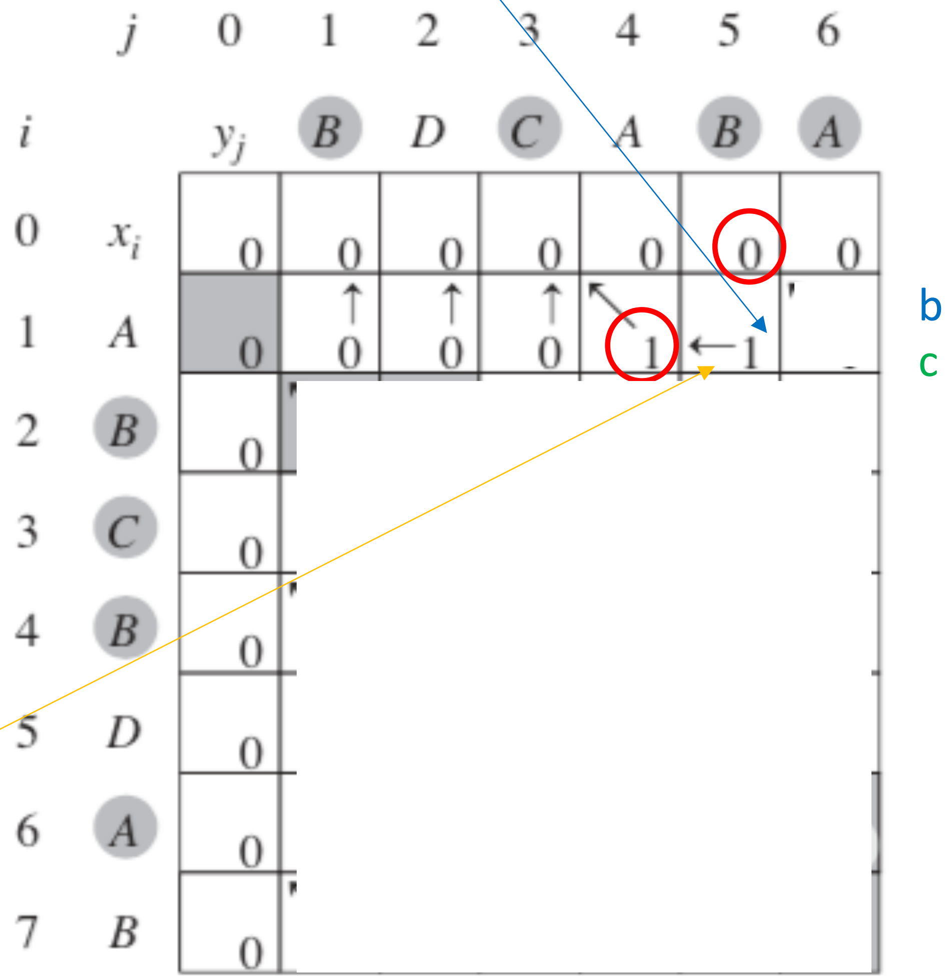
```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
                if x_i == y_j
                        c[i, j] = c[i − 1, j − 1] + 1
                        b[i, j] = "↖"
                elseif c[i − 1, j] ≥ c[i, j − 1]
                        c[i, j] = c[i − 1, j]
                        b[i, j] = "↑"
                else c[i, j] = c[i, j − 1]
                        b[i, j] = "←"
return c and b
```

