Dynamic Programming

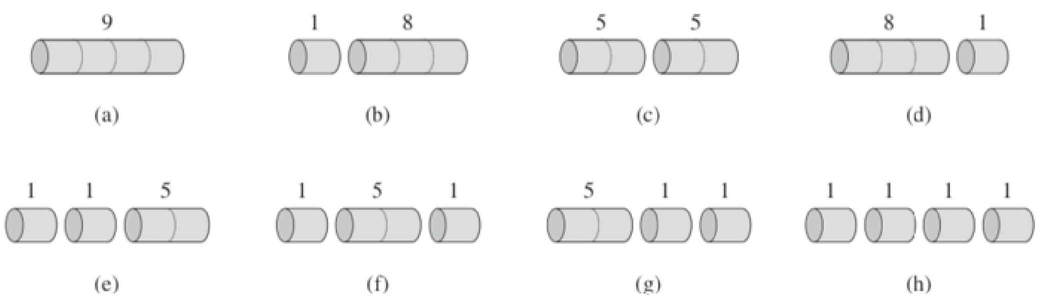
- 표를 만들어 채워가면서 답을 구하는 방법
- Divide and Conquer 와의 차이점 : overlaps in subproblems
- Meaning of "programming" here : tabular method
- Used in solving optimization problem
    - find an optimal solution, as opposed to the optimal solution
- 순서
    - 최적해의 구조적 특징을 찾는다
    - 최적해의 값을 재귀적으로 정의한다
    - 최적해의 값을 일반적으로 상향식 방법으로 계산한다
    - 계산된 정보들로부터 최적해를 구성한다

Rod Cutting

- n인치 막대를 잘라서 판매하여 얻을 수 있는 최대 수익 r_n을 찾아라
- 막대를 자르는 비용은 0
- price table

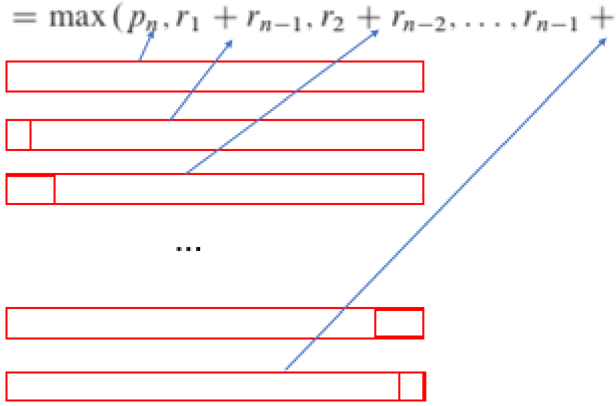
length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

- 4인치 로드를 자르는 법



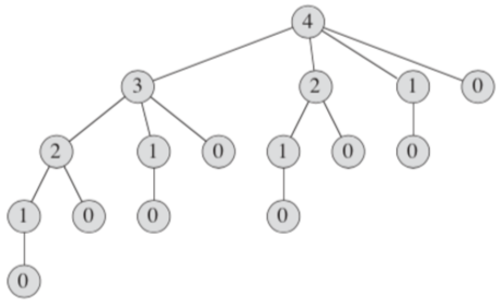
- 자르는 방법은 2^n가지(각 1칸 별로 자르거나 말거나 경우의 수)
- r_i for i < n 으로부터 r_n을 구할 수 있음
    - Optimal Substructure를 가졌다

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1) \quad \Rightarrow \quad r_n = \max_{1 \le i \le n}(p_i + r_{n-i})$$



