Dynamic Programming

Background

Recursive structure

- A problem contains the same problems of smaller size(s)
- Bless if used suitably, fatal if abused
  - Problems become simple, looking at them in a relationship-based view
  - Good candidate for recursive algorithms. But…
  - Recursive algorithms are sometimes fatal, due to overlapping call

Duality of Recursive Solutions

- Good examples
  - Quicksort, mergesort, …
  - Factorial
  - DFS
  - …
- Fatal examples
  - Fibonacci numbers
  - Sequence of matrix multiplication
  - …

An Introductory Problem

Fibonacci Sequence

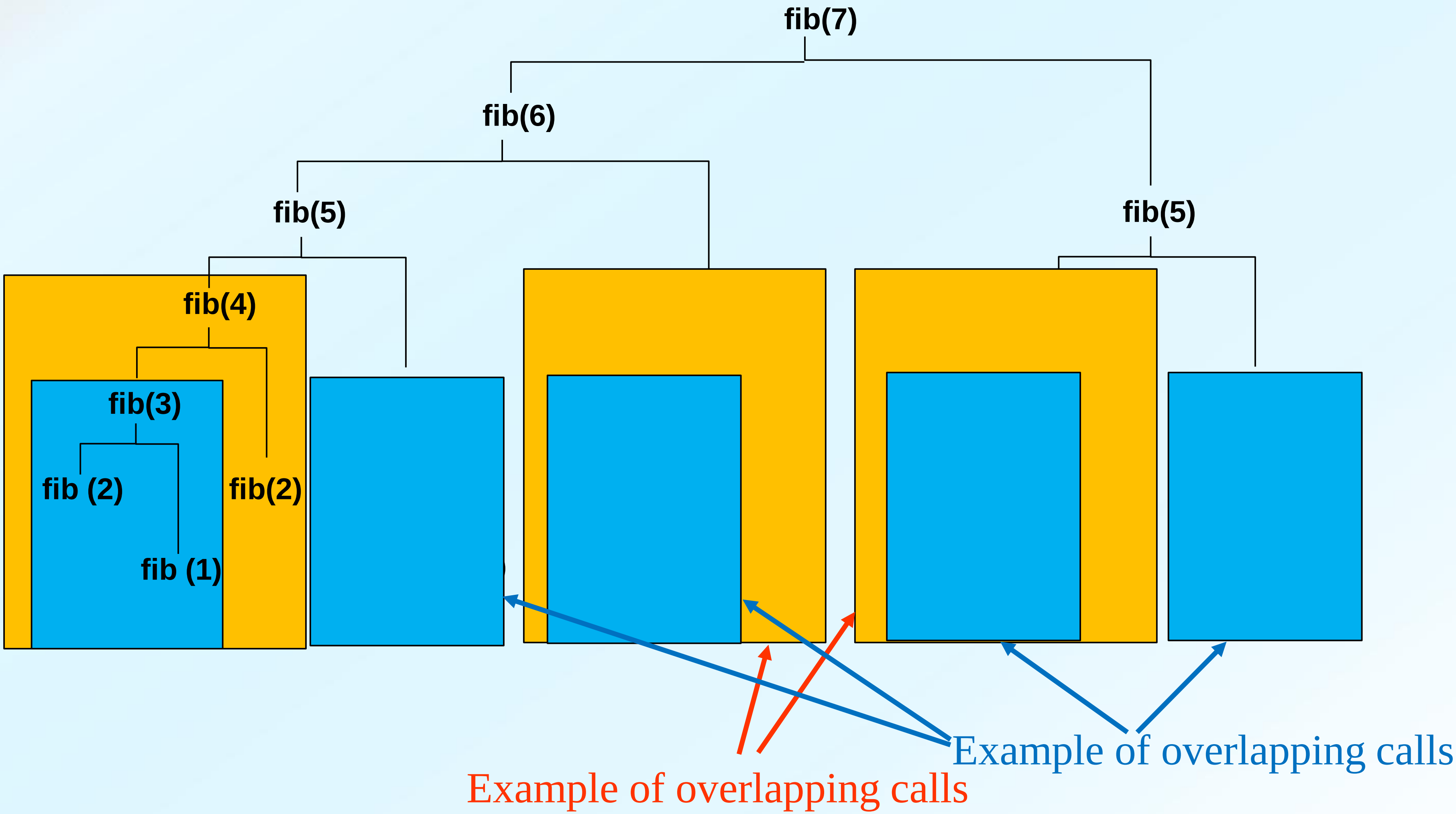
- $f(n) = f(n\text{-}1) + f(n\text{-}2)$
  $f(1) = f(2) = 1$
- A simple problem, but..
  - Contains all features for dynamic programming

Recursive Algorithm

```
fib(n):
        if (n = 1 or n = 2) return 1
        else return (fib(n-1) +fib(n-2))
```

✓ 낭비적인 중복 호출이 어마어마하다

Call Tree



fib(7)
fib(6)
fib(5)
fib(5)
fib(4)
fib(3)
fib (2)
fib(2)
fib (1)
Example of overlapping calls
Example of overlapping calls

재귀적 fib(100)은 얼마나 걸릴까?

내 데스크 탑 PC: Pentium 3GHz

fib(50) – 36초

fib(66) – 하루 정도

fib(100) – 3만5천년 정도

fib(136) – 1조년 초과

지수함수적 중복 호출로 인해 이런 치명적인 비효율이 발생한다

A Dynamic Programming Algorithm

fibonacci(n):

$f_1 \leftarrow f_2 \leftarrow 1$

for $i \leftarrow 3$ **to** n

$f_i \leftarrow f_{i-1} + f_{i-2}$

return f_n

✓ Complete in $\Theta(n)$ time

fib(n):

if ($n = 1$ **or** $n = 2$) **return** 1

else return (**fib**(n-1) +**fib**(n-2))

✓ $\Omega(2^{\frac{n}{2}})$

Conditions of Dynamic Programming

- Optimal substructure 최적 부분구조
  - An optimal solution contains optimal solutions of smaller problems
- Overlapping recursive calls 재귀호출시 중복
  - A recursive algorithm undergoes enormous overlapping calls

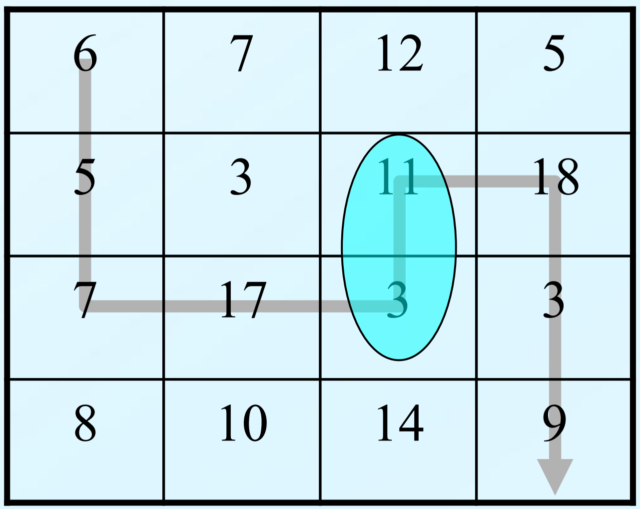
➡ Dynamic Programming is a resolution!

Problem 1: Paths in Matrix

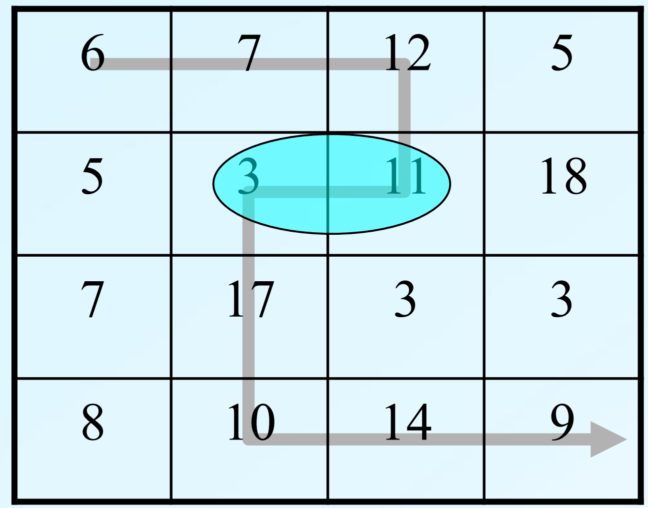
- Given an $n \times n$ matrix of positive numbers, we move from position (1, 1) to position (n, n)
- Rules
  - Only right or downward moving is allowed
  - Left, upward, diagonal movings are not allowed
- Object:

  Find the maximal sum of numbers out of all possible paths

Examples of illegal Moves

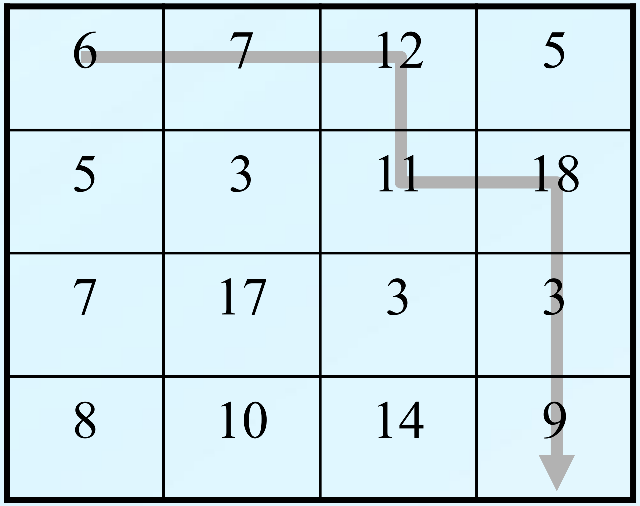


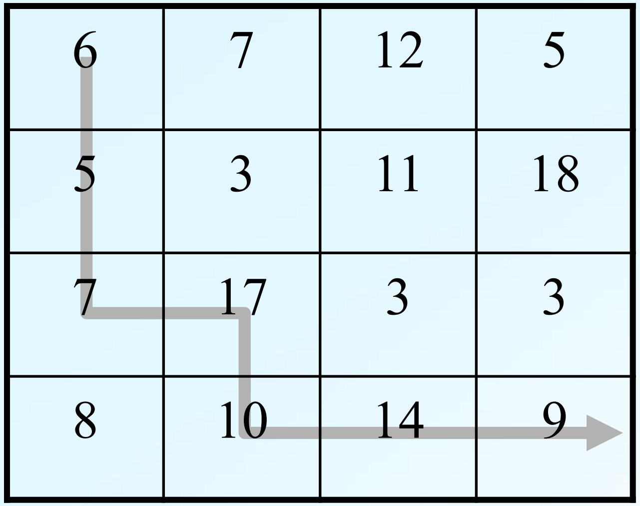
Upward move



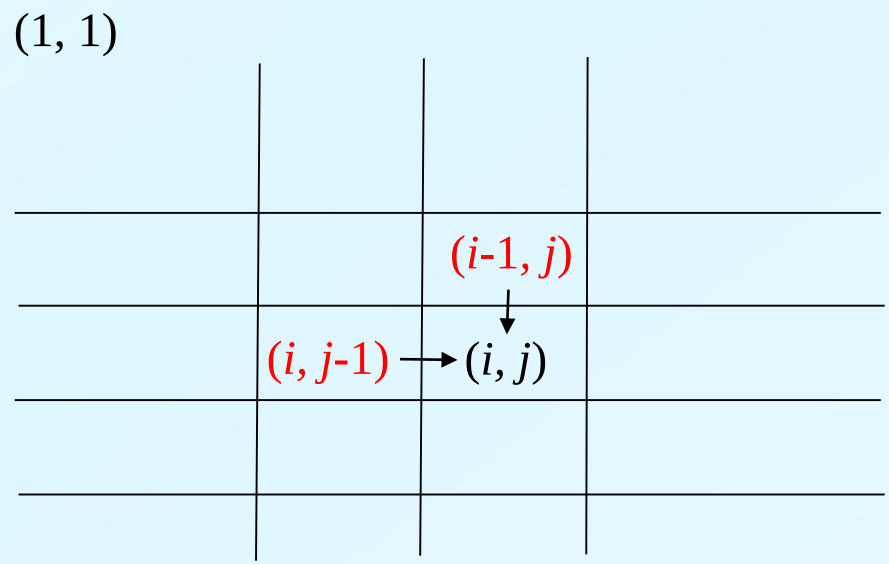
Left move

Example of Legal Paths





Optimal Substructure



✓ There are just two immediately previous slot to (i, j)

$$c_{ij} = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ m_{ij} + \max\{c_{i,j-1}, c_{i-1,j}\} & \text{, otherwise} \end{cases}$$

where

c_{ij}: (1, 1)에서 (i, j)에 이르는 최대 점수

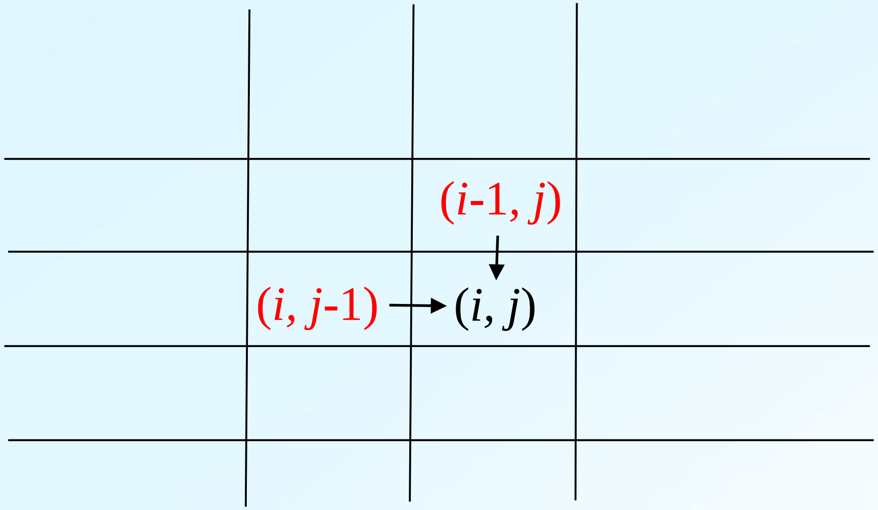
m_{ij}: (i, j)에 있는 값

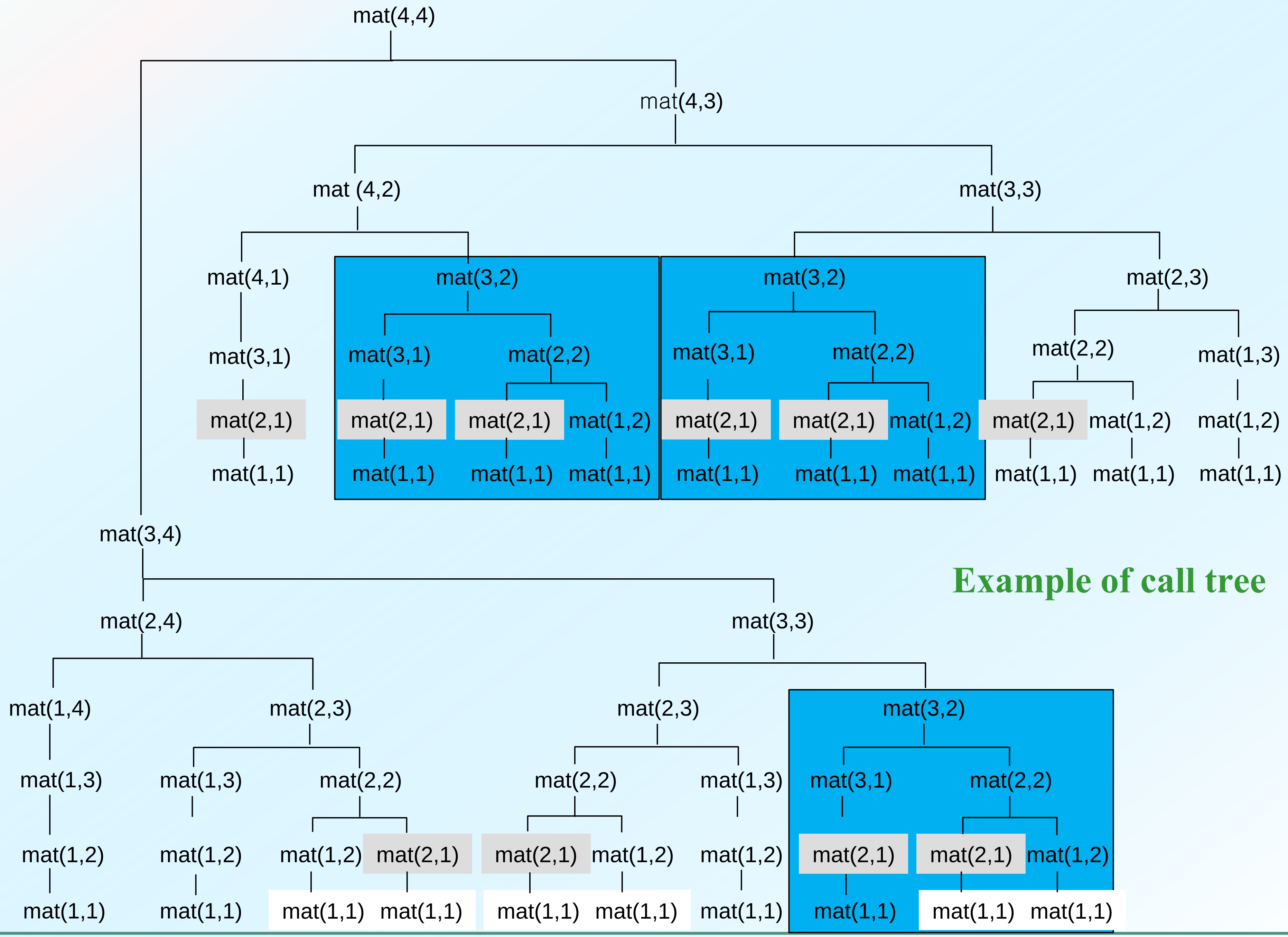
A Recursive Algorithm

matrixPath(i, j):
◀ (1, 1)에서 (i, j)에 이르는 최대 점수 찾기

if ($i = 0$ **or** $j = 0$) **return** 0

else return (m_{ij} + (max(matrixPath(i-1, j), matrixPath(i, j-1))))