X_i와 Y_j의 LCS를 찾음

case 1

case 2or3

X_1(A)와 Y_5(BDCAB)의 LCS 길이



X_7(ABCBDAB)와 Y_6(BDCABA)의 LCS 길이

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
        c[i, 0] = 0
for j = 0 to n
        c[0, j] = 0
for i = 1 to m
        for j = 1 to n
            if x_i == y_j
                c[i, j] = c[i − 1, j − 1] + 1
                b[i, j] = "↖"
            elseif c[i − 1, j] ≥ c[i, j − 1]
                c[i, j] = c[i − 1, j]
                b[i, j] = "↑"
            else c[i, j] = c[i, j − 1]
                b[i, j] = "←"
return c and b
```

X_i와 Y_j의 LCS를 찾음

case 1

case 2or3

i \ j		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

b

c

LCS-LENGTH(X, Y)

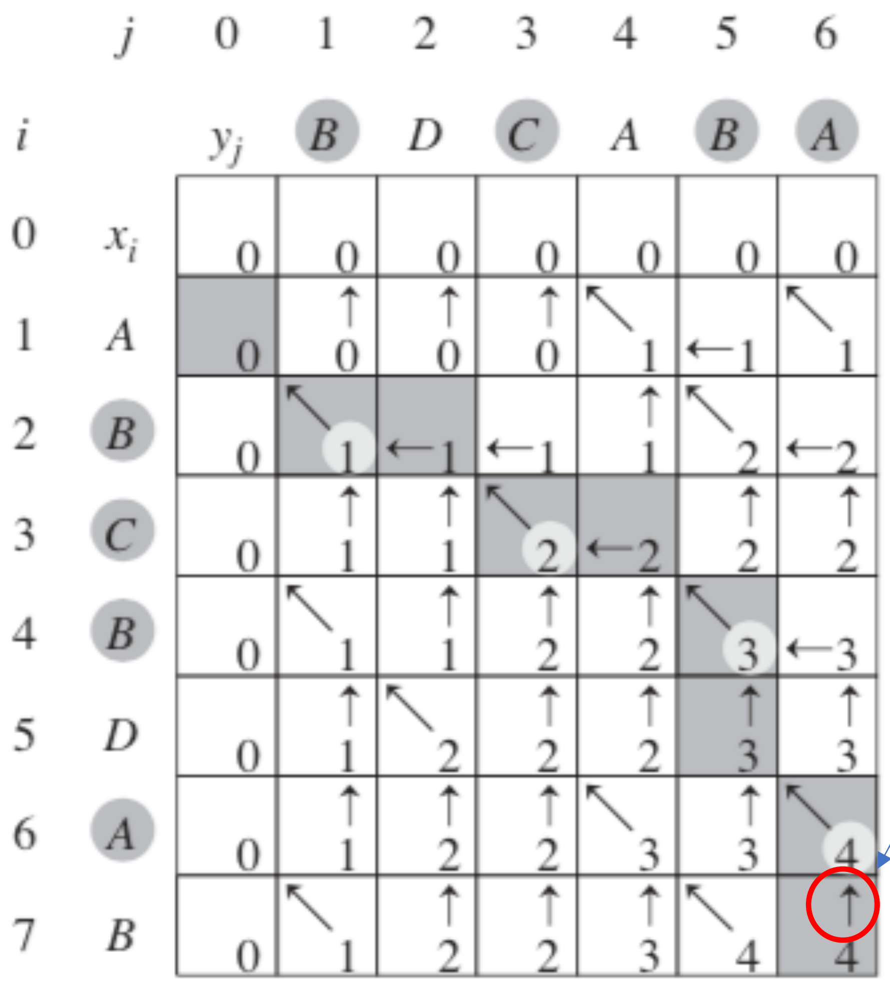
```
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

$= \Theta(mn)$

“ABCBDAB”

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```



PRINT-LCS(b,X,7,6)

|

PRINT-LCS(b,X,6,6)

"ABCBDAB"

step 4. constructing LCS : PRINT-LCS(b, X, m, n)

