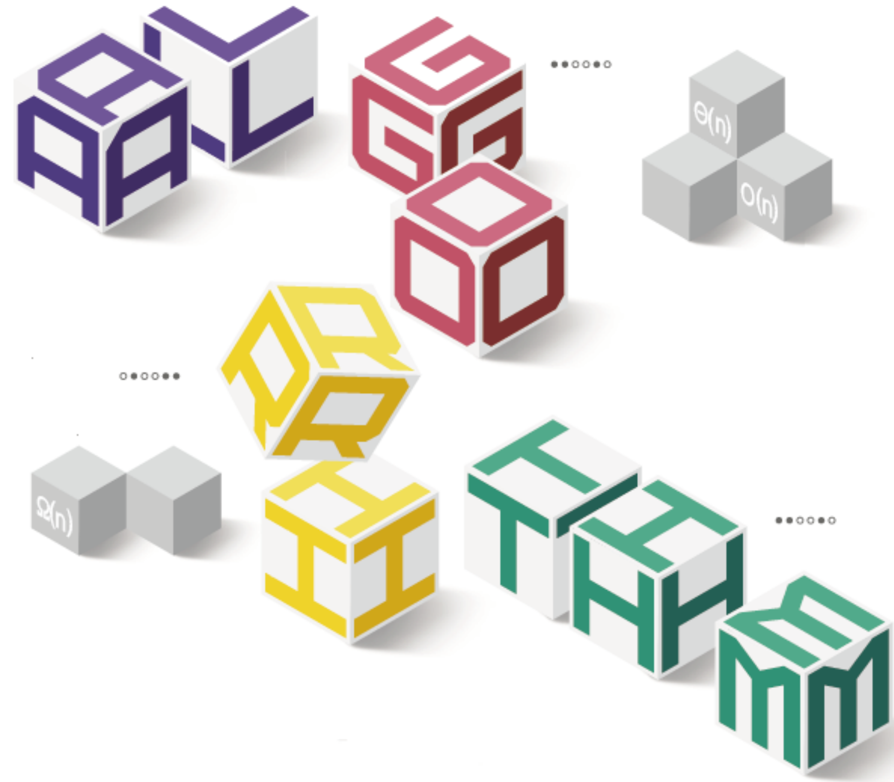


관계 중심의 사고법

쉽게 배우는 알고리즘

9장. 동적 프로그래밍 Dynamic Programming (DP)

9장. 동적 프로그래밍

Dynamic Programming (DP)

배경

- 재귀적 해법
  - 큰 문제에 닮음꼴의 작은 문제가 깃든다
  - 잘쓰면 보약, 잘못쓰면 맹독
    - 관계중심으로 파악함으로써 문제를 간명하게 볼 수 있다
    - 재귀적 해법을 사용하면 심한 중복 호출이 일어나는 경우가 있다

재귀적 해법의 빛과 그림자

- 재귀적 해법이 바람직한 예
  - 퀵정렬, 병합정렬 등의 정렬 알고리즘
  - 계승(factorial) 구하기
  - 그래프의 DFS
  - …
- 재귀적 해법이 치명적인 예
  - 피보나치수 구하기
  - 행렬곱셈 최적순서 구하기
  - …

도입문제: 피보나치수 구하기

- $f(n) = f(n\text{-}1) + f(n\text{-}2)$

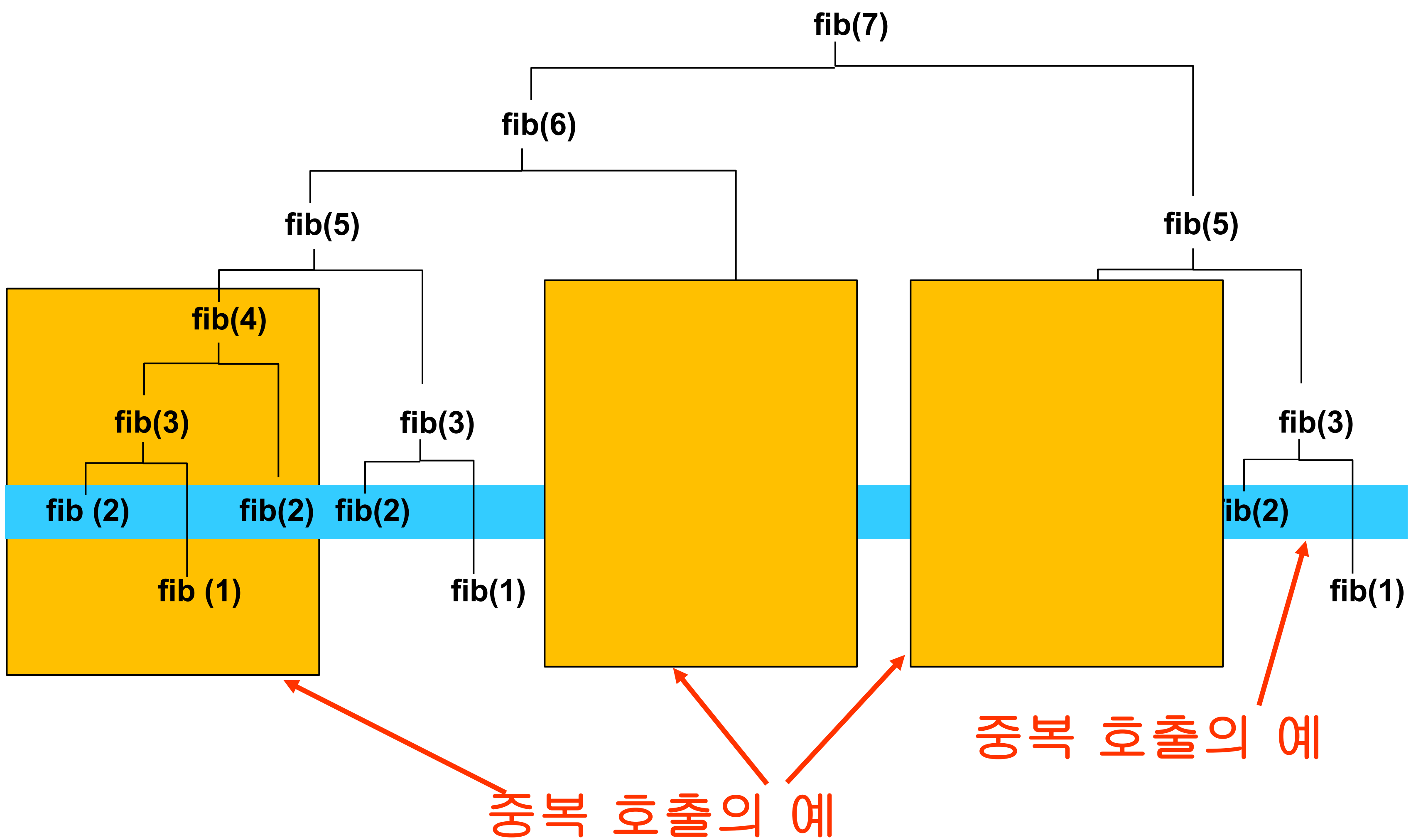
  $f(1) = f(2) = 1$
- 아주 간단한 문제지만
  - 동적 프로그래밍의 속성이 다 포함되어 있다

피보나치수를 구하는 재귀 알고리즘

```
fib(n)
{
    if (n = 1 or n = 2)
        then return 1;
        else return (fib(n-1) +fib(n-2));
}
```

✓ 엄청난 중복 호출이 존재한다

피보나치 수열의 호출 트리



피보나치수를 구하는 **DP** 알고리즘

```
fibonacci(n)
{
    f[1] ← f[2] ← 1;
    for i ← 3 to n
        f[i] ← f[i-1] +f[i-2];
    return f[n];
}
```

✓ $\Theta(n)$ 시간에 끝난다

Dynamic Programming의 적용 요건

- Optimal substructure (최적 부분구조)
  - 큰 문제의 최적 솔루션에 작은 문제의 최적 솔루션이 포함됨
- Overlapping recursive calls (재귀호출시 중복)
  - 재귀적 해법으로 풀면 같은 문제에 대한 재귀호출이 심하게 중복됨

➡ 동적 프로그래밍이 그 해결책!