- 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
```



$$T(n) = 1 + \sum_{j=0}^{n-1} T(j)\,. \quad \rightarrow T(n) = 2^n$$

- top down

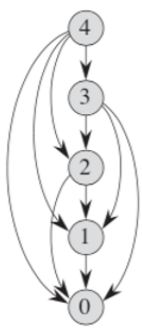
```
MEMOIZED-CUT-ROD(p, n)
let r[0..n] be a new array
for i = 0 to n
    r[i] = −∞
return MEMOIZED-CUT-ROD-AUX(p, n, r)
```

$$\Theta(n^2)$$

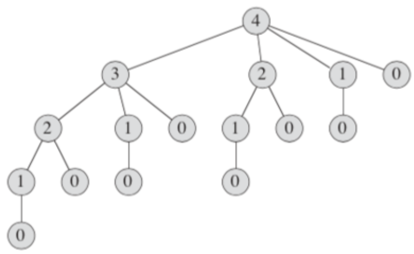
```
MEMOIZED-CUT-ROD-AUX(p, n, r)
if r[n] ≥ 0
    return r[n]
if n == 0
    q = 0
else q = −∞
    for i = 1 to n
        q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r))
r[n] = q
return q
```

- 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]
```



vs.



subproblem graph

$$\Theta(n^2)$$

- Reconstructing a solution : $\theta(n^2)$

```
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]
```

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

Matrix 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
```

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

$(A_1A_2)A_3 : 2\times3\times5 + 2\times5\times6 = 90$

$A_1(A_2A_3) : 3\times5\times6 + 2\times3\times6 = 126$

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

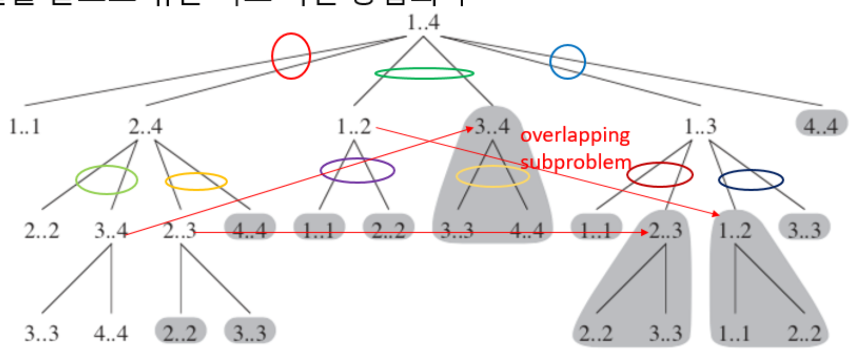
- 행렬 곱셈의 순서를 정하는 문제(곱셈 연산 X)

- Exhaustive search when n = 4

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

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

P(n) : n 개의 행렬을 괄호로 묶는 서로 다른 방법의 수



- 순서대로
  - 최적해의 구조적 특징을 찾는다
  - 최적해의 값을 재귀적으로 정의한다

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

$m[i,j]$: 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

- 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 $\Omega(2^n)$

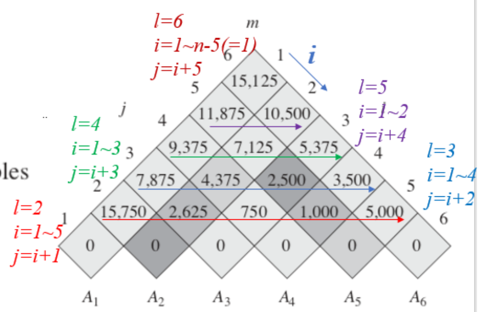
- optimal substructure 를 가지고 subproblem 들이 overlapped 되어있다.

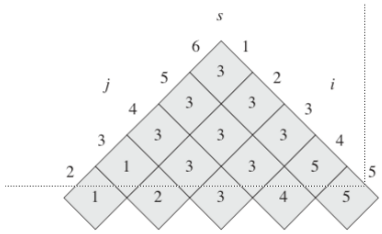
→ dynamic programming 의 조건

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
```