Example of call tree

중복 호출이 증가하는 모습

수행되는 matrixPath()	matrixPath(2,1)의 중복 호출 횟수
matrixPath(2,2)	1
matrixPath(3,3)	3
matrixPath(4,4)	10
matrixPath(5,5)	35
matrixPath(6,6)	126
matrixPath(7,7)	462
matrixPath(8,8)	1,716
matrixPath(9,9)	6,435

Applying DP

- Satisfies conditions for DP
  - Optimal substructure
    - c_{ij} includes $c_{i,j-1}$ and $c_{i-1,j}$
    - An optimal solution contains optimal solutions of smaller problems
  - Overlapping recursive calls
    - A recursive algorithm undergoes enormous overlapping calls

DP Algorithm

$$c_{ij} = \begin{cases} 0 & \text{, if } i = 0 \text{ or } j = 0 \\ m_{ij} + \max\{c_{i,j-1}, c_{i-1,j}\} & \text{, otherwise} \end{cases}$$

matrixPath(n):

◀ Find maximal point of paths to (n, n)

> **for** $i \leftarrow 0$ **to** n
>
> $\quad c_{i,0} \leftarrow 0$
>
> **for** $j \leftarrow 1$ **to** n
>
> $\quad c_{0,j} \leftarrow 0$
>
> **for** $i \leftarrow 1$ **to** n
>
> $\quad$ **for** $j \leftarrow 1$ **to** n
>
> $\quad\quad c_{i,j} \leftarrow m_{i,j} + \max(c_{i-1,j}, c_{i,j-1})$
>
> **return** $c_{n,n}$

Running time: $\Theta(n^2)$

Problem 2: Placing Stones

- A number(either positive or negative) on each slot of a $3 \times N$ table
- Rules
  - Any two adjacent slots(horizontally or vertically) cannot both have stones
  - There should be at least one stone on each column
- Objective:

  Find the maximal sum of numbers out of all possible stone placements

Example Table

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

Legal Example

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

illegal Example

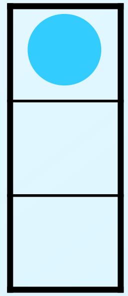
6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

Violation!

Possible patterns

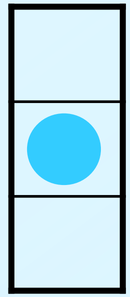
Only 4 patterns possible for a column

Pattern 1:



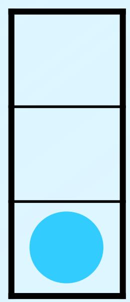
6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

Pattern 2:



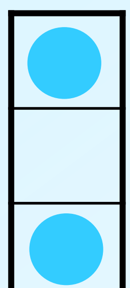
6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

Pattern 3:



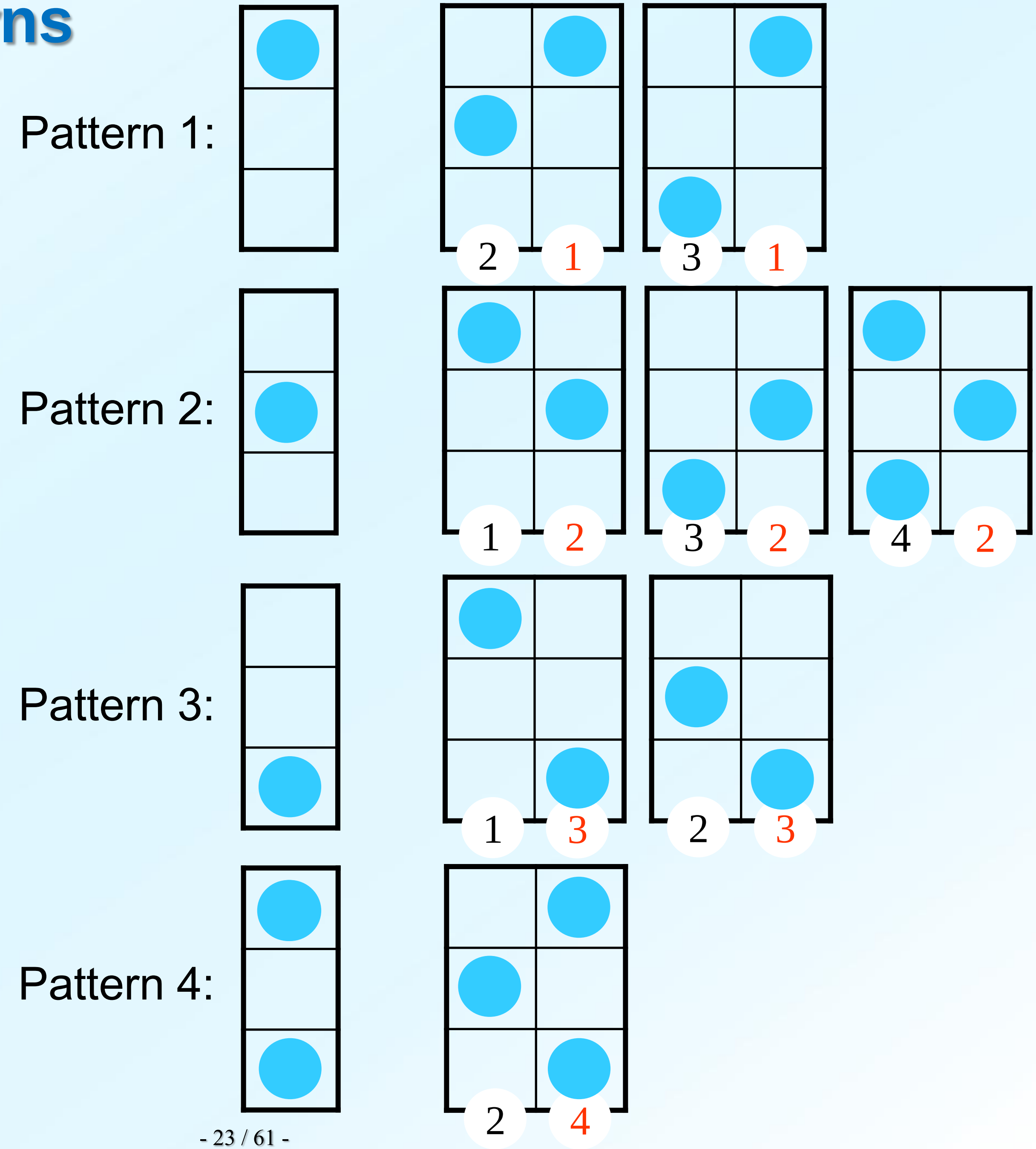
6	7	12	-5	5	3	11	3
-8	10	14	9	7	13	8	5
11	12	7	4	8	-2	9	4

Pattern 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

Compatible Patterns



Pattern 1: patterns 2 and 3
Pattern 2: patterns 1, 3, and 4
Pattern 3: patterns 1 and 2
Pattern 4: pattern 2

Relationship between columns *i* and *i*-1

If column *i* is covered by pattern 2

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

Column *i*-1 is covered by pattern 1

by pattern 3

or by pattern 4

Optimal Substructure

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

$$c_{ip} = \begin{cases} w_{ip} & ,\text{if } i = 1 \\ w_{ip} + \max\limits_{q \text{ compatible to pattern } p} c_{i-1,q} & ,\text{if } i > 1 \end{cases}$$

where

c_{ip}: maximal sum when column i covered by pattern p

w_{ip}: point sum of stones of column i when column i is covered by pattern p, $p \in \{1, 2, 3, 4\}$

Recursive Algorithm

$$c_{ip} = \begin{cases} w_{ip} & \text{, if } i = 1 \\ w_{ip} + \max\limits_{q \text{ compatible to pattern } p} c_{i-1,q} & \text{, if } i > 1 \end{cases}$$

```
pebble(i, p):
◀ maximal sum with column i covered by pattern p
◀ w[i, p]: point sum of stones of column i when column i is covered by pattern p. p ∈{1, 2, 3, 4}
    if (i = 1)
        return w[1, p]
    else
            max ← -∞
            for q ← 1 to 4
                if (pattern q compatible to pattern p)
                    tmp ← pebble(i-1, q)
                    if (tmp > max)
                        max ← tmp
            return (max + w[i, p])
```