문제예 1: 행렬 경로 문제

- 양수 원소들로 구성된 $n \times n$ 행렬이 주어지고, 행렬의 좌상단에서 시작하여 우하단까지 이동한다
- 이동 방법 (제약조건)
  - 오른쪽이나 아래쪽으로만 이동할 수 있다
  - 왼쪽, 위쪽, 대각선 이동은 허용하지 않는다
- 목표: 행렬의 좌상단에서 시작하여 우하단까지 이동하되, 방문한 칸에 있는 수들을 더한 값이 최소화되도록 한다

불법 이동의 예

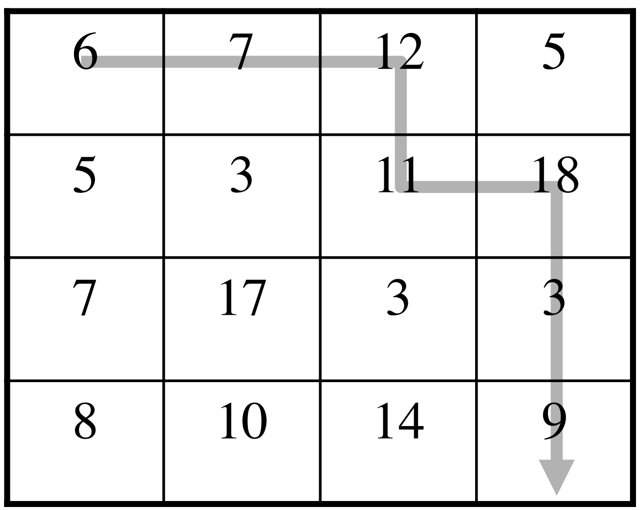
6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

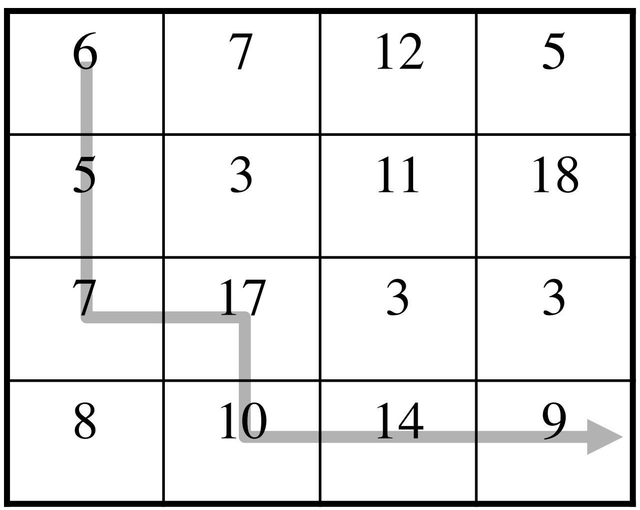
불법 이동 (상향)

6	7	12	5
5	3	11	18
7	17	3	3
8	10	14	9

불법 이동 (좌향)

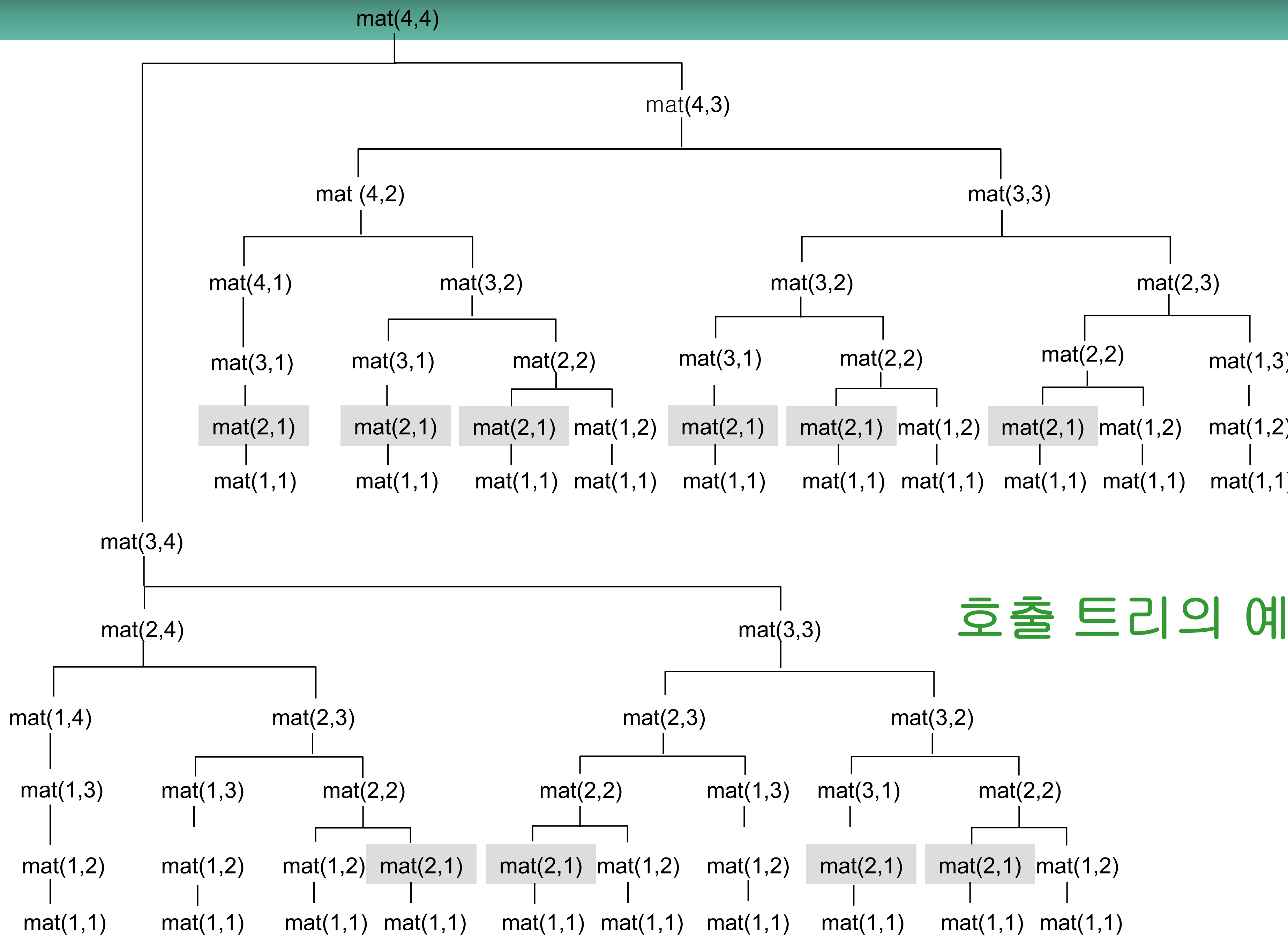
유효한 이동의 예





재귀 알고리즘

matrixPath(i, j)
▷ (i, j)에 이르는 최고점수
{
    if (i = 0 **or** j = 0) **then return** 0;
    else return (m_{ij} + (max(matrixPath(i-1, j), matrixPath(i, j-1))));
}


