09장. 동적 프로그래밍

Youn-Hee Han
LINK@KOREATECH
http://link.koreatech.ac.kr

계시란 바깥 어딘가에서 우리한테 갑자기 주어지는
객관적 지식이 아니다.
만물의 근원에 대한 본질적인 귀속감,
우리가 거기에 아주 밀접하게 닿아 있다는 관계성을
스스로가 발견해내는 것이 계시다.

-데이빗 스타인들-라스트

데이비드 스타인들-라스트 (David Steindl-Rast)

출가자

영어에서 번역됨 - David Steindl-Rast OSB는 미국 가톨릭 베네딕트 수도사, 작가 및 강사입니다. 그는 종교 간 대화에 전념하고 영성과 과학의 상호 작용을 다루었습니다.

학습 목표

- ◈ 동적 프로그래밍이 무엇인지 이해한다.
- ◈ 어떤 특성을 가진 문제가 동적 프로그래밍의 적용 대상인지 감지할 수 있도록 한다.
- ◈ 기본적인 몇 가지 문제를 동적 프로그래밍으로 해결할 수 있도록 한다.

01. 어떤 문제를 동적 프로그래밍으로 푸는가?

재귀적 해법의 빛과 그림자

◈ 피보나치수 구하기

– 피보나치 수열 정의

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}, \text{ for } n \geq 2$$

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, ⋯

알고리즘 9-1 피보나치 수(재귀호출)

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

재귀적 해법의 빛과 그림자

표 9–1 fib()에서 문제 크기가 커짐에 따라 중복 호출이 증가하는 모습

수행되는 fib( )	fib(2)의 중복 호출 횟수
fib(3)	1
fib(4)	2
fib(5)	3
fib(6)	5
fib(7)	8
fib(8)	13
fib(9)	21
fib(10)	34