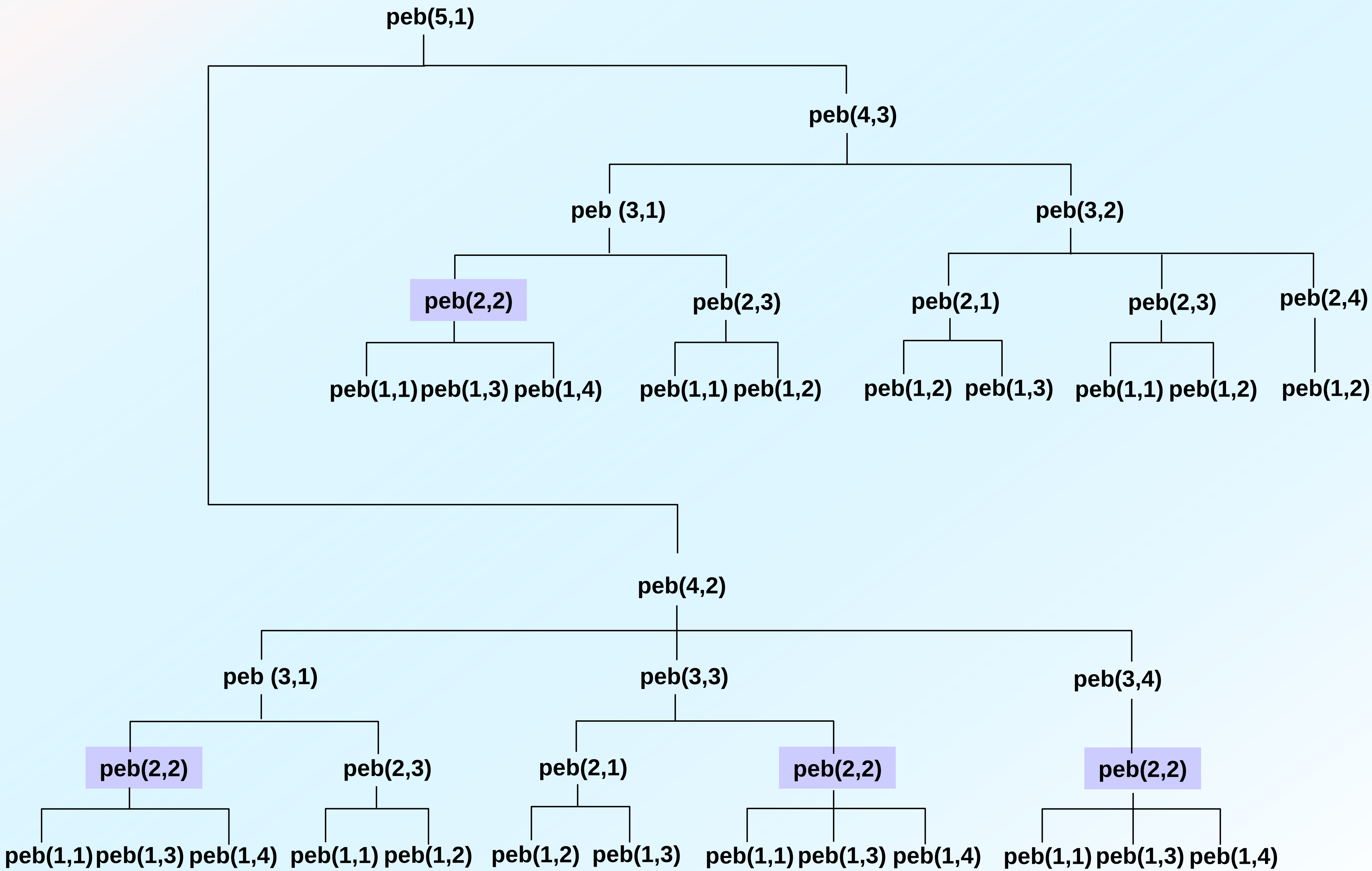
pebbleSum(n):

◀ maximal sum after all column are covered with pebbles

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

✓ Final solution: max of pebble(i, 1), …, pebble(i, 4)

Call Tree



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)

중복 호출이 증가하는 모습

문제의 크기	Subproblem의 총 수	함수 pebble()의 총 호출 횟수
1	4	4
2	8	12
3	12	30
4	16	68
5	20	152
6	24	332
7	28	726

Applying DP

- Satisfying conditions for DP
  - Optimal substructure
    - pebble(i, .) includes pebble(i-1, .)
    - An optimal solution contains optimal solutions of smaller problems
  - Overlapping recursive calls
    - A recursive algorithm undergoes enormous overlapping calls

DP Algorithm

pebble(n):

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

        $\text{peb}_{1,p} \leftarrow w_{1,p}$

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

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

            $\text{peb}_{i,p} \leftarrow \max\limits_{q \text{ compatible to } p} \{\text{peb}_{i\text{-}1,q}\} + w_{i,p}$

    return $\max\limits_{p=1,2,3,4} \{ \text{peb}_{n,p} \}$

✓Complexity: $\Theta(n)$

Complexity Analysis

at most 4 iterations

ignore

at most n iterations

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

    $\text{peb}[i, p] \leftarrow \max_{q \text{ compatible to } p} \{\text{peb}[i\text{-}1, q]\} + \text{w}[i, p]$

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

at most 3 cases

✓Complexity: $\Theta(n)$

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

Problem 3: Matrix-Chain Multiplication

$A: p \times q$ $B: q \times r$	⇨	Cost of multiplication AB: pqr

Matrix multiplication is transitive

- (AB)C = A(BC)

For A:10 x 100, B:100 x 5, C:5 x 50

- (AB)C: cost = 10 x 100 x 5 + 10 x 5 x 50 = 7,500
  (7,500 scalar multiplications)
- A(BC): cost = 100 x 5 x 50 + 10 x 100 x 50 = 75,000
  (75,000 scalar multiplications)

Objective:

- Find the minimal cost to compute $A_1A_2A_3\ldots A_n$
- How to compute n-1 matrix multiplications in total?

Recursive Structure

- The situation right before the last multiplication
  - There are n-1 possibilities
    - $A_1(A_2 \quad \ldots \quad A_n)$
    - $(A_1A_2)(A_3 \quad \ldots \quad A_n)$
    - $(A_1A_2A_3)(A_4 \ldots A_n)$
    - …
    - $(A_1 \ldots A_{n-2})(A_{n-1}A_n)$
    - $(A_1 \quad \ldots \quad A_{n-1})A_n$
  - Which is the least costly?

General Form

- ✓ c_{ij}: the minimal cost to compute $A_i \ldots A_j$
- ✓ Dimension of A_k : $p_{k-1} \times p_k$

$$c_{1n} = \begin{cases} 0 & \text{if } n=1 \\ \underbrace{\min_{1 \le k \le n-1} \{c_{1k} + c_{k+1,j} + p_0 p_k p_n\}} & \text{if } 1<n \end{cases}$$

$A_1(A_2 \quad \ldots \quad A_n)$

$(A_1A_2)(A_3 \quad \ldots \quad A_n)$

$\ldots$

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

$\ldots$

$(A_1 \ldots A_{n-2})(A_{n-1}A_n)$

$(A_1 \quad \ldots \quad A_{n-1})A_n$

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

Further Generalization

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

General form: $(A_i \ldots A_k)\ (A_{k+1} \ldots A_j)$

$A_i(A_{i+1} \quad \ldots \quad A_j)$

$\ldots$

$\boxed{(A_i \ldots A_k)(A_{k+1} \ldots A_j)}$

$\ldots$

$(A_i \quad \ldots \quad A_{j-1})A_j$

Recursive Algorithm

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

rMatrixChain(i, j):
◀ minimal cost to compute $A_i \ldots A_j$
  if $(i = j)$ **return** 0 ◀ singleton
  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

✓ Tremendous overlapping calls!