mat(4,4)
mat(4,3)
mat (4,2)
mat(3,3)
mat(4,1)
mat(3,2)
mat(3,2)
mat(2,3)
mat(3,1)
mat(3,1)
mat(2,2)
mat(3,1)
mat(2,2)
mat(2,2)
mat(1,3)
mat(2,1)
mat(2,1)
mat(2,1)
mat(1,2)
mat(2,1)
mat(2,1)
mat(1,2)
mat(2,1)
mat(1,2)
mat(1,2)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(3,4)
mat(2,4)
mat(3,3)
mat(1,4)
mat(2,3)
mat(2,3)
mat(3,2)
mat(1,3)
mat(1,3)
mat(2,2)
mat(2,2)
mat(1,3)
mat(3,1)
mat(2,2)
mat(1,2)
mat(1,2)
mat(1,2)
mat(2,1)
mat(2,1)
mat(1,2)
mat(1,2)
mat(2,1)
mat(2,1)
mat(1,2)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)
mat(1,1)

호출 트리의 예

DP 알고리즘

```
matrixPath(n)
▷ (n, n)에 이르는 최고점수
{
    for i ← 0 to n
        c[i, 0] ← 0;
    for j ← 1 to n
        c[0, j] ← 0;
    for i ← 1 to n
        for j ← 1 to n
            c[i, j] ← m_ij + max(c[i-1, j], c[i, j-1]);
    return c[n, n];
}
```

수행 시간: $\Theta(n^2)$

문제예 2: 돌 놓기

- $3 \times N$ 테이블의 각 칸에 양 또는 음의 정수가 기록되어 있다
- 돌을 놓는 방법 (제약조건)
  - 가로나 세로로 인접한 두 칸에 동시에 돌을 놓을 수 없다
  - 각 열에는 적어도 하나 이상의 돌을 놓는다
- 목표: 돌이 놓인 자리에 있는 수의 합을 최대가 되도록 조약돌 놓기

테이블의 예

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

합법적인 예

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

합법적이지 않은 예

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

Violation!

가능한 패턴

패턴 1:

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

패턴 2:

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

패턴 3:

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

패턴 4:

6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

임의의 열을 채울 수 있는 패턴은 4가지뿐이다

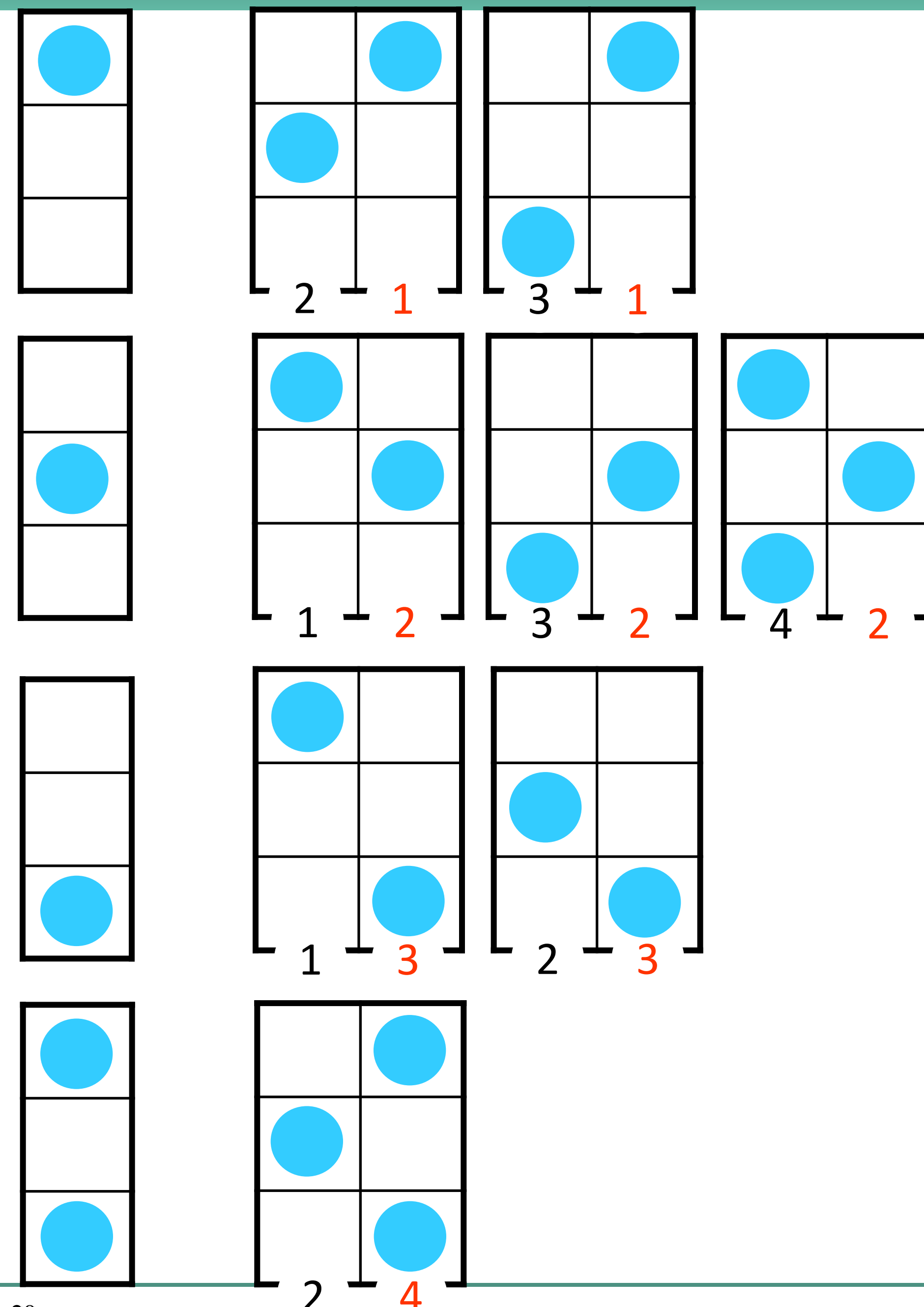
서로 양립할 수 있는 패턴들

패턴 1:

패턴 2:

패턴 3:

패턴 4:



패턴 1은 패턴 2, 3과
패턴 2는 패턴 1, 3, 4와
패턴 3은 패턴 1, 2와
패턴 4는 패턴 2와 양립할 수 있다

i열과 i-1열의 관계

	i-1	i			
	-5	5	3	11	3
...	9	7	13	8	5
	4	8	-2	9	4

i-1열이 패턴 1로 끝나거나

i-1열이 패턴 3으로 끝나거나

i-1열이 패턴 4로 끝나거나

재귀 알고리즘

```
pebble(i, p)
▷ i 열이 패턴 p로 놓일 때의 i 열까지의 최대 점수 합 구하기
▷ w[i, p] : i 열이 패턴 p로 놓일 때 i 열에 돌이 놓인 곳의 점수 합. p ∈{1, 2, 3, 4}
{
    if (i = 1)
        then return w[1, p] ;
        else {
            max ← -∞ ;
            for q ← 1 to 4 {
                if (패턴 q가 패턴 p와 양립)
                then {
                    tmp ← pebble(i—1, q) ;
                    if (tmp > max) then max ← tmp ;
                }
            }
            return (max + w[i, p]) ;
        }
}
```