◈ 피보나치수 구하기

– 재귀적 알고리즘의 단점
  - 과다한 중복 호출
  - 전체적인 재귀함수 호출 횟수: $\Omega(2^{\frac{n}{2}})$

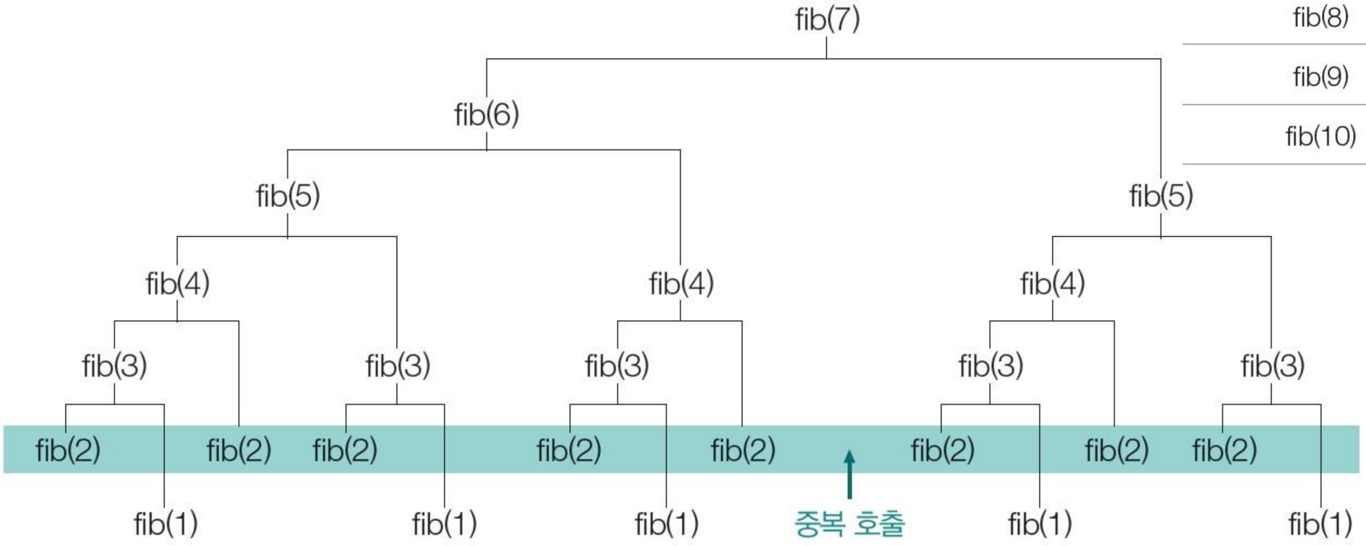


그림 9–1 재귀적 구현으로 동일한 문제가 중복 호출되는 상황을 보여주는 트리 예

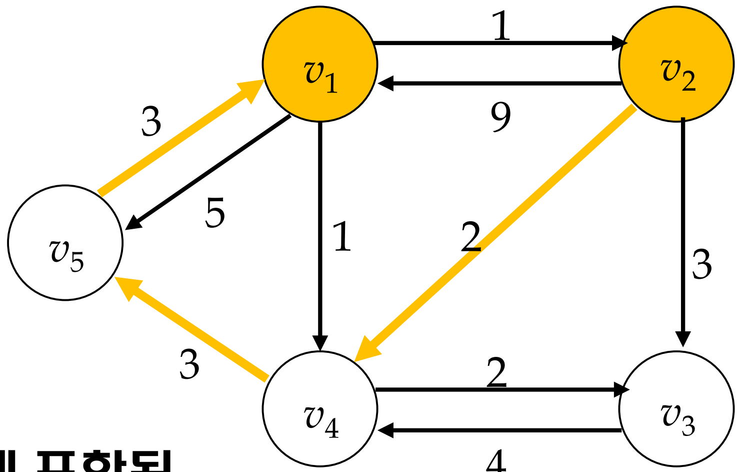
동적 프로그래밍의 적용 요건

◈ 문제 A가 **최적 부분 구조**를 지님

- 문제 A는 최적의 원칙 (The Principle of Optimality)을 따름
- "어떤 문제에 대한 최적 해가 그 문제를 분할한 부분문제에 대한 최적 해를 항상 포함"
- 피보나치 수열 문제

$$f_0 = 0$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2}, \text{ for } n \geq 2$$

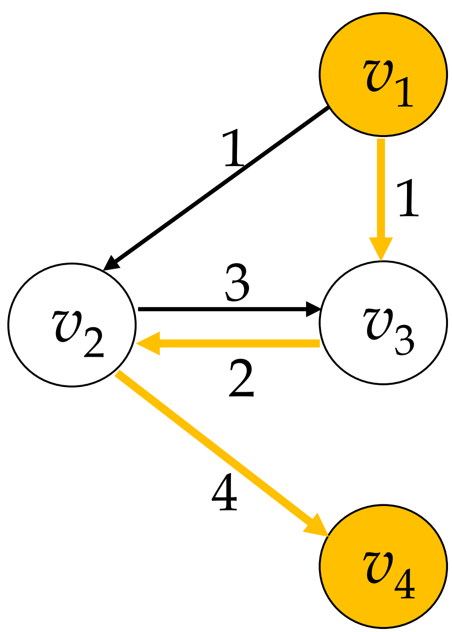
- 최단 경로(Shortest Path) 찾기 문제
  - v_2에서 v_1으로 가는 최단 경로
    - ➢ v_2 ➔ v_4 ➔ v_5 ➔ v_1
  - v_2에서 v_5 까지 가는 부분 최단 경로
    - ➢ v_2 ➔ v_4 ➔ v_5
  - 이 부분 최단 경로는 원문제의 최적해에 포함됨



동적 프로그래밍의 적용 요건

◈ 최적 부분 구조를 가지지 않는 문제

- 최장 경로(Longest Path) 찾기 문제
  - 가정: 문제에 대한 해를 단순 경로 (Simple Path)로 제한



- ➢ v_1에서 v_4로의 최장경로는 $[v_1, v_3, v_2, v_4]$가 된다.
  - Length($[v_1, v_3, v_2, v_4]$)=7
- ➢ 그러나, 이 경로의 부분 경로인 v_1에서 v_3으로의 최장경로는 $[v_1, v_3]$이 아니고, $[v_1, v_2, v_3]$이다.
  - Length($[v_1, v_3]$)=1
  - Length($[v_1, v_2, v_3]$)=4
- ➢ 따라서 최장 경로 찾기 문제는 최적 부분 구조를 지니지 않는다.

동적 프로그래밍의 적용 요건

◈ **최적 부분 구조**optimal substructure

- 큰 문제의 최적 솔루션에 작은 문제의 최적 솔루션이 포함됨

◈ **재귀호출시 비효율적인 중복**overlapping recursive calls

- 재귀적 해법으로 풀면 동일한 부분 문제에 대한 재귀호출이 심하게 중복됨

➡ **동적 프로그래밍이 해결책!**

동적 프로그래밍의 적용 요건

◈ 동적 프로그래밍 적용 예 1

- 알고리즘 복잡도: $\Theta(n)$

알고리즘 9-2	피보나치 수(동적 프로그래밍 1)

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

◈ 동적 프로그래밍 적용 예 2

- 추천하지는 않음. Why?
- 향상 방법은?