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

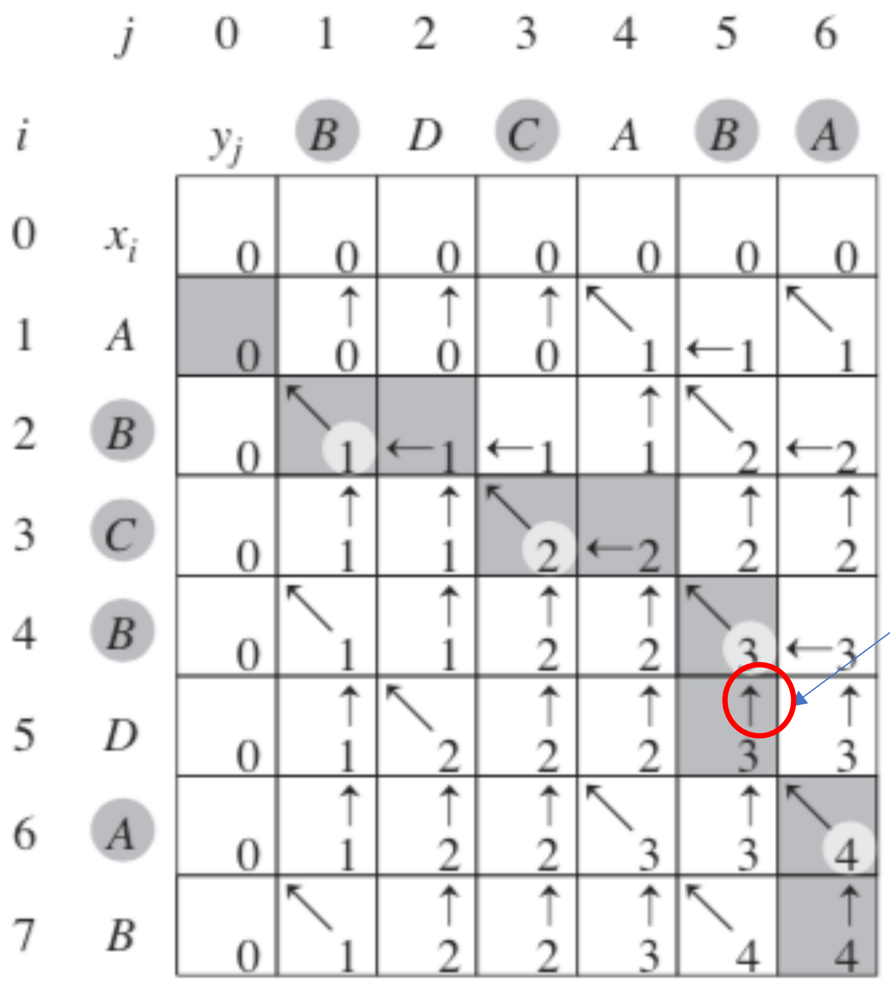
i \ j		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

PRINT-LCS(b,X,7,6)
|
PRINT-LCS(b,X,6,6)
|
PRINT-LCS(b,X,5,5) "A"

“ABCBDAB”

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j == 0
      return
  if b[i, j] == "↖"
      PRINT-LCS(b, X, i − 1, j − 1)
      print x_i
  elseif b[i, j] == "↑"
      PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```



PRINT-LCS(b,X,7,6)

|

PRINT-LCS(b,X,6,6)

|

PRINT-LCS(b,X,5,5) “A”

|

PRINT-LCS(b,X,4,5)

“ABCBDAB”

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

PRINT-LCS(b, X, i, j)

```
if i == 0 or j == 0
        return
if b[i, j] == "↖"
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
elseif b[i, j] == "↑"
        PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

PRINT-LCS(b,X,7,6)

PRINT-LCS(b,X,6,6)

PRINT-LCS(b,X,5,5) “A”

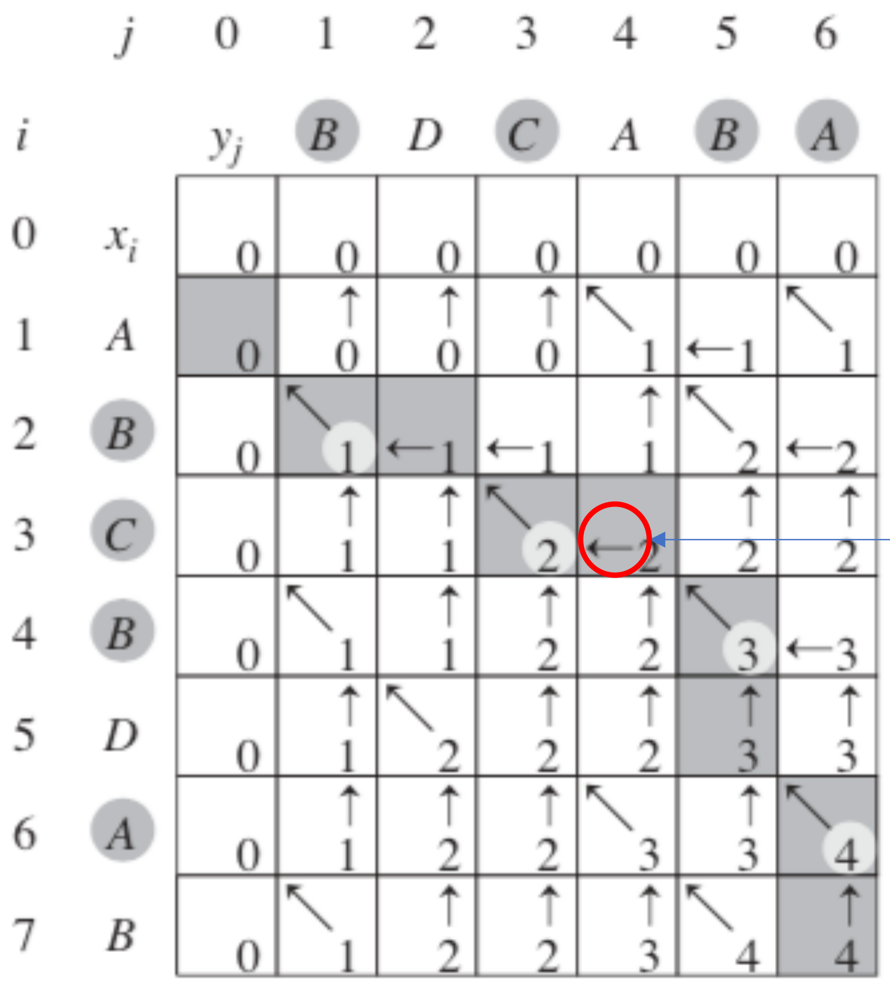
PRINT-LCS(b,X,4,5)

PRINT-LCS(b,X,3,4) “B”

“ABCBDAB”

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
    if i == 0 or j == 0
        return
    if b[i, j] == "↖"
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
    elseif b[i, j] == "↑"
        PRINT-LCS(b, X, i − 1, j)
    else PRINT-LCS(b, X, i, j − 1)
```