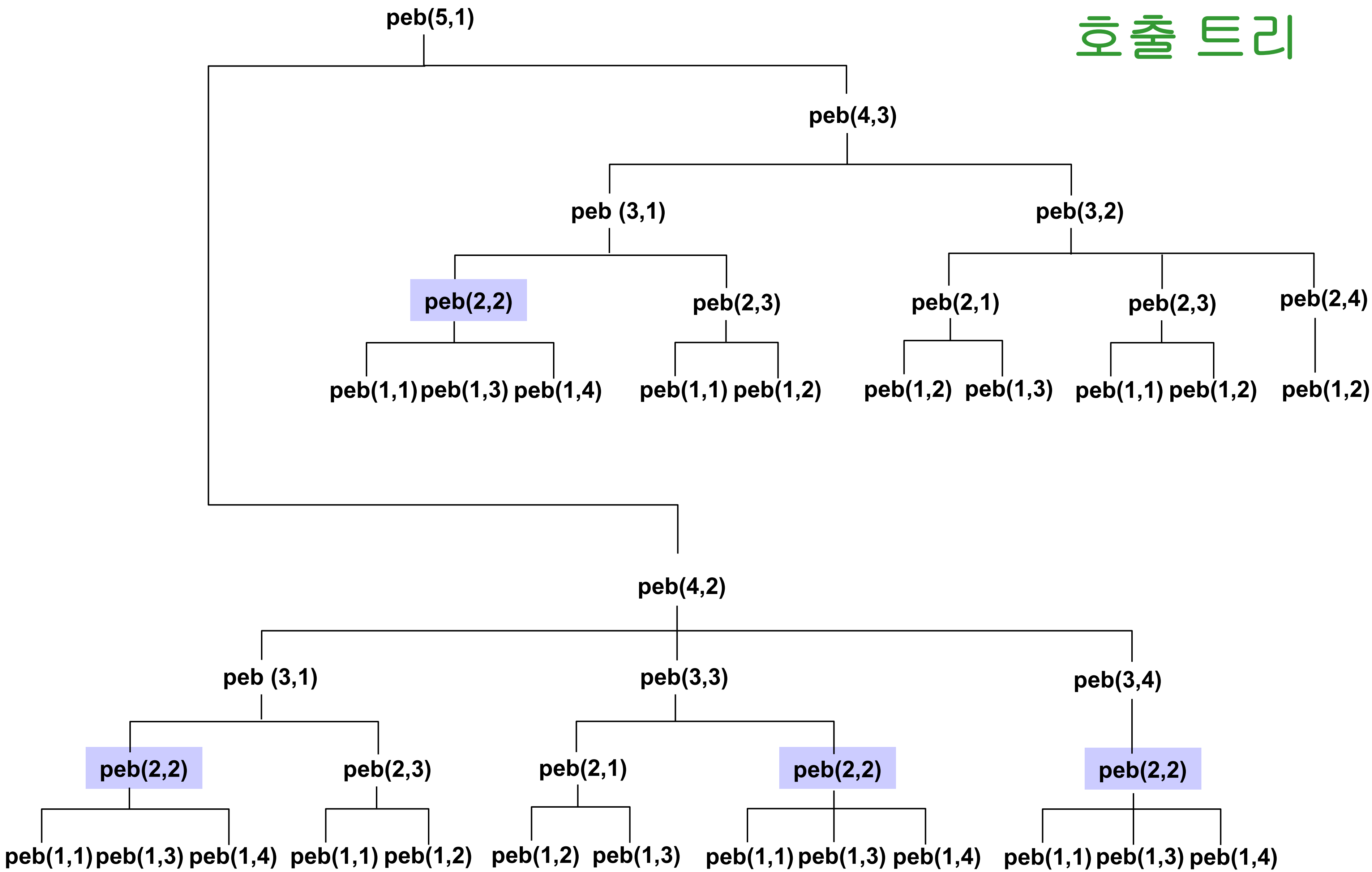
pebbleSum(n)

▷ n 열까지 조약돌을 놓은 방법 중 최대 점수 합 구하기

$$
\{
\quad \textbf{return} \ \max_{p=1,2,3,4} \{ \text{pebble}(n, p) \} ;
\}
$$

✓ pebble(*i*, *1*), …, pebble(*i*, *4*) 중 최대값이 최종적인 답

호출 트리


peb(5,1)
peb(4,3)
peb (3,1)
peb(3,2)
peb(2,2)
peb(2,3)
peb(2,1)
peb(2,3)
peb(2,4)
peb(1,1) peb(1,3) peb(1,4)
peb(1,1) peb(1,2)
peb(1,2) peb(1,3)
peb(1,1) peb(1,2)
peb(1,2)
peb(4,2)
peb (3,1)
peb(3,3)
peb(3,4)
peb(2,2)
peb(2,3)
peb(2,1)
peb(2,2)
peb(2,2)
peb(1,1) peb(1,3) peb(1,4)
peb(1,1) peb(1,2)
peb(1,2) peb(1,3)
peb(1,1) peb(1,3) peb(1,4)
peb(1,1) peb(1,3) peb(1,4)

DP 적용

- DP의 요건 만족
  - 최적 부분구조
    - pebble(i, .)에 pebble(i-1, .)이 포함됨
    - 즉, 큰 문제의 최적 솔루션에 작은 문제의 최적 솔루션이 포함됨
  - 재귀호출시 중복
    - 재귀적 알고리즘에 중복 호출 심함

pebble (n)
{
    for $p \leftarrow 1$ **to** 4
        $\text{peb}[1, p] \leftarrow \text{w}[1, p]$;
    for $i \leftarrow 2$ **to** n
        for $p \leftarrow 1$ **to** 4
            $\text{peb}[i, p] \leftarrow \max_{p\text{와 양립하는 패턴 } q} \{\text{peb}[i\text{-}1, q]\} + \text{w}[i, p]$;
    return $\max_{p=1,2,3,4} \{ \text{peb}[n, p] \}$;
}

✓복잡도 : $\Theta(n)$

복잡도 분석

기껏 4 바퀴

무시

pebble(n)
{

for $p \leftarrow 1$ **to** 4

$\quad\quad$ $\text{peb}[1, p] \leftarrow \text{w}[1, p]$;

기껏 n 바퀴

for $i \leftarrow 2$ **to** n

$\quad\quad$ **for** $p \leftarrow 1$ **to** 4

$\quad\quad\quad\quad$ $\text{peb}[i, p] \leftarrow \max\ \{\text{peb}[i\text{-}1, q]\}+\text{w}[i, p]$;

p와 양립하는 패턴 q

return $\max_{p=1,2,3,4} \{ \text{peb}[n, p] \}$;
}

기껏 3 가지

✓복잡도 : $\Theta(n)$

$n * 4 * 3 = \Theta(n)$

문제 예 3: 행렬 곱셈 순서

- 행렬 A, B, C
  - (AB)C = A(BC)
- 예: A:10 x 100, B:100 x 5, C:5 x 50
  - (AB)C: 7,500번의 곱셈 필요
  - A(BC): 75,000번의 곱셈 필요
- $A_1, A_2, A_3, \ldots, A_n$을 곱하는 최적의 순서는?
  - 총 n-1회의 행렬 곱셈을 어떤 순서로 할 것인가?

재귀적 관계

- 마지막 행렬 곱셈이 수행되는 상황
  - n-1가지 가능성
    - $A_1(A_2 \quad \ldots \quad A_n)$
    - $(A_1A_2)(A_3 \quad \ldots \quad A_n)$
    - $(A_1A_2A_3)(A_4 \ldots A_n)$
    - …
    - $(A_1 \ldots A_{n-2})(A_{n-1}A_n)$
    - $(A_1 \quad \ldots \quad A_{n-1})A_n$
  - 어느 경우가 가장 매력적인가?