알고리즘 9-3	피보나치 수(동적 프로그래밍 2)

```
▷ 배열 f[]의 모든 원소는 0으로 초기화되어 있다.
▷ f[i]값이 0이면 fib(i)가 아직 한 번도 수행되지 않았음을 의미한다.
fib(n)
{
  ❶ if (f[n] ≠ 0) then return f[n];
    else {
            if (n=1 or n=2)
                then f[n] ← 1;
                else f[n] ← fib(n-1)+fib(n-2);
            return f[n];
    }
}
```

동적 프로그래밍의 특징

◈ 동적 프로그래밍 특징

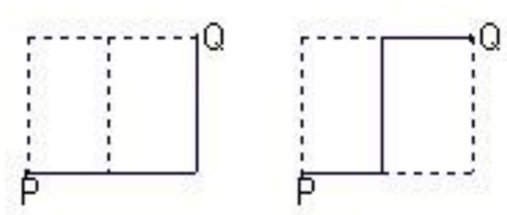
- 재귀적 관계식 (Recursive Relation)
  - 하지만, 재귀적 호출 방법보다 반복적 방법 사용
- Memoization (메모하기) (≈ Memorization)
  - 이미 풀어서 답을 알고 있는 부분의 결과가 다시 필요한 경우에는 반복하여 계산하는 대신에 이미 계산된 결과를 사용
- 상향식 해결법(Bottom-up approach)을 사용하여 알고리즘을 설계 ← 계획적 (Planning ≈ Programming)
  - 부분 결과를 한 번씩만 계산하고, 그러한 부분 결과를 차곡차곡 저장하고 활용하면서 원하는 답을 찾아

동적 계획법 – 이름 유래
https://ko.wikipedia.org/wiki/%EB%8F%99%EC%A0%81_%EA%B3%84%ED%9A%8D%EB%B2%95

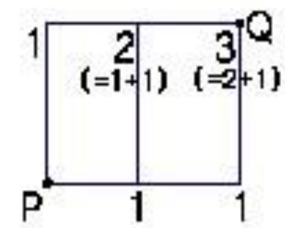
동적 프로그래밍의 특징

◈ 동적 프로그래밍 적용 예 3

– 도로 지도에서 출발점에서 목적지까지의 최단 경로의 수

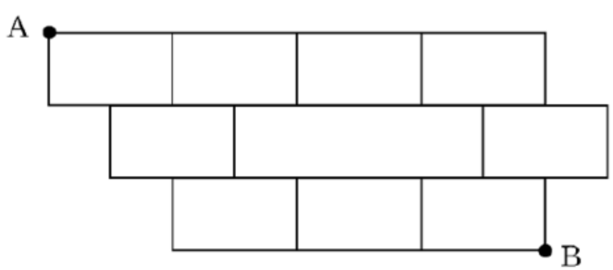


[그림1]



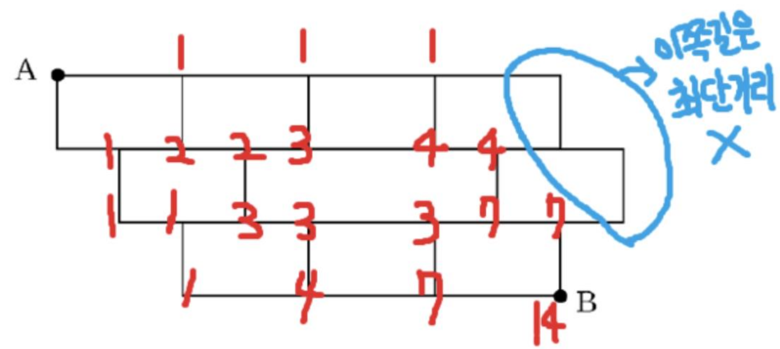
[그림2]

12. 그림과 같은 모양의 도로망이 있다. 지점 A에서 지점 B까지 도로를 따라 최단 거리로 가는 경우의 수는? (단, 가로 방향 도로와 세로 방향 도로는 각각 서로 평행하다.) [4점]



① 14 ② 16 ③ 18 ④ 20 ⑤ 22

<문제 1 정답 및 풀이>



정답 ①.