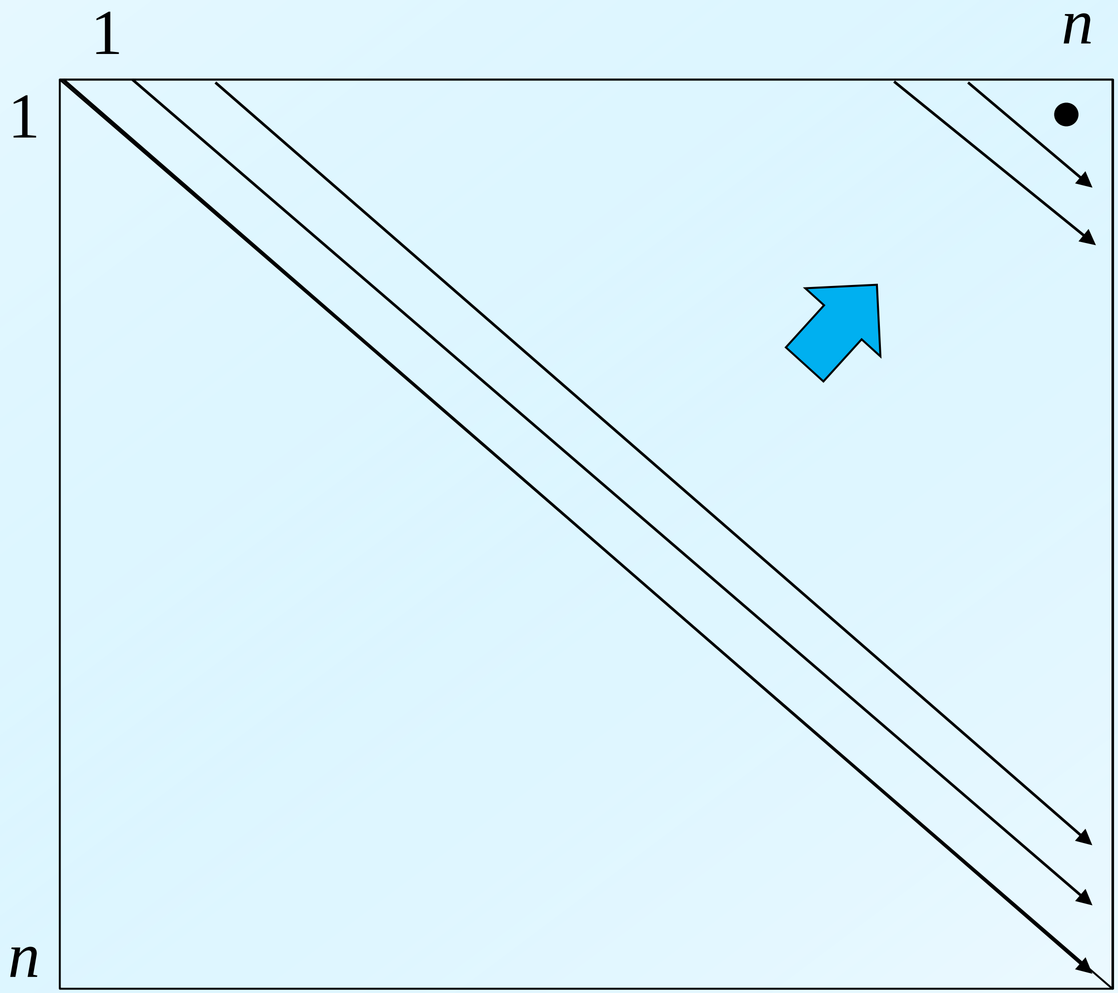
DP

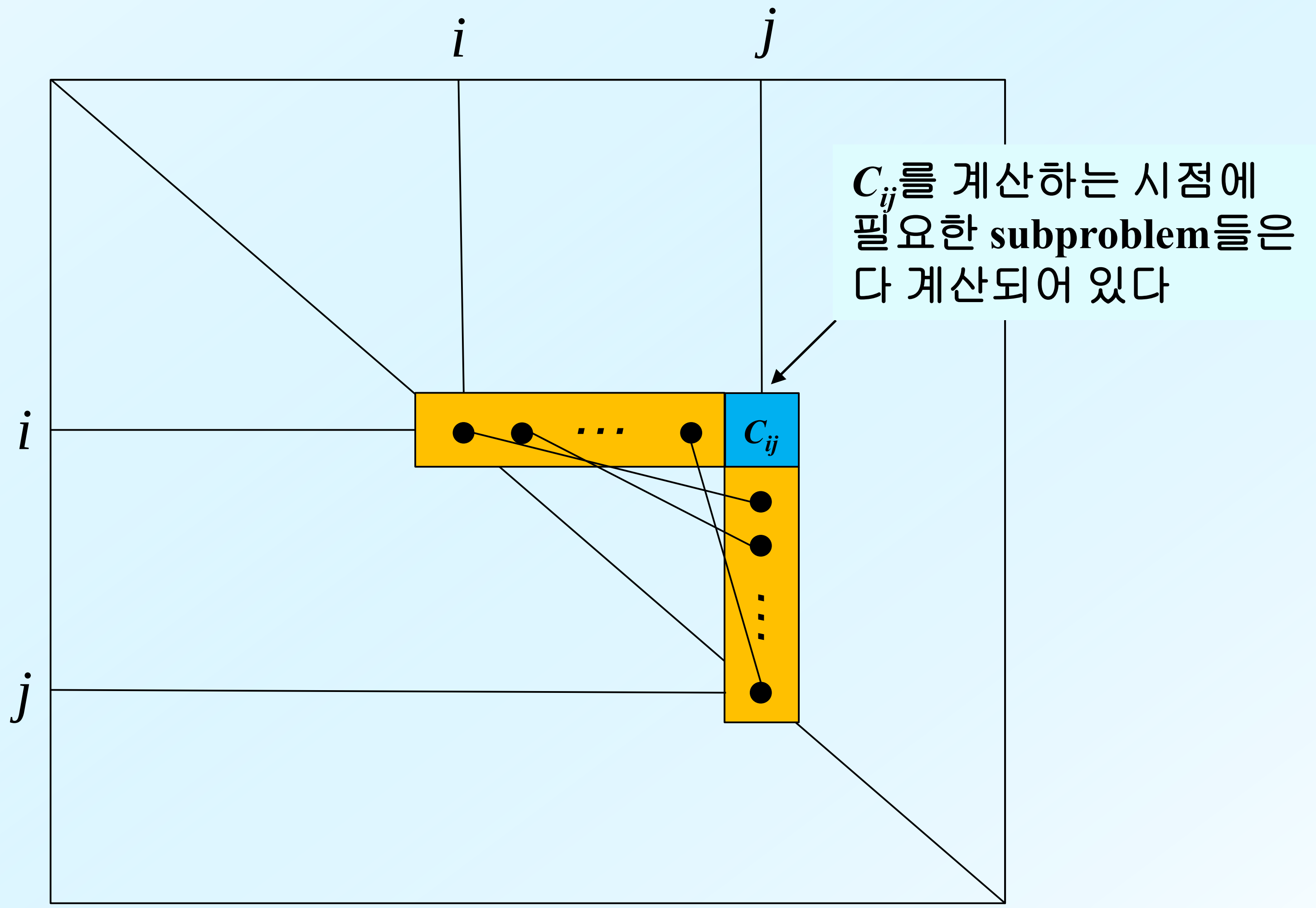
matrixChain(i, j):

```
for i ← 1 to n
        c_{i,i} ← 0    ◀ singleton case: cost 0
for r ← 1 to n-1    ◀ r+1: the problem size
        for i ← 1 to n-r
                j ← i+r
                c_{i,j} ← min_{i ≤ k ≤ j-1} {c_{i,k} + c_{k+1,j} + p_{i-1} p_k p_j}
return c_{1,n}
```

$$c_{i,j} \leftarrow \min_{i \le k \le j-1}\{c_{i,k} + c_{k+1,j} + p_{i-1}p_k p_j\}$$

✓ Complexity: $\Theta(n^3)$


1
n
1
n


i
j
i
j
Cij
Cij를 계산하는 시점에
필요한 subproblem들은
다 계산되어 있다

Problem 4: Longest Common Subsequence

Subsequence

- Example: <bcdb> is a subsequence of <abcbdab>

Common subsequence

- Example: <bca> is a common subsequence of <abcbdab> and <bdcaba>

Longest common subsequence[LCS]

- The longest among the common subsequences
- Example:
- <bcba> is the longest common subsequence of <abcbdab> and <bdcaba>

Objective: Given two strings,
find the longest common subsequence of them

Optimal Substructure

X_m : $x_1 x_2 x_3 \;\dots \quad \dots \quad \dots\; x_{m-1} x_m$

Y_n : $y_1 y_2 y_3 \;\dots \quad \dots \quad \dots\; y_{n-1} y_n$

Case 1: $x_m = y_n$

X_{m-1}: $x_1 x_2 x_3 \;\dots \quad \dots \quad \dots\; x_{m-1}$ x_m

Y_{n-1} : $y_1 y_2 y_3 \;\dots \quad \dots \quad \dots\; y_{n-1}$ y_n

LCS of X_m and Y_n = "LCS of $X_{m\text{-}1}$ and $Y_{n\text{-}1}$" + 1

* LCS: the length of LCS for convenience

Optimal Substructure

Case 2: $x_m \neq y_n$

X_{m-1}: $x_1 x_2 x_3 \;\dots \quad \dots \quad \dots\; x_{m-1}$ x_m

Y_n : $y_1 y_2 y_3 \;\dots \quad \dots \quad \dots\; y_{n-1}\; y_n$

LCS of X_{m-1} and Y_n

LCS of X_m and Y_n = 둘 중 큰 것

LCS of X_m and Y_{n-1}

X_m: $x_1 x_2 x_3 \;\dots \quad \dots \quad \dots\; x_{m-1}\; x_m$

Y_{n-1} : $y_1 y_2 y_3 \;\dots \quad \dots \quad \dots\; y_{n-1}$ y_n

Optimal Substructure

For two strings $X_m = <x_1x_2 \ldots x_m>$ and $Y_n = <y_1y_2 \ldots y_n>$

Case $x_m = y_n$:

LCS of X_m and Y_n = LCS of $X_{m\text{-}1}$ and $Y_{n\text{-}1}$ + 1

Case $x_m \neq y_n$:

LCS of X_m and Y_n = max{LCS of X_m and $Y_{n\text{-}1}$, LCS of $X_{m\text{-}1}$and Y_n의 LCS