- ✓ c_{ij}: 행렬 $A_i, \ldots, A_j$의 곱 $A_i \ldots A_j$를 계산하는 최소 비용
- ✓ A_k의 차원: $p_{k-1} \times p_k$

$$c_{ij} = \begin{cases} 0 & \text{if } i=j \\ \min\limits_{i \le k \le j-1} \{\underbrace{c_{ik} + c_{k+1,j} + p_{i-1}p_k p_j}\} & \text{if } i<j \end{cases}$$

일반형: $(A_1 \ldots A_k)\ (A_{k+1} \ldots A_n)$

재귀적 구현

rMatrixChain(i, j)
▷ 행렬곱 $A_i \ldots A_j$를 구하는 최소 비용 구하기
{
    if ($i = j$) **then return** 0;   ▷ 행렬이 하나뿐인 경우의 비용은 0
    min ← ∞;
    for $k \leftarrow i$ **to** j-1 {
        $q \leftarrow$ rMatrixChain(i, k) + rMatrixChain(k+1, j) + $p_{i-1}p_kp_j$;
        if ($q <$ min) **then** min ← q;
    }
    return min;
}

✓ 엄청난 중복 호출이 발생한다!

동적 프로그래밍

```
matrixChain(i, j)
{
      for i ← 1 to n
            m[i, i] ← 0;   ▷ 행렬이 하나뿐인 경우의 비용은 0
      for r ← 1 to n-1      ▷ r: 문제 크기를 결정하는 변수, 문제의 크기 = r+1
            for i ← 1 to n-r {
                  j ← i+r;
                  m[i, j] ← min{m[i, k] + m[k+1, j] + p_{i-1}p_k p_j};
                          i≤k≤j-1
            }
      return m[1, n];
}
```

$$m[i,j] \leftarrow \min_{i \le k \le j-1}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\};$$

✓ 복잡도: $\Theta(n^3)$

문제 예 4: 최장 공통 부분순서LCS

- 두 문자열에 공통적으로 들어있는 공통 부분순서 중 가장 긴 것을 찾는다
- 부분순서의 예
  - <bcdb>는 문자열 <abcbdab>의 부분순서다
- 공통 부분순서의 예
  - <bca>는 문자열 <abcbdab>와 <bdcaba>의 공통 부분순서다
- 최장 공통 부분순서longest common subsequence(LCS)
  - 공통 부분순서들 중 가장 긴 것
  - 예: <bcba>는 문자열 <abcbdab>와 <bdcaba>의 최장 공통 부분순서다

최적 부분구조

- 두 문자열 $X_m = <x_1 x_2 \ldots x_m>$과 $Y_n = <y_1 y_2 \ldots y_n>$에 대해
  - $x_m = y_n$이면
    X_m과 Y_n의 LCS의 길이는 X_{m-1}과 Y_{n-1}의 LCS의 길이보다 1이 크다
  - $x_m \neq y_n$이면
    X_m과 Y_n의 LCS의 길이는
    X_m과 Y_{n-1}의 LCS의 길이와 X_{m-1}과 Y_n의 LCS의 길이 중 큰 것과 같다

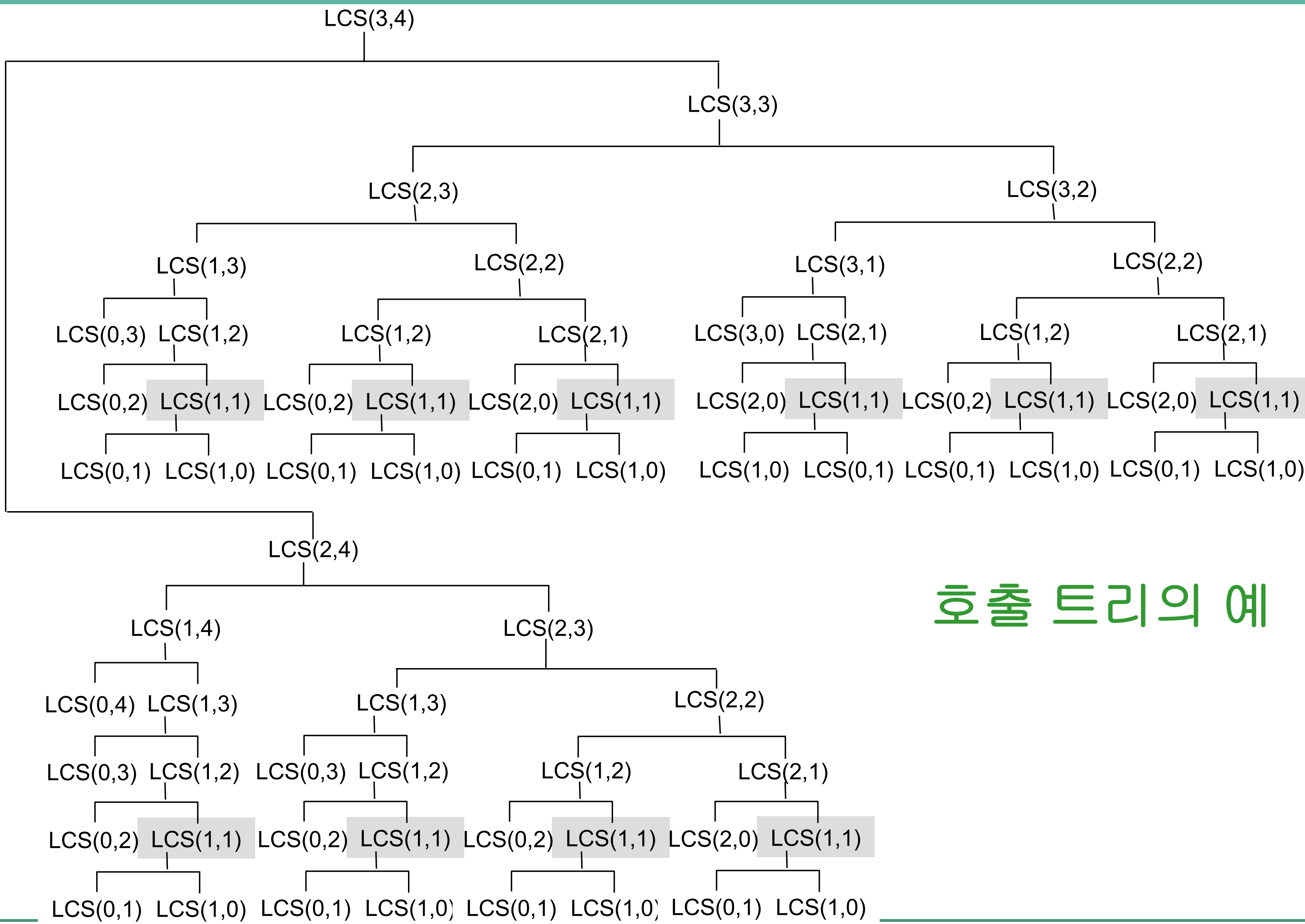
- $$c_{ij} = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c_{i-1,j-1} + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{c_{i-1,j}, c_{i,j-1}\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

✓ c_{ij} : 두 문자열 $X_i = <x_1 x_2 \ldots x_i>$과 $Y_j = <y_1 y_2 \ldots y_j>$의 LCS 길이

재귀적 구현

```
LCS(m, n)
▷ 두 문자열 X_m과 Y_n의 LCS 길이 구하기
{
    if (m = 0 or n = 0) then return 0;
    else if (x_m = y_n) then return LCS(m-1, n-1) + 1;
    else return max(LCS(m-1, n), LCS(m, n-1));
}
```

✓ 엄청난 중복 호출이 발생한다!