02. 행렬 경로 문제

행렬 경로 문제

◈ 행렬 경로 문제 정의

– 양수 원소들로 구성된 $n \times n$ 행렬이 주어지고, 행렬의 좌상단에서 시작하여 우하단까지 이동한다

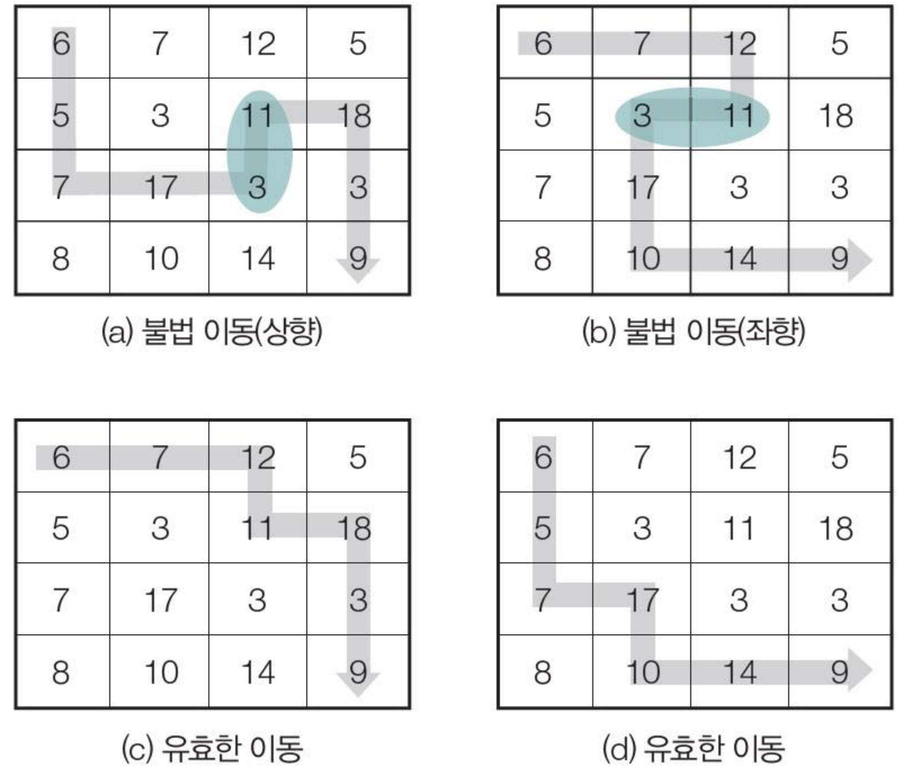


그림 9-2 허용되지 않는 이동과 허용되는 이동의 예

– 이동 방법 (제약조건)
  - 오른쪽이나 아래쪽으로만 이동할 수 있다
  - 왼쪽, 위쪽, 대각선 이동은 허용하지 않는다

– 목표: 행렬의 좌상단에서 시작하여 우하단까지 이동하되, 방문한 경로에 있는 수들의 합계 값이 최대가 되는 경로 및 그 값을 구함

행렬 경로 문제

	1	2	3	4
1	6	7	12	5
2	5	3	11	18
3	7	17	3	3
4	8	10	14	9

◈ 최적 부분 구조 확인 및 재귀적 관계 도출

– 임의의 (i, j) 위치에 도달하기 직전에 방문할 수 있는 위치는 $(i-1, j)$ 및 $(i, j-1)$ 단 두 개

– 즉, (i, j) 위치까지의 행렬 경로 문제의 해답은 다음 두 개의 부분 문제 해답을 포함

- 1) $(i, j-1)$ 위치까지의 행렬 경로 문제 해답 (부분 문제 해답)
- 2) $(i-1, j)$ 위치까지의 행렬 경로 문제 해답 (부분 문제 해답)

– 재귀적 관계 도출

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

- $m_{i,j}$: (i, j) 위치에 존재하는 수
- $c_{i,j}$: (i, j) 위치까지의 행렬 경로 문제의 해답

행렬 경로 문제의 재귀적 해법

◈ 재귀적 해법

– 중복 호출 횟수 증가

알고리즘 9-4	행렬 경로 문제(재귀호출)

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

```
matrixPath(i, j)
▷ (i, j)에 이르는 최고 점수
{
    if (i=0 or j=0) then return 0;
    else return (m_ij + (max(matrixPath(i-1, j), matrixPath(i, j-1))));
}
```

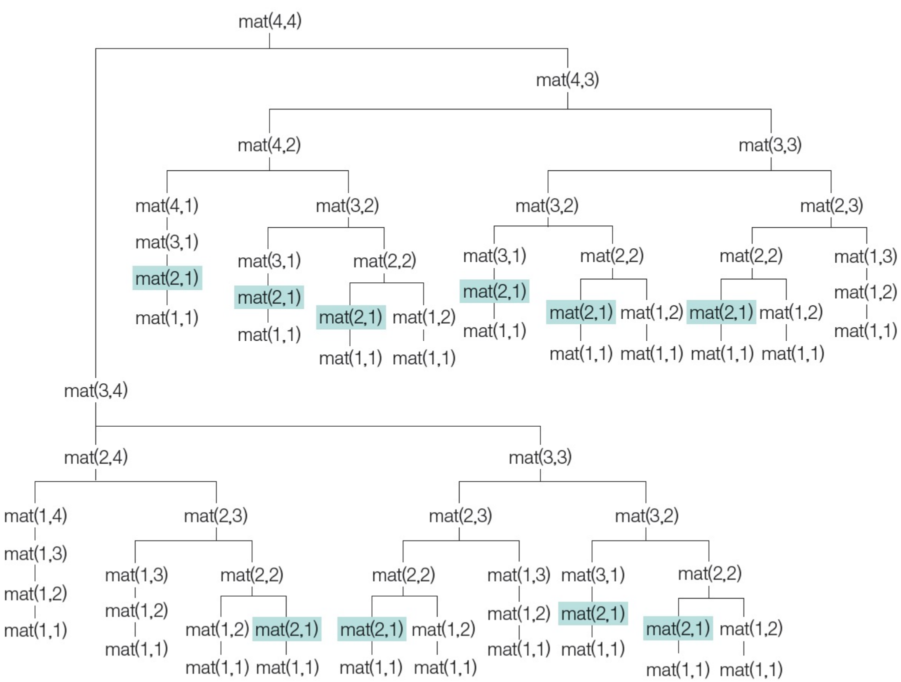


그림 9-3 matrixPath(4, 4)를 수행하는 과정의 재귀적 호출 관계를 나타내는 트리

표 9-2 matrixPath()에서 문제 크기가 커짐에 따라 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