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

✓ c_{ij} : LCS length of $X_i = <x_1x_2 \ldots x_i>$ and $Y_j = <y_1y_2 \ldots y_j>$

c_{mn} : final solution

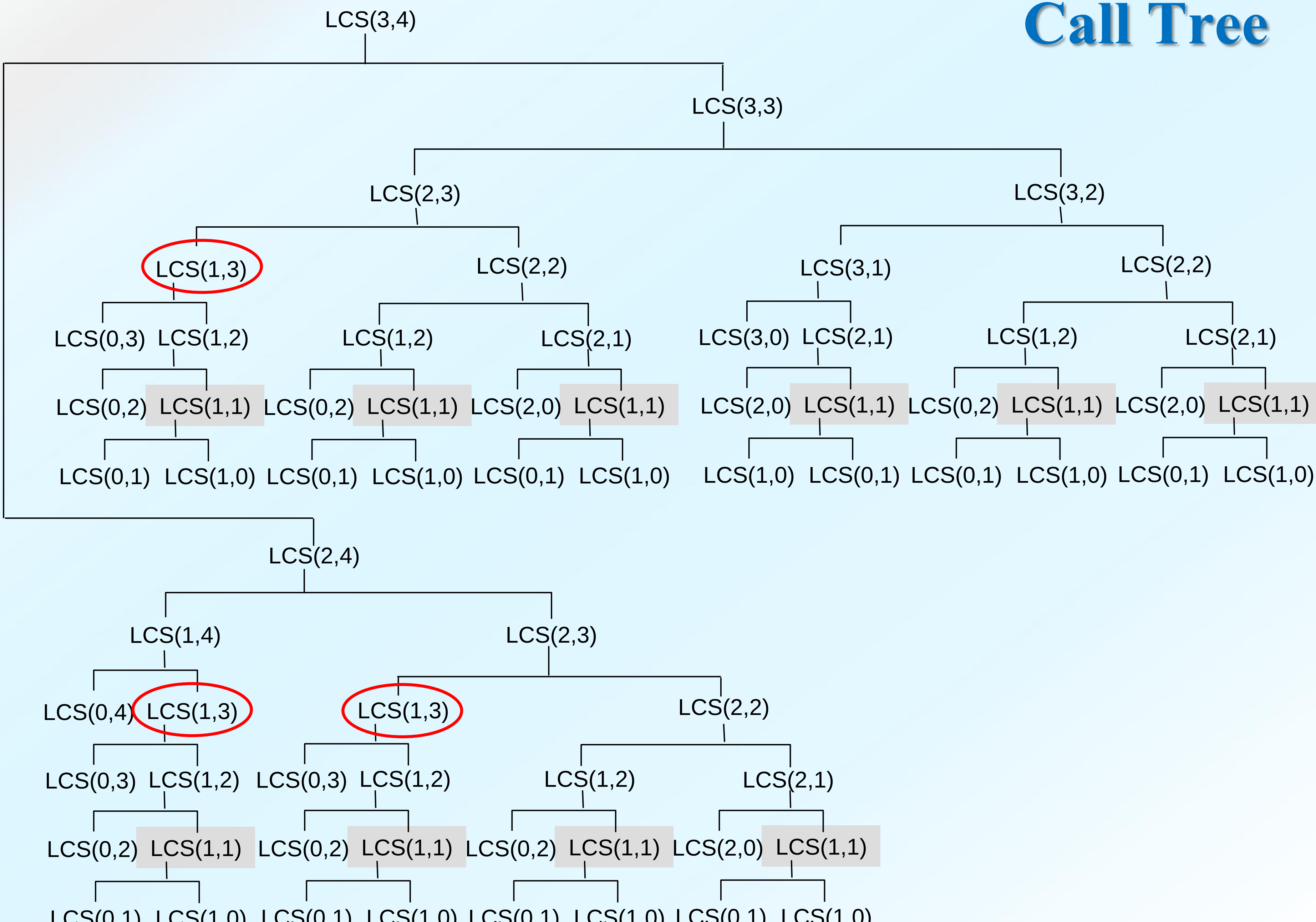
Recursive Algorithm

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

```
LCS(m, n):
◀ LCS length of X_m and Y_n
    if (m = 0 or n = 0) return 0
    else if (x_m = y_n) return LCS(m-1, n-1) + 1
    else return max(LCS(m-1, n), LCS(m, n-1))
```

✓ Tremendous overlapping calls!

Call Tree


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)

DP

LCS(m, n):
    for $i \leftarrow 0$ **to** m
        $C_{i,0} \leftarrow 0$
    for $j \leftarrow 0$ **to** n
        $C_{0,j} \leftarrow 0$
    for $i \leftarrow 1$ **to** m
        for $j \leftarrow 1$ **to** n
            if $(x_i = y_j)$ $C_{i,j} \leftarrow C_{i-1,j-1} + 1$
            else $C_{i,j} \leftarrow \max(C_{i-1,j}, C_{i,j-1})$
    return $C_{m,n}$

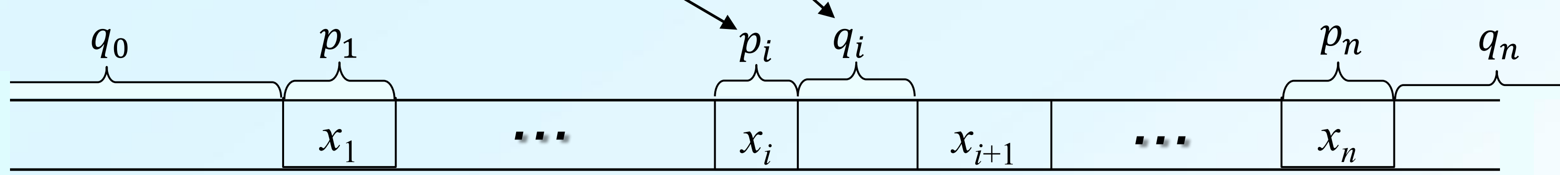
✓ Complexity: $\Theta(mn)$

Problem 5: Optimal Binary Search Tree

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

Given Condition

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

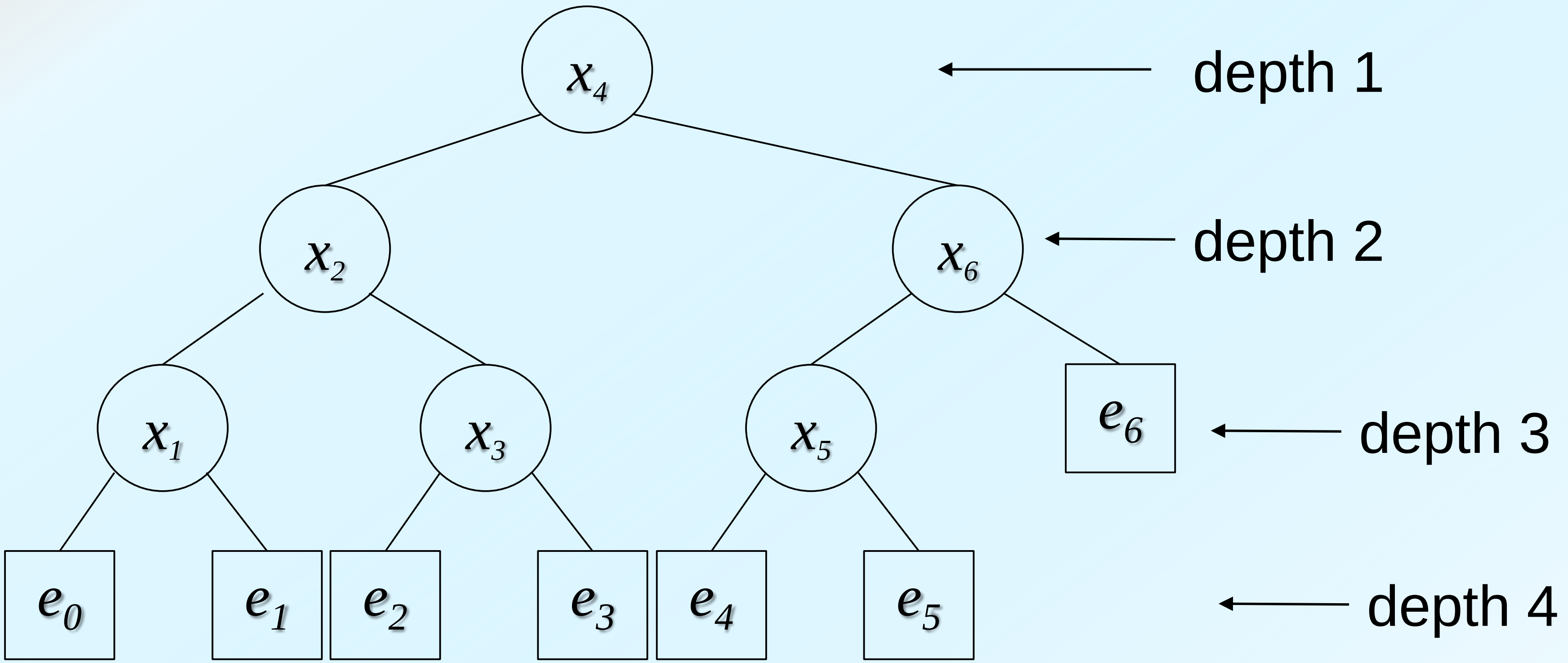


Object

Find a binary search tree

that has the minimum expected number of key comparisons

An Example B.S.T.