LCS(3,4)
LCS(3,3)
LCS(2,3) LCS(3,2)
LCS(1,3) LCS(2,2) LCS(3,1) LCS(2,2)
LCS(0,3) LCS(1,2) LCS(1,2) LCS(2,1) LCS(3,0) LCS(2,1) LCS(1,2) LCS(2,1)
LCS(0,2) LCS(1,1) LCS(0,2) LCS(1,1) LCS(2,0) LCS(1,1) LCS(2,0) LCS(1,1) LCS(0,2) LCS(1,1) LCS(2,0) LCS(1,1)
LCS(0,1) LCS(1,0) LCS(0,1) LCS(1,0) LCS(0,1) LCS(1,0) LCS(1,0) LCS(0,1) LCS(0,1) LCS(1,0) LCS(0,1) LCS(1,0)
LCS(2,4)
LCS(1,4) LCS(2,3)
LCS(0,4) LCS(1,3) LCS(1,3) LCS(2,2)
LCS(0,3) LCS(1,2) LCS(0,3) LCS(1,2) LCS(1,2) LCS(2,1)
LCS(0,2) LCS(1,1) LCS(0,2) LCS(1,1) LCS(0,2) LCS(1,1) LCS(2,0) LCS(1,1)
LCS(0,1) LCS(1,0) LCS(0,1) LCS(1,0) LCS(0,1) LCS(1,0) LCS(0,1) LCS(1,0)

호출 트리의 예

동적 프로그래밍

```
LCS(m, n)
▷ 두 문자열 X_m과 Y_n의 LCS 길이 구하기
{
      for i ← 0 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) then C[i, j] ← C[i-1, j-1] + 1;
                  else C[i, j] ← max(C[i-1, j], C[i, j-1]);
      return C[m, n];
}
```

✓ 복잡도: $\Theta(mn)$

문제 예 5: Optimal Binary Search Tree

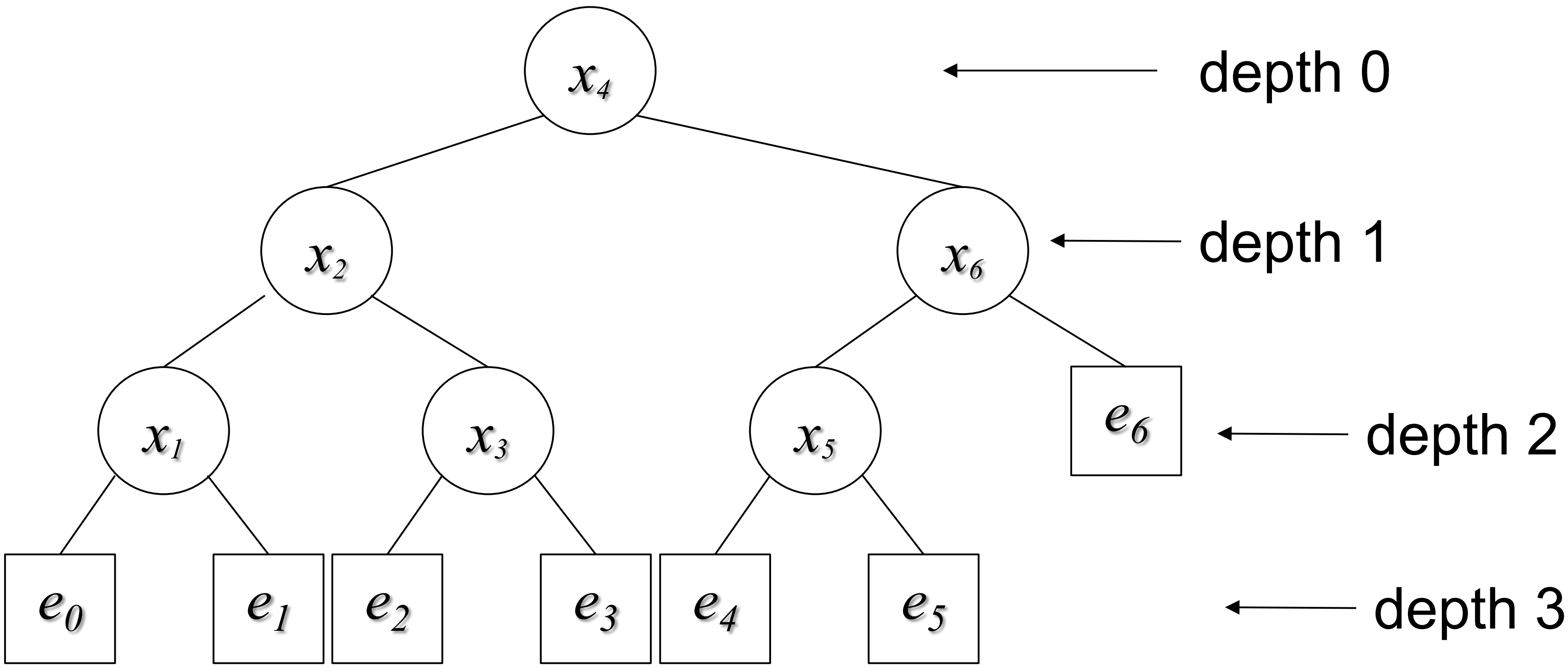
- Dynamic search tree vs. static search tree
  - Changing vs. fixed
- In the static case, we can find an optimal binary search tree
  - All the keys are given in advance

Given Condition

1. $S = \{x_1, x_2, \ldots, x_n\}$ where $x_1 < x_2 < \cdots < x_n$ (the set of keys)
2. p_i : the probability that search(S, x_i) is called $(i = 1,2, \ldots, n)$
3. q_i : the probability that search(S, x) is called for $x_i < x < x_{i+1}, i = 0,1, \ldots, n$ (let $x_0 = -\infty, x_{n+1} = \infty$ for boundary condition)

Object

Find a binary search tree
that has the minimum expected number of key comparisons