PRINT-LCS(b,X,7,6)

|

PRINT-LCS(b,X,6,6)

|

PRINT-LCS(b,X,5,5) “A”

|

PRINT-LCS(b,X,4,5)

|

PRINT-LCS(b,X,3,4) “B”

|

PRINT-LCS(b,X,3,3)

"ABCBDAB"

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

PRINT-LCS(b,X,7,6)

PRINT-LCS(b,X,6,6)

PRINT-LCS(b,X,5,5) "A"

PRINT-LCS(b,X,4,5)

PRINT-LCS(b,X,3,4) "B"

PRINT-LCS(b,X,3,3)

PRINT-LCS(b,X,2,2) "C"

“ABCBDAB”

step 4. constructing LCS : PRINT-LCS(b, X, m, n)

PRINT-LCS(b, X, i, j)

```
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

PRINT-LCS(b,X,7,6)
|
PRINT-LCS(b,X,6,6)
|
PRINT-LCS(b,X,5,5) “A”
|
PRINT-LCS(b,X,4,5)
|
PRINT-LCS(b,X,3,4) “B”
|
PRINT-LCS(b,X,3,3)
|
PRINT-LCS(b,X,2,2) “C”
|
PRINT-LCS(b,X,2,1)

"ABCBDAB"

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
    if i == 0 or j == 0
        return
    if b[i, j] == "↖"
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
    elseif b[i, j] == "↑"
        PRINT-LCS(b, X, i − 1, j)
    else PRINT-LCS(b, X, i, j − 1)
```

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

PRINT-LCS(b,X,7,6)
|
PRINT-LCS(b,X,6,6)
|
PRINT-LCS(b,X,5,5) "A"
|
PRINT-LCS(b,X,4,5)
|
PRINT-LCS(b,X,3,4) "B"
|
PRINT-LCS(b,X,3,3)
|
PRINT-LCS(b,X,2,2) "C"
|
PRINT-LCS(b,X,2,1)
|
PRINT-LCS(b,X,1,0) "B"

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

“ABCBDAB”

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑0	↑0	↑0	↖1	←1	↖1
2	B	0	↖1	←1	←1	↑1	↖2	←2
3	C	0	↑1	↑1	↖2	←2	↑2	↑2
4	B	0	↖1	↑1	↑2	↑2	↖3	←3
5	D	0	↑1	↖2	↑2	↑2	↑3	↑3
6	A	0	↑1	↑2	↑2	↖3	↑3	↖4
7	B	0	↖1	↑2	↑2	↑3	↖4	↑4