행렬 경로 문제 동적 프로그래밍 해법

◈ 동적 프로그래밍 해법

알고리즘 9-5 행렬 경로 문제(동적 프로그래밍)

Recursive relation

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

matrixPath(n)

▷ (n, n)에 이르는 최고 점수

{

  for $i \leftarrow 0$ to n

    $c[i, 0] \leftarrow 0$;

  for $j \leftarrow 1$ to n

    $c[0, j] \leftarrow 0$;

① for $i \leftarrow 1$ to n

    for $j \leftarrow 1$ to n

      ② $c[i, j] \leftarrow m_{ij} + \max(c[i-1, j], c[i, j-1])$;

  return $c[n, n]$;

}

Bottom-up

Memoization

– 시간 복잡도: $\Theta(n^2)$

- 입력 크기(n)에 대하여 제곱 시간
- 행렬 원소 개수(n^2)에 대하여 선형 시간

03. 돌 놓기 문제

돌 놓기 문제

◈ 돌 놓기 문제 정의

– $3 \times N$ 테이블의 각 칸에 양의 정수 또는 음의 정수가 기록되어 있음

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

(a) 문제의 예

– 조약돌을 놓는 방법 (제약조건)

- 가로나 세로로 인접한 두 칸에 동시에 조약돌을 놓을 수 없다
- 각 열에는 적어도 하나 이상의 조약돌을 놓는다

– 목표: 돌이 놓인 자리에 있는 수의 합을 최대가 되도록 조약돌 놓기

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

(b) 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

(c) 합법적이지 않은 예

그림 9–4 돌 놓기 문제의 예와 돌을 놓은 예

돌 놓기 문제

◈ Brute-Force 방법

- 돌을 놓을 수 있는 모든 경우의 수를 따져본 다음 가장 높은 점수를 구함
- 알고리즘 복잡도: $\Theta(2^{3\times N})$

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

(a) 문제의 예

돌 놓기 문제

◈ 최적 부분 구조 확인

– 임의의 열을 기준으로 <u>돌을 놓을 수 있는 방법 4가지</u> (4가지 패턴)