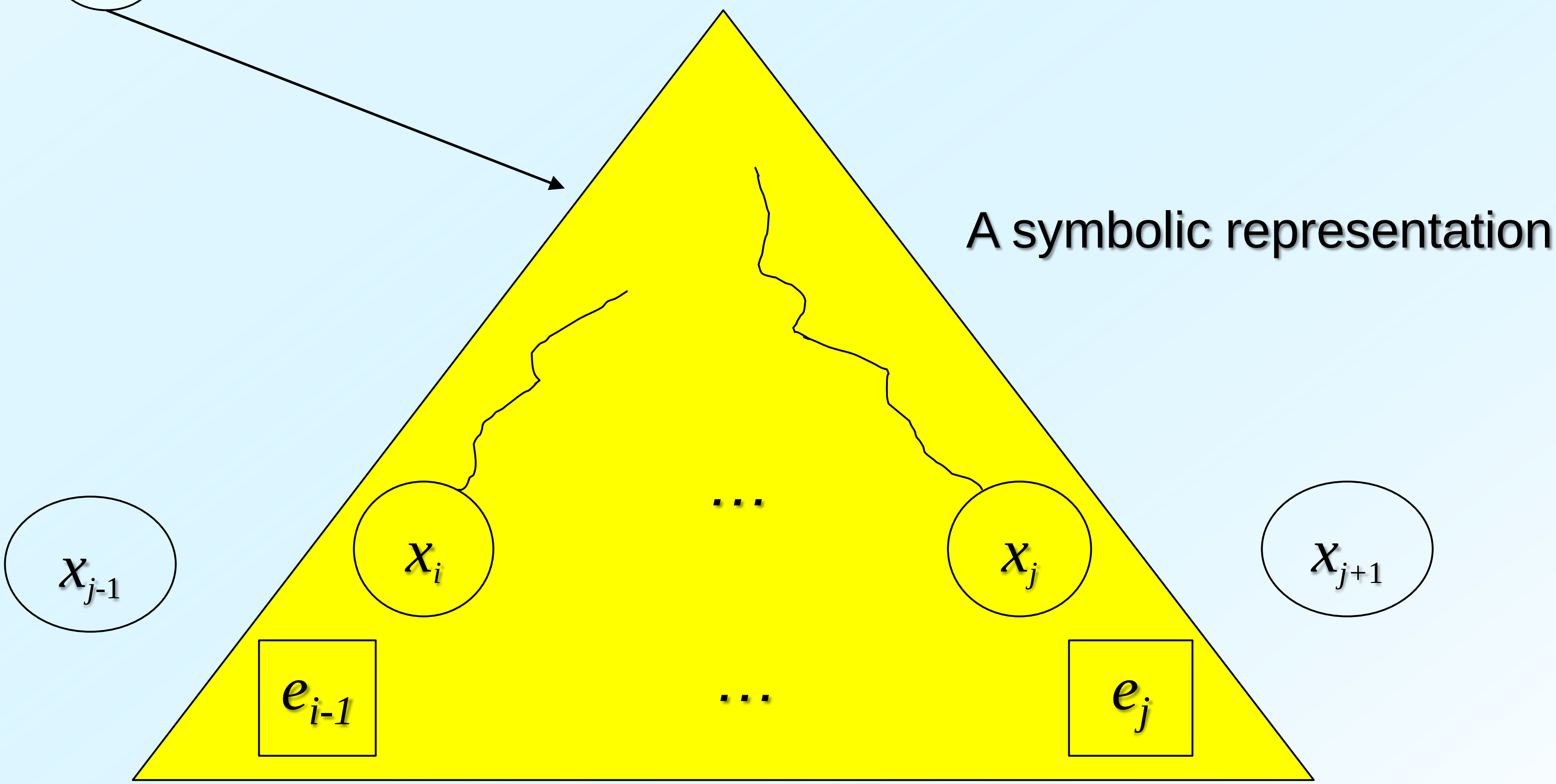
The cost(# of key comparisons) of a b.s.t. with $x_1 < x_2 < \cdots < x_n$

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

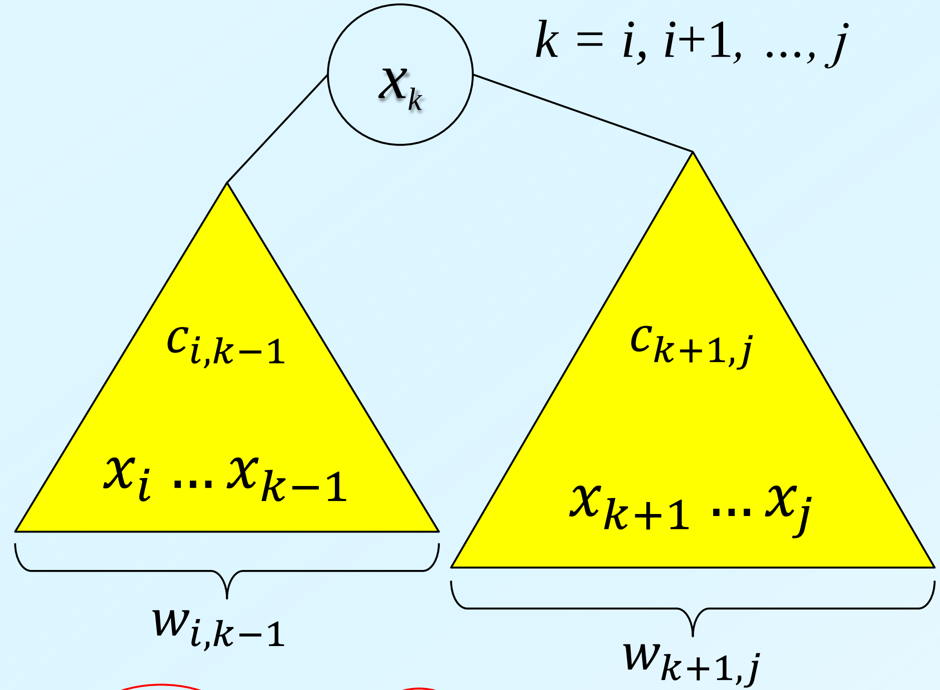
Consider the general case to optimize with the set $\{x_i, \ldots, x_j\}$

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 in $\{x_i, \ldots, x_j\}$



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

Optimal substructure

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

DP

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

BST(n):
    for $i \leftarrow 1$ **to** $n+1$
        $c_{i,i-1} \leftarrow q_{i-1}$
    for $m \leftarrow 1$ **to** n ◀ problem size
        for $i \leftarrow 1$ **to** $n - m + 1$ ◀ starting index
            $j \leftarrow i + m - 1$ ◀ ending index
            $c_{ij} \leftarrow \min\limits_{k=i,\ldots,j} \left(c_{i,k-1} + c_{k+1,j}\right) + w_{ij}$
    return c_{1n}

✓ Complexity: $\Theta(n^3)$

Running Time

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

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

$= m$ ← problem size

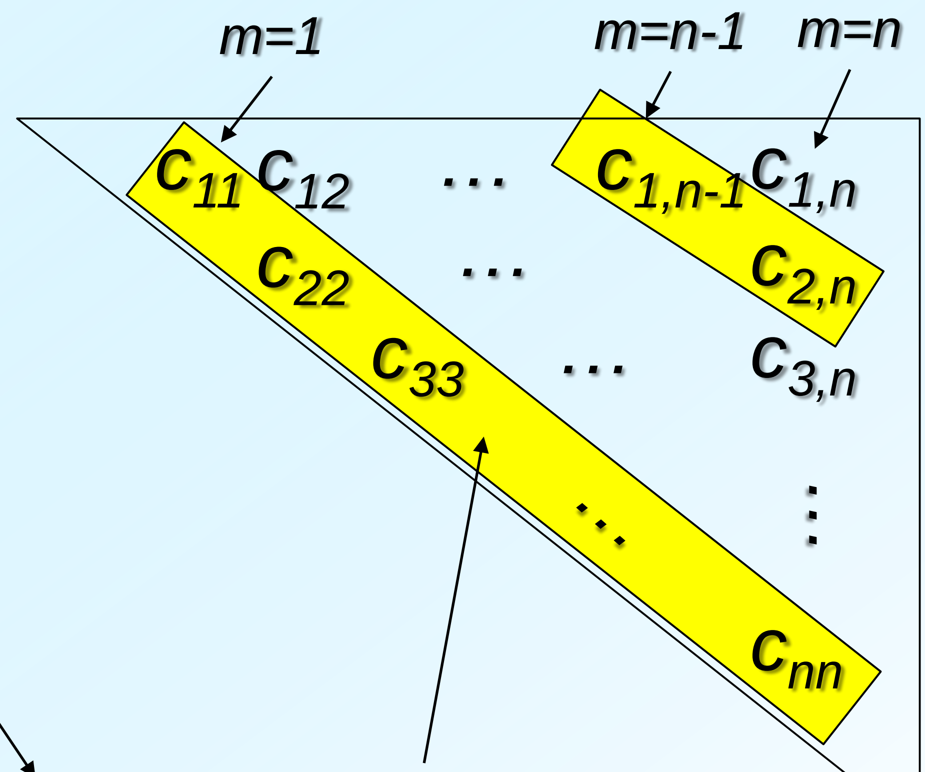
To compute c_{1n} :

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

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

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

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



n-m+1 = n-1+1 = n problems of size 1

Memoization DP

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

- All c_{ij}'s initialized to EMPTY
- All w_{ij}'s are computed in advance in $\Theta(n^2)$ time

BST(i, j):

 if $c_{ij} \neq$ EMPTY

  return c_{ij}

 else

  if $j = i - 1$

   $c_{ij} \leftarrow q_{i-1}$

  else

   $c_{ij} \leftarrow \min\limits_{k=i,\ldots,j} (\mathrm{BST}(i, k-1) + \mathrm{BST}(k+1, j)) + w_{ij}$

  return c_{ij}

✓ Solution: BST(1, n)

✓ Complexity: $\Theta(n^3)$

Memoization for pebble()

$$c_{ip} = \begin{cases} w_{ip} & \text{, if } i = 1 \\ w_{ip} + \max\limits_{q \text{ compatible to pattern } p} c_{i-1,q} & \text{, if } i > 1 \end{cases}$$

- Initialized: $\text{peb}_{i,p} \leftarrow -\infty$, i=2,3,4,…,n, p=1,2,3,4
- $\text{peb}_{i,p} \leftarrow \text{w}_{i,p}$, p=1,2,3,4

```
pebble(i, p):
    if (peb_{i,p} ≠ -∞)
        return peb_{i,p}
    else
        max ← -∞
        for every pattern q compatible to pattern p
            tmp ← pebble(i-1, q)
            if (tmp > max)
                max ← tmp
        return (peb_{i,p} ← max + w_{i,p})
```