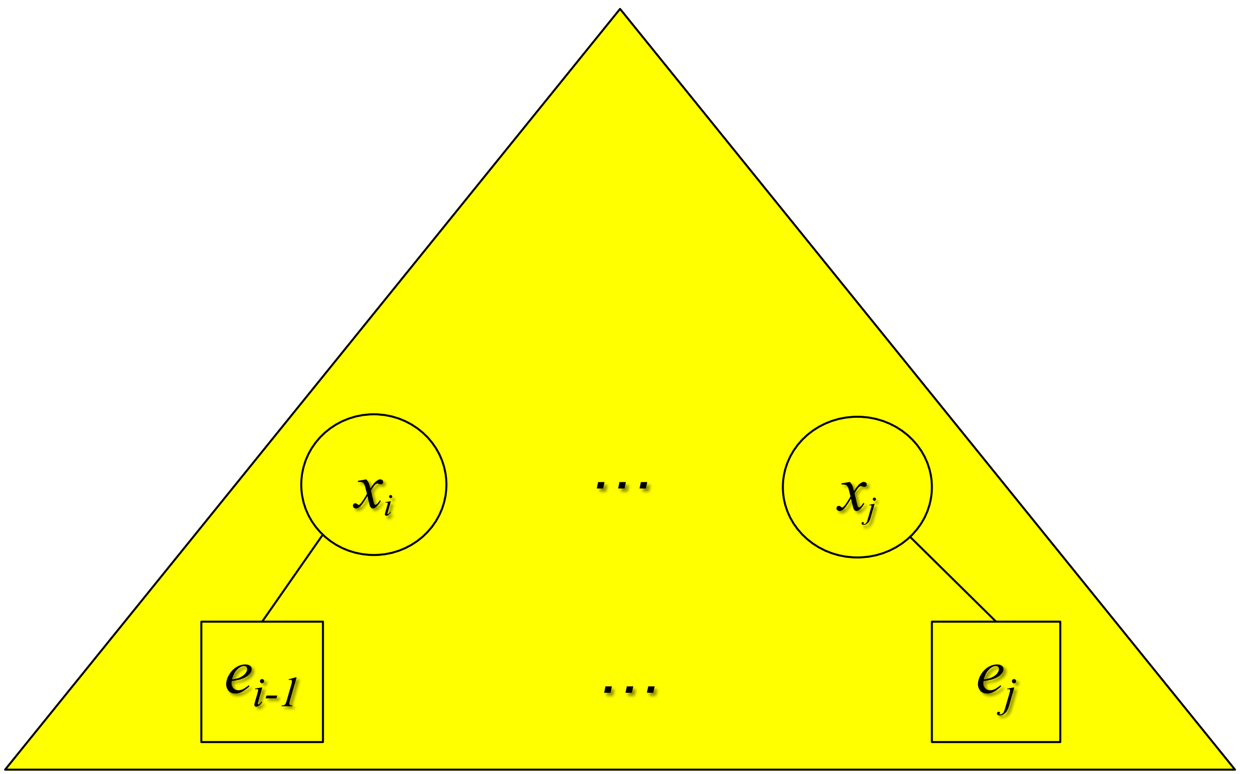
The cost of a b.s.t. with $x_1 < x_2 < \cdots < x_n$

$$= \sum_{i=1}^{n} p_i * (\text{depth}(x_i) + 1) + \sum_{i=0}^{n} q_i * \text{depth}(e_i)$$

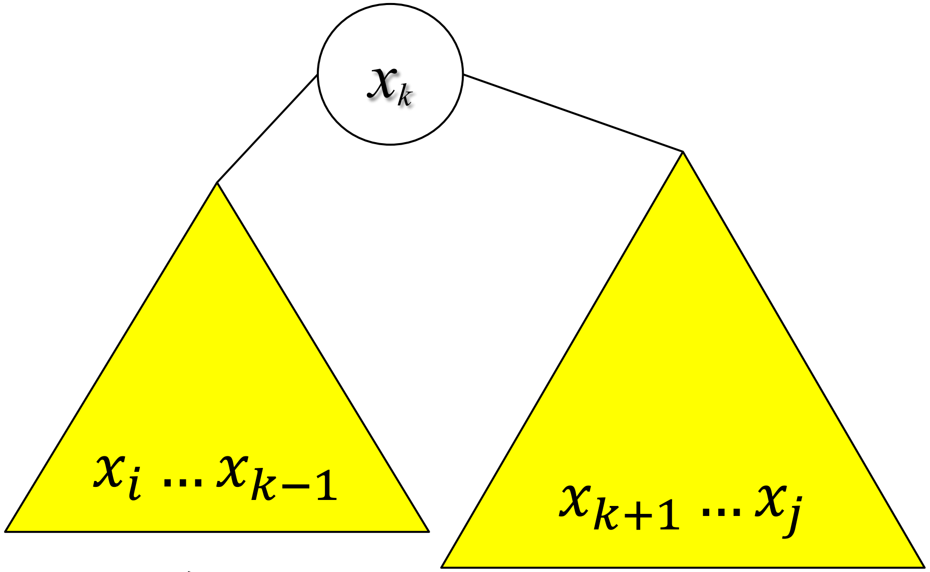
Consider the general case that we are going to optimize the set $\{x_i, \ldots, x_j\}$.

Let c_{ij} : the optimal cost for binary trees for $\{x_i, \ldots, x_j\}$ of prob. w_{ij}

w_{ij} : the probability of $x_{i-1} < x < x_{j+1}$ (i.e., $w_{ij} = \sum_{l=i-1}^{j} q_l + \sum_{l=i}^{j} p_l$)



Assume x_k is the root



$$c_{ij} = (c_{i,k-1} + 1 \cdot w_{i,k-1}) + (c_{k+1,j} + 1 \cdot w_{k+1,j}) + 1 \cdot p_k$$

$$= c_{i,k-1} + c_{k+1,j} + w_{ij}$$

Optimal substructure

$$c_{ij} = \begin{cases} q_j & \text{if } j = i-1 \\ \min\limits_{k=i,\dots,j} \left(c_{i,k-1} + c_{k+1,j}\right) + w_{ij} & \text{if } i \leq j \end{cases}$$

수행 시간

$$\min_{k=i,\ldots,j}\left(c_{i,k-1} + c_{k+1,j}\right) + w_{ij}$$

For c_{ij}, we look at $\underbrace{j - i + 1}_{=\, m}$ cases each taking constant time

let

$$\sum_{m=1}^{n} (n - m + 1) \cdot \Theta(m)$$

$$= \sum_{m=1}^{n} \Theta(nm - m^2 + m)$$

$$= \Theta\Big(\sum_{i=1}^{n} (nm - m^2 + 1)\Big)$$

$$= \Theta(n^3)$$

optional

문제 예 6: 최단경로

- Weighted digraph $G=(V, E)$
  - w_{ij} : vertex i에서 vertex j에 이르는 edge의 길이
    - Edge가 없으면 ∞
- 목표
  - 시작점 s에서 다른 각 정점vertex에 이르는 최단거리를 모두 구한다

- d_t^k : 중간에 최대 k 개의 edge를 거쳐 s로부터 vertex t에 이르는 최단거리
- 목표: d_t^{n-1}
- Note! For $t \neq s$,
  - $d_t^0 = \infty$
  - $d_t^1 = w_{s,t}$

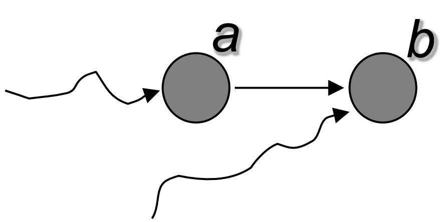
다음 페이지로 넘어가기 전에
무엇을 중심으로 관계를 파악할 지
스스로 생각해보자