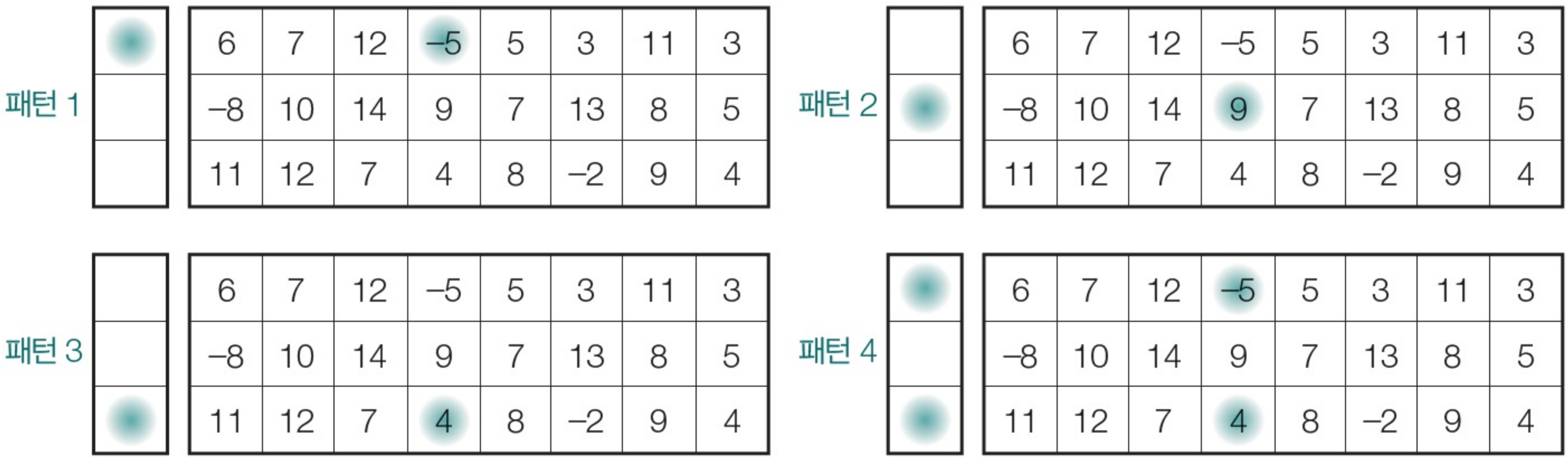


그림 9–5 임의의 열에 놓을 수 있는 네 가지 패턴 예

돌 놓기 문제

◈ 최적 부분 구조 확인

– 4가지 패턴 각각에 대하여 양립할 수 있는 패턴 정리

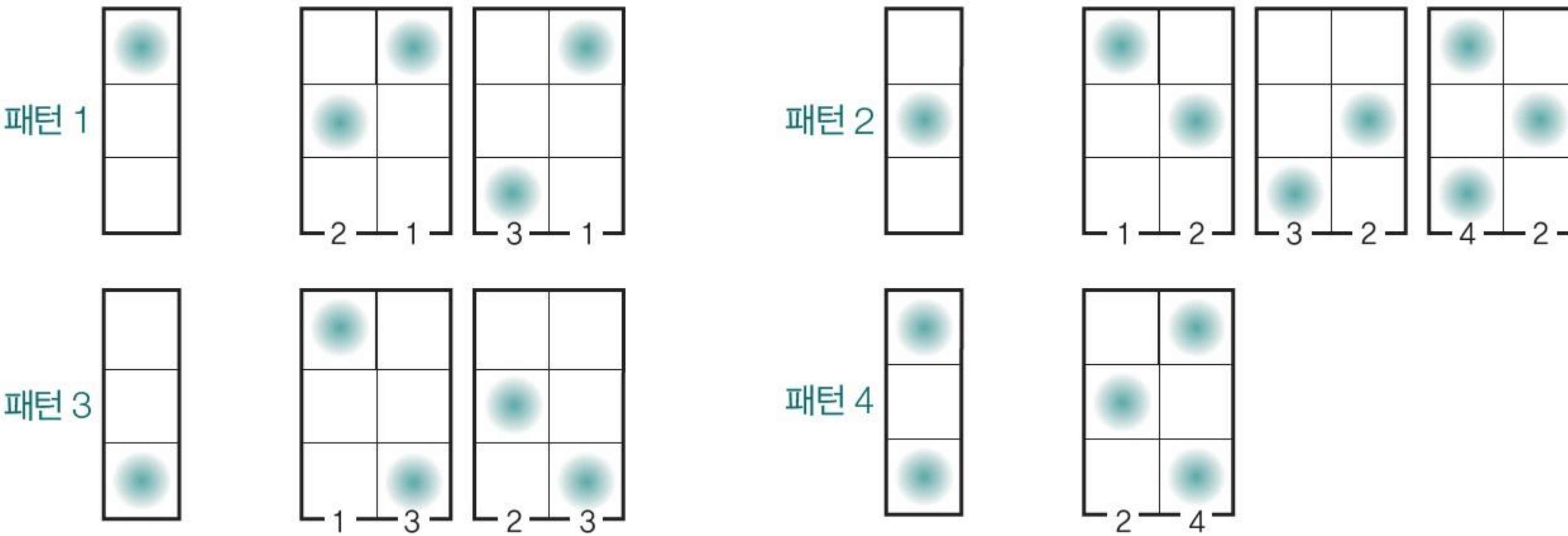


그림 9–6 서로 양립할 수 있는 패턴들

돌 놓기 문제

◈ 최적 부분 구조 확인

- N 열 중 1열 부터 차례로 i열까지 돌을 놓는 것으로 생각
  - 1열 부터 i열까지 합의 최고치를 고려

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

- i열에 대해서 고려해야 할 두 가지
  - i열에는 다음 4 가지 패턴 중 하나를 선택
    - ➢ i열이 패턴 1로 놓여 있을 경우
    - ➢ i열이 패턴 2로 놓여 있을 경우
    - ➢ i열이 패턴 3으로 놓여 있을 경우
    - ➢ i열이 패턴 4로 놓여 있을 경우
  - 1열 부터 $i-1$열까지의 최고 점수
    - ➢ 이 때, i열에 돌이 놓아지는 패턴을 고려하여 <u>$i-1$열 돌이 놓여지는 패턴</u>은 유효한 패턴만 고려
  - 결국 **i열까지 고려한 최적 해에 $i-1$열까지의 최적 해가 포함**됨

돌 놓기 문제

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

◈ 재귀적 관계 도출

$$c_{i,p} = \begin{cases} w_{1,p} & \text{if } i = 1 \\ \max\limits_{p\text{와 양립하는 각 패턴 } q} \{c_{i-1,q}\} + w_{i,p} & \text{if } i > 1 \end{cases}$$