Problem 6: Shortest Paths optional

- Given a weighted digraph $G=(V, E)$
  - w_{ij} : edge weight from vertex i to vertex j
    - ∞ if no edge
- Objective
  - Find the length of shortest path from starting vertex s to all other vertices

- d_t^k : Shortest path length from s to vertex t with at most k intermediate edges
- Objective: d_t^{n-1}
- Note! For $t \neq s$,
  - $d_t^0 = \infty$
  - $d_t^1 = w_{s,t}$

Before next page, think about what to use to figure out the core relationship

Optimal Substructure

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

DP

Bellman-Ford(G, s):

$d_s \leftarrow 0$

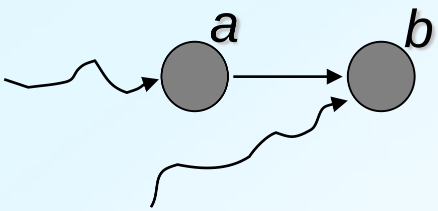
for all vertices $i \neq s$

$d_i \leftarrow \infty$

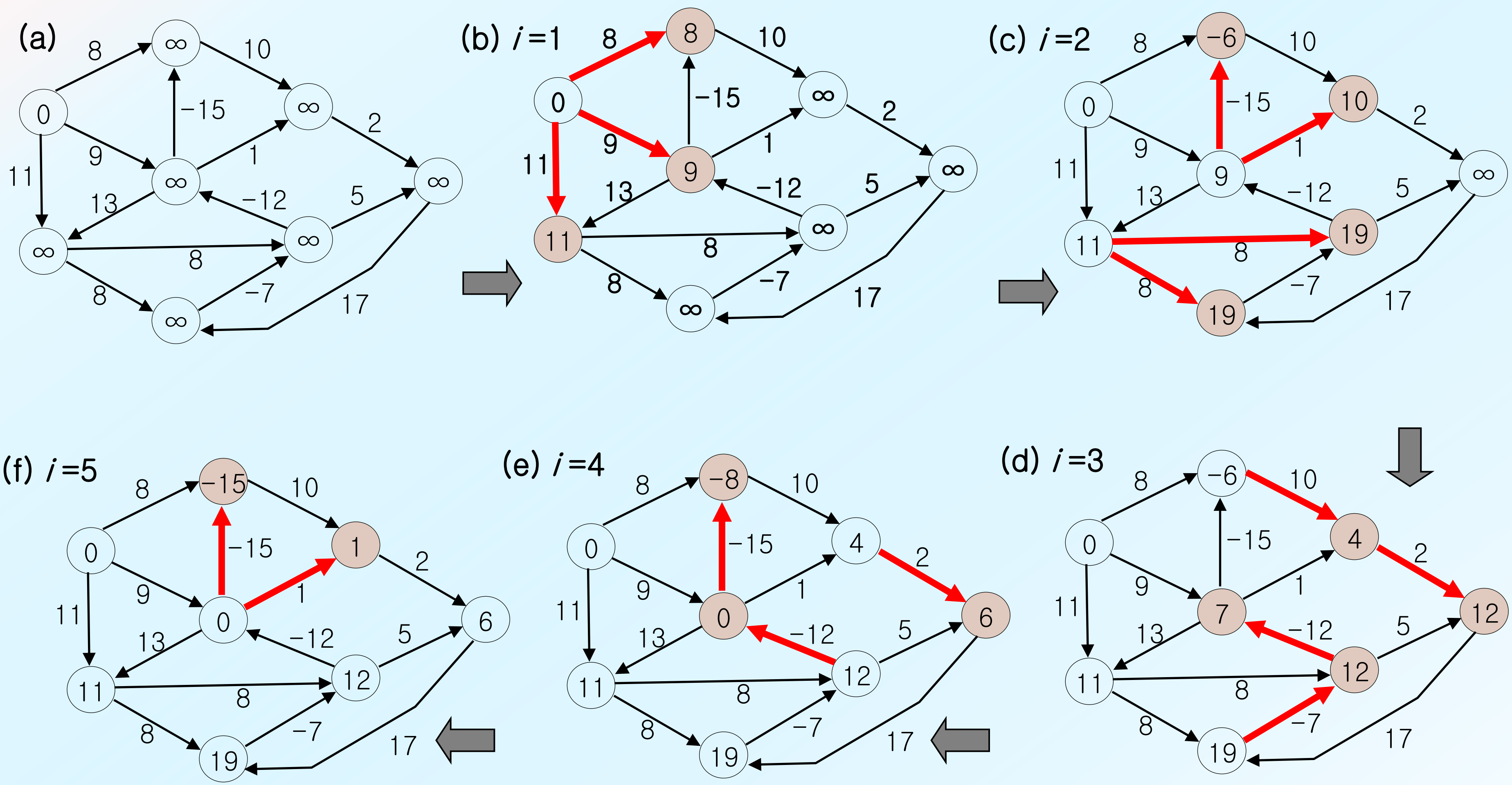
for $k \leftarrow 1$ **to** n-1

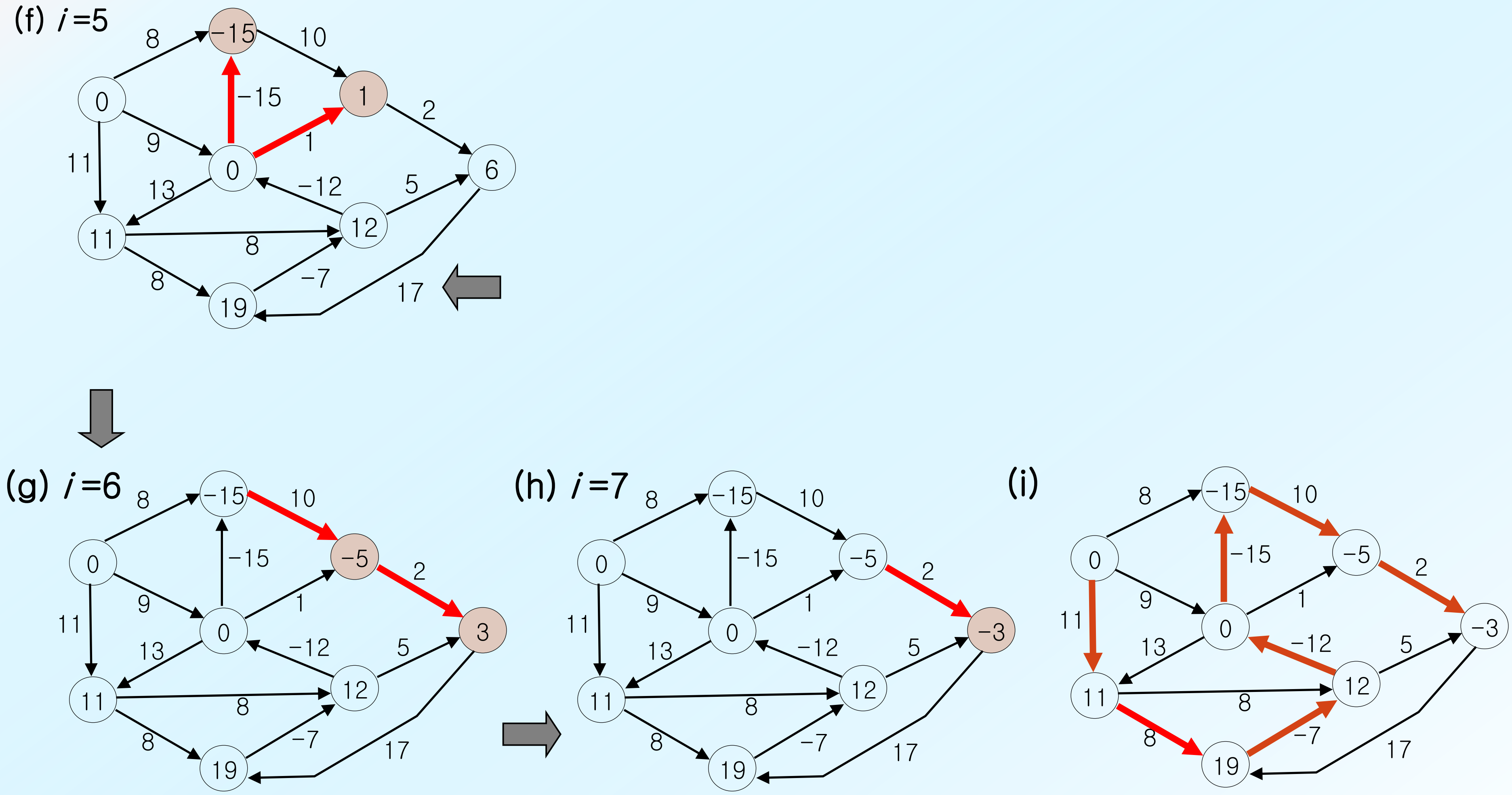
for all edges (a, b)

if $(d_a + w_{ab} < d_b)$ **then** $d_b \leftarrow d_a + w_{ab}$



✓ d_i 값 수정이 propagation 되어가는 모습이 직관적으로 그려지길 바람


(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)