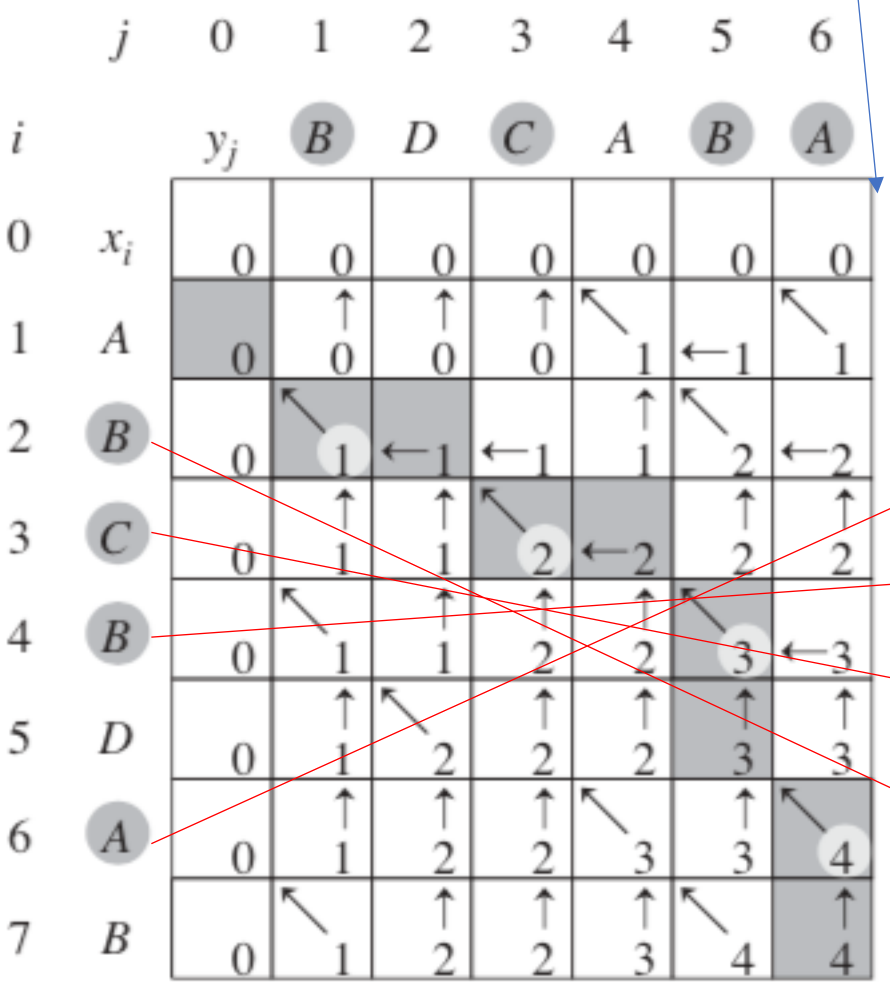
PRINT-LCS(b,X,7,6)
|
PRINT-LCS(b,X,6,6)
|
PRINT-LCS(b,X,5,5) “A”
|
PRINT-LCS(b,X,4,5)
|
PRINT-LCS(b,X,3,4) “B”
|
PRINT-LCS(b,X,3,3)
|
PRINT-LCS(b,X,2,2) “C”
|
PRINT-LCS(b,X,2,1)
|
PRINT-LCS(b,X,1,0) “B”

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

$= O(m+n)$

”BCBA”



PRINT-LCS(b,X,7,6)
PRINT-LCS(b,X,6,6)
PRINT-LCS(b,X,5,5) “A”
PRINT-LCS(b,X,4,5)
PRINT-LCS(b,X,3,4) “B”
PRINT-LCS(b,X,3,3)
PRINT-LCS(b,X,2,2) “C”
PRINT-LCS(b,X,2,1)
PRINT-LCS(b,X,1,0) “B”

step 4. constructing LCS : PRINT-LCS(b,X,m,n)

```
PRINT-LCS(b, X, i, j)
  if i == 0 or j == 0
        return
  if b[i, j] == "↖"
        PRINT-LCS(b, X, i − 1, j − 1)
        print x_i
  elseif b[i, j] == "↑"
        PRINT-LCS(b, X, i − 1, j)
  else PRINT-LCS(b, X, i, j − 1)
```

$= O(m+n)$

j		0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

"BCBA"

PRINT-LCS(b,X,7,6)
|
PRINT-LCS(b,X,6,6)
|
PRINT-LCS(b,X,5,5) "A"
|
PRINT-LCS(b,X,4,5)
|
PRINT-LCS(b,X,3,4) "B"
|
PRINT-LCS(b,X,3,3)
|
PRINT-LCS(b,X,2,2) "C"
|
PRINT-LCS(b,X,2,1)
|
PRINT-LCS(b,X,1,0) "B"