– $c_{i,p}$: 1열 부터 $i-1$열까지의 최고 점수에 i열이 패턴 p로 놓일 때의 점수를 더한 값

– $w_{i,p}$: i열이 패턴 p로 놓일 때 점수 합
  - 각각의 i열 모두에 대해 알고리즘 수행 전 미리 구성 가능

– 위와 같은 관계식을 활용하여 최종적으로는 $c_{N,1}$~ $c_{N,4}$ 중 가장 큰 값이 해당 문제의 최종 답

돌 놓기 문제의 재귀적 해법

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

◈ 재귀적 해법

알고리즘 9–6 돌 놓기 문제(재귀호출)

```
pebble(i, p)                                                    c_{i,p}
▷ i열이 패턴 p로 놓일 때 최고 점수
▷ w[i, p] : i열이 패턴 p로 놓일 때 i열에 돌이 놓인 곳의 점수 합, p∈{1, 2, 3, 4}      w_{i,p}
{
    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]);
        }
}
```

각각의 i열 모두에 대해 미리 구성

$$c_{i,p} = \begin{cases} w_{1,p} & \text{if } i = 1 \\ \max_{p\text{와 양립하는 각 패턴 } q} \{c_{i-1,q}\} + w_{i,p} & \text{if } i > 1 \end{cases}$$

돌 놓기 문제의 재귀적 해법

◈ 재귀적 해법

– 중복 호출 횟수 증가

표 9-3 pebble()에서 문제가 커짐에 따라 중복 호출의 비율이 증가하는 모습

문제의 크기(n)	부분 문제의 총수	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

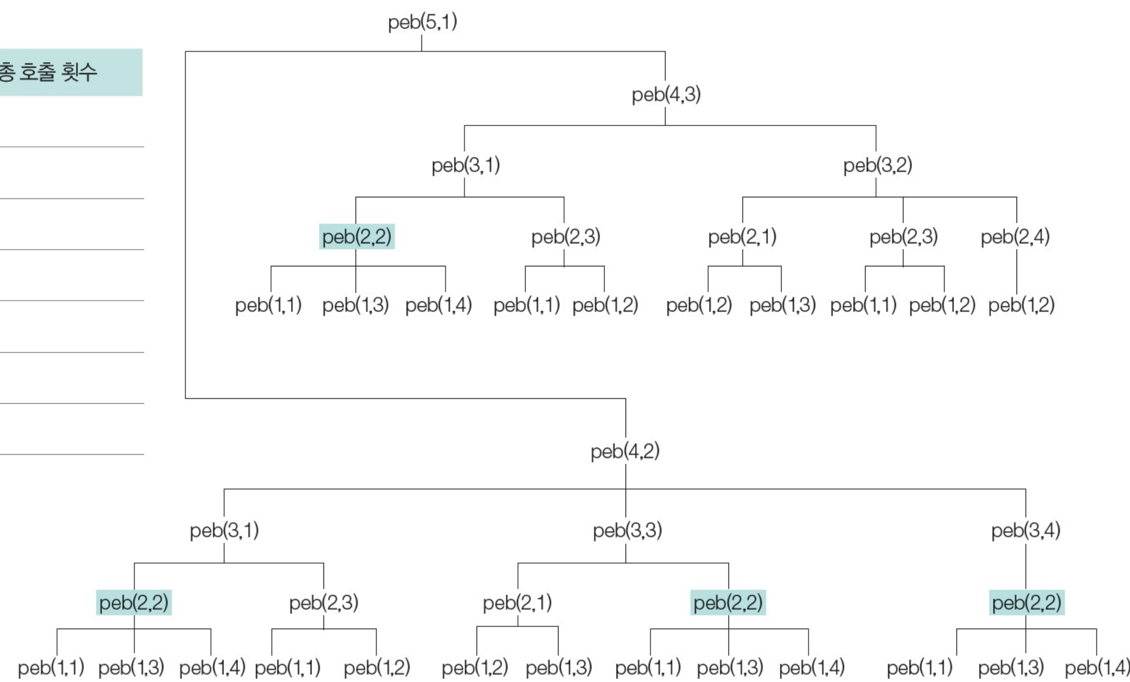


그림 9-7 pebble(5, 1)을 수행하는 과정의 재귀적 호출 관계를 나타내는 트리

돌 놓기 문제의 재귀적 해법

◈ 동적 프로그래밍 해법

알고리즘 9-7	돌 놓기 문제(동적 프로그래밍)

pebble(n) ← $c_{i,n}$

▷ n열까지 돌을 놓을 때 최고 점수

{

  for p ← 1 to 4

    $peb[1, p] \leftarrow w[1, p]$;

❶ for i ← 2 to n

    for p ← 1 to 4

      $peb[i, p] \leftarrow \max_{p\text{와 양립하는 패턴 } q}\{peb[i-1, q]\} + w[i, p]$;