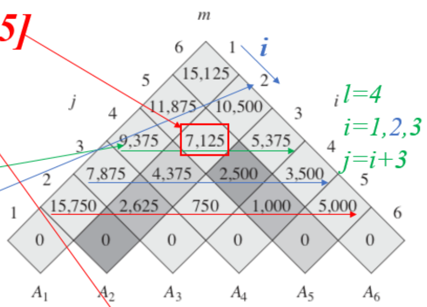


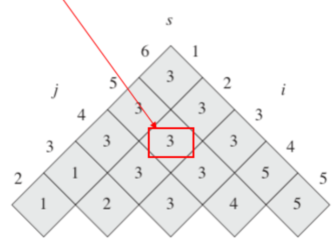
computing *m[2,5]* and *s[2,5]*

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
```

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

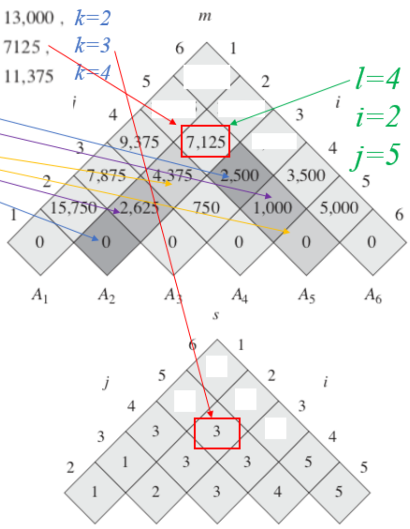




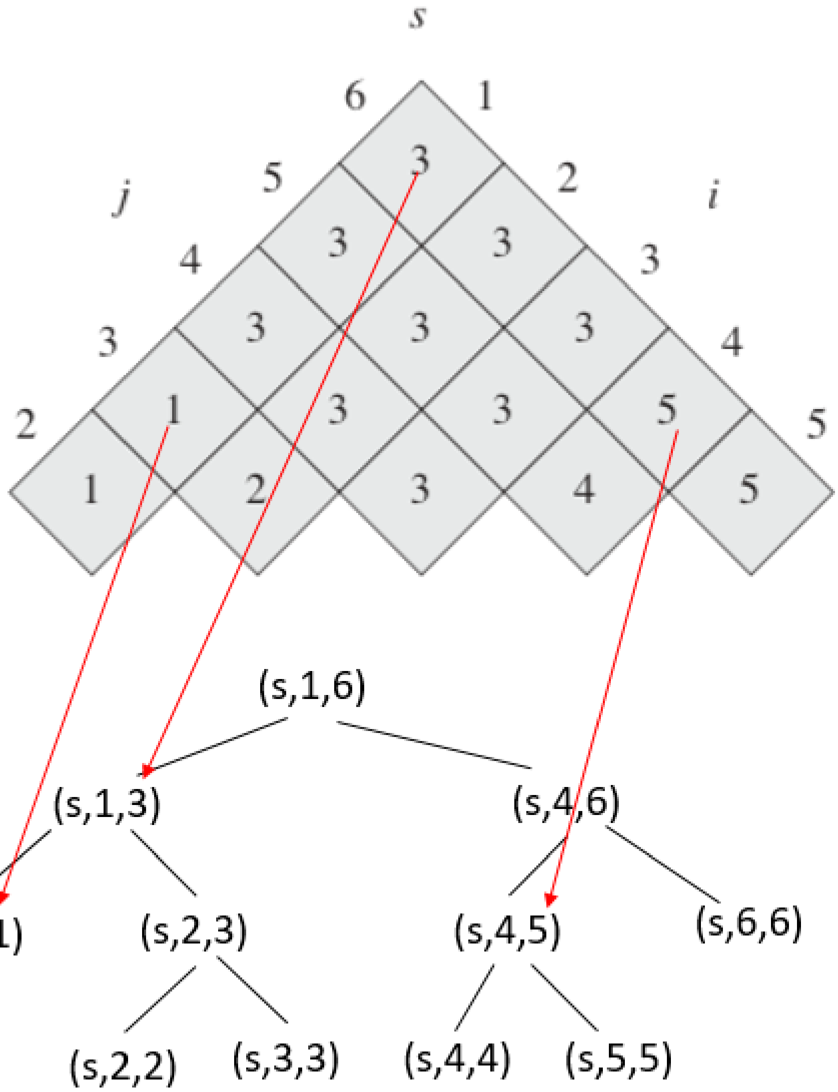
$$m[2,5] = \min \begin{cases} \boxed{m[2,2]} + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13{,}000\,, & k=2 \\ \boxed{m[2,3]} + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125\,, & k=3 \\ \boxed{m[2,4]} + m[5,5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11{,}375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

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
```