재귀적 관계

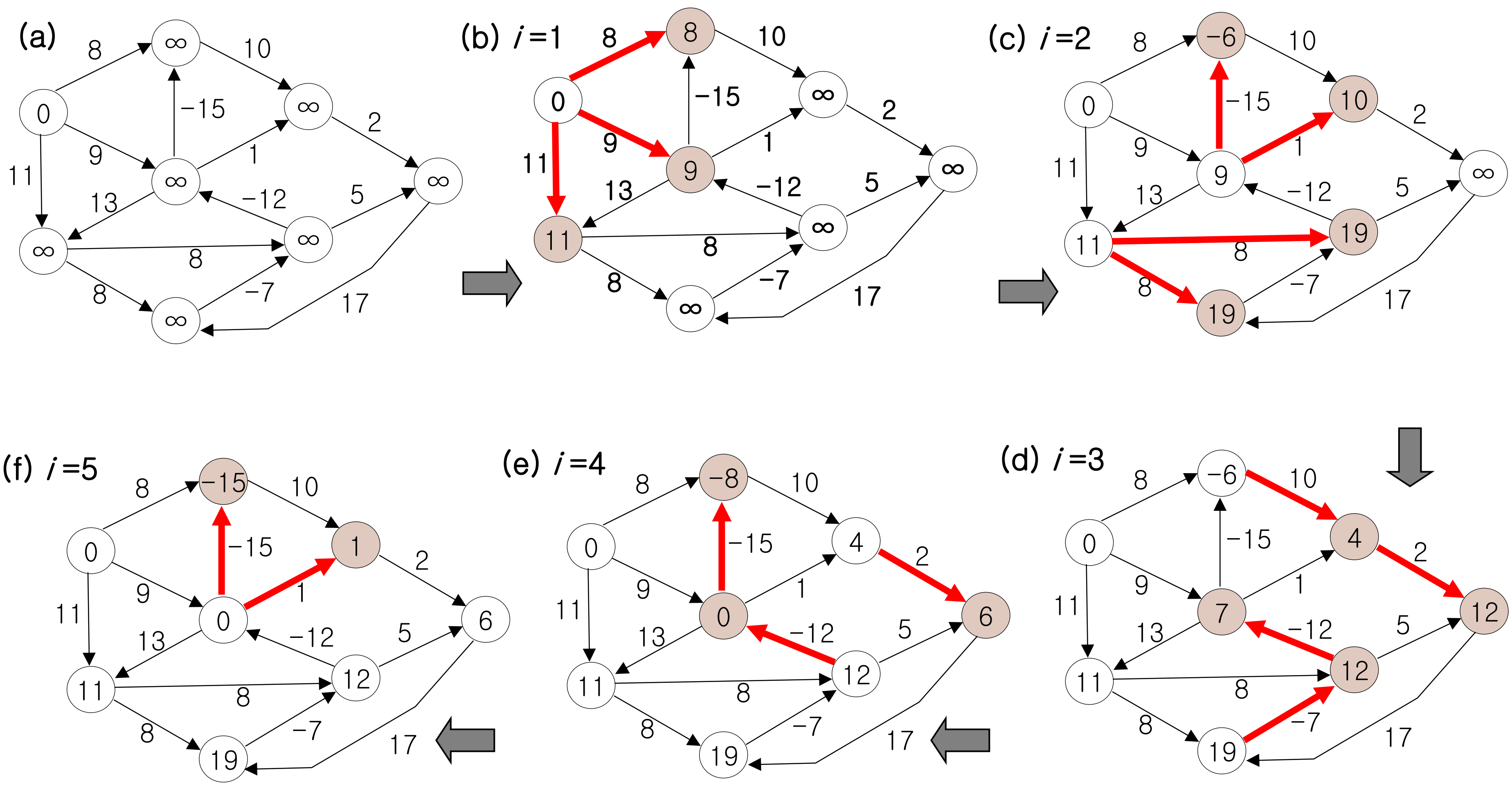
$$
\begin{cases}
d_t^{k} = \min\limits_{\text{for all edges } (r,\, t)} \{ d_r^{k-1} + w_{rt} \} \\
d_s^{0} = 0; \\
d_t^{0} = \infty;
\end{cases}
$$

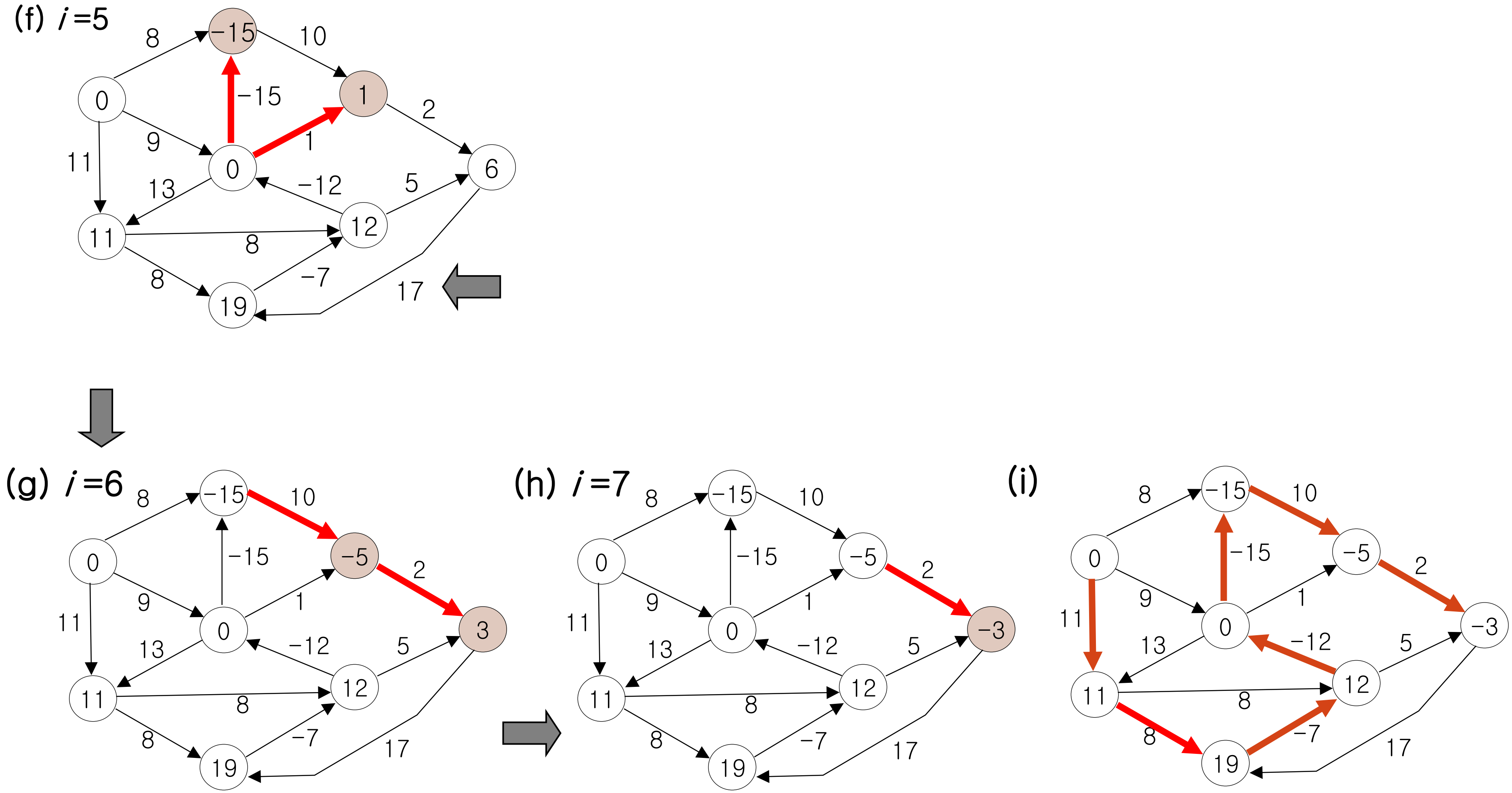
DP 알고리즘

Ballman-Ford(G, s)
{
    $d_s \leftarrow 0$;
    for all vertices $i \neq s$
        $d_i \leftarrow \infty$;
    for $k \leftarrow 1$ **to** n-1 {
        for all edges (a, b) {
            if $(d_a + w_{ab} < d_b)$ **then** $d_b \leftarrow d_a + w_{ab}$;
        }
    }
}



✓ Propagation 되는 모습이 떠오르면 잘 이해한 것!


(a)
(b) i=1
(c) i=2
(d) i=3
(e) i=4
(f) i=5


(f) i=5
(g) i=6
(h) i=7
(i)