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



- 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 ")"
```

- Elements of dynamic programming
  - Optimal Structure : 문제의 최적해가 subproblem의 최적해를 포함한다
  - Overlapping Subproblems : 최적화 문제의 부분 문제를 풀기 위한 재귀 알고리즘이 같은 문제를 반복해서 푼다

Longest Common Subsequence(LCS)

- Subsequence Z = {z1 ~ zk} of sequence X = {x1 ~ xm} 단조 증가하는 X의 인덱스 시퀀스 {i1 ~ ik} such that xij zj가 있다

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

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

- Brute Force approach
  - LCS : Longest Common Subsequence
  - X의 모든 subsequence X'를 찾음(2^m개, m = X의 길이)
  - X'이 Y의 subsequence 인지 확인하고 가장 긴 것을 찾음
  - $O(n2^m)n$ = Y의 길이
- Common subsequence Z of X and Y : Z is subsequence of X, and of Y

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}.

→

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

1. X 와 Y 의 마지막 글자(x_m)가 같으면 (1) LCS Z의 마지막 글자 (z_m) 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.

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.

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 X과 Y_{n-1}의 common subsequence 이다. X과 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)

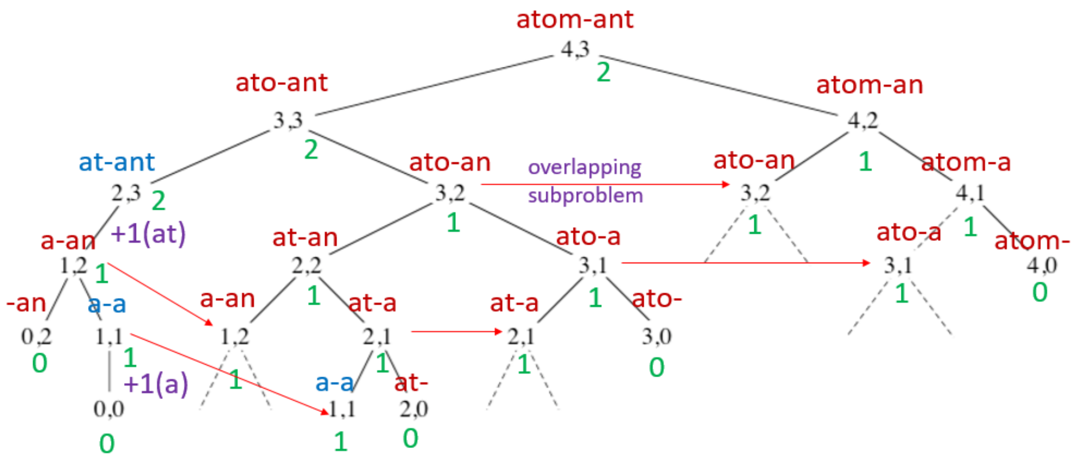
Let $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, \ldots, z_k \rangle$ be any LCS of X and Y.

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.

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




```
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 2/3

X_1(A)와 Y_4(BDCA)의 LCS 길이

b

c

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

- Constructing LCS (STEP 4)

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)
```

$= O(m+n)$

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

"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"

- Maximum subarray나 matrix multiplication을 dynamic programming으로 풀 수 있는가?