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

}

$$c_{i,p} = \begin{cases} w_{1,p} & if\ i = 1 \\ \max_{p\text{와 양립하는 각 패턴 } q}\{c_{i-1,q}\} + w_{i,p} & if\ i > 1 \end{cases}$$

$w_{i,p}$ 각각의 i열 모두에 대해 미리 구성

– 알고리즘 복잡도: $\Theta(n)$

04. 행렬 곱셈 순서 문제

행렬 곱셈 순서 문제

◈ 행렬 곱셈 순서 문제 정의

- $i \times j$ 행렬과 $j \times k$ 행렬을 곱하기 위해서는 일반적으로 $\underline{i \times j \times k}$번 만큼의 기본적인 곱셈이 필요

2×3 행렬과 3×4 행렬을 곱하기

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 & 1 \\ 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 \end{bmatrix} = \begin{bmatrix} 29 & 35 & 41 & 38 \\ 74 & 89 & 104 & 83 \end{bmatrix}$$

- $29 = 1\times7 + 2\times2 + 3\times6$, 이와 같은 계산을 결과 행렬 $i\times k$번 수행해야 한다.

➔ j번의 곱셈을 총 $i \times k$번 수행

즉, 전체적으로 곱셈을 $i \times j \times k$ *번 수행*

행렬 곱셈 순서 문제

◈ 행렬 곱셈 순서 문제 정의

- 연쇄적으로 행렬을 곱할 때, 어떤 행렬 곱셈을 먼저 수행 하느냐에 따라서 필요한 기본적인 곱셈의 횟수가 달라지게 된다.
- 예를 들어서, 다음 행렬 곱셈 문제를 생각해 보자:
  - $(A_1 \times A_2) \times A_3 \quad = \quad A_1 \times (A_2 \times A_3)$
    - A_1의 크기는 10×100이고,
    - A_2의 크기는 100×5이고,
    - A_3의 크기는 5×50라고 하자.
  - $A_1 \times A_2$를 먼저 계산한다면, 기본적인 곱셈의 총 횟수는 7,500회
    - (10 * 100 * 5) + (10 * 5 * 50) = 5,000 + 2,500 = 7,500
  - $A_2 \times A_3$를 먼저 계산한다면, 기본적인 곱셈의 총 횟수는 75,000회
    - (100 * 5 * 50) + (10 * 100 * 50) = 25,000 + 50,000 = 75,000

행렬 곱셈 순서 문제

◈ 행렬 곱셈 순서 문제 정의

- **또 다른 예제**
  - 다음과 같은 일련의 행렬을 곱하는 경우를 생각하여 보자.

$$\underbrace{A}_{20\times 2} \times \underbrace{B}_{2\times 30} \times \underbrace{C}_{30\times 12} \times \underbrace{D}_{12\times 8}$$

  - 중요한 사실. 곱하는 순서는 결과에 영향을 주지 않는다.
    - 예) $A(B(CD)) = (AB)(CD)$
  - 일련의 행렬을 곱하는 경우에는 곱하는 순서에 따라 필요한 기본 연산의 수가 크게 달라진다.
    - 예5.4)

$A(B(CD)$	$30\times12\times8+2\times30\times8+20\times2\times8 = 3680$
$(AB)(CD)$	$20\times2\times30+30\times12\times8+20\times30\times8 = 8880$
$A((BC)D)$	$2\times30\times12+2\times12\times8+20\times2\times8 = 1232$
$((AB)C)D$	$20\times2\times30+20\times30\times12+20\times12\times8 = 10320$

- **알고리즘 개발 목표**: n**개의 행렬**$(A_1, A_2, \ldots, A_n)$ **을 연쇄적으로 곱할 때 기본적인 곱셈의 횟수가 가장 적게 되는 최적의 순서를 결정**

행렬 곱셈 순서 문제

◈ Brute-Force Algorithm

- 가능한 모든 순서를 전부 고려해 보고, 그 중에서 곱셈 연산의 수가 가장 최소인 것을 택한다.
- 시간복잡도 분석: 최소한 지수(exponential-time) 시간 ($\Omega(2^n)$)

[증명]

- n개의 행렬($A_1, A_2, \ldots, A_n$)을 곱할 수 있는 모든 순서의 가지 수: t_n
- $n=2$일 때, $t_2 = 1$이라는 사실은 쉽게 알 수 있다.
- $n>2$일 때, 만약 A_1이 마지막으로 곱하는 행렬이라고 하면,
  행렬 $A_2, \ldots, A_n$을 곱하는 데는 t_{n-1}개의 가지 수가 있다고 가정하자.
- A_n이 마지막으로 곱하는 행렬이라고 하면,
  행렬 $A_1, \ldots, A_{n-1}$을 곱하는 데는 또한 수$_{-1}$개의 가지 수가 있다.
- 그러면, $t_n \geq t_{n-1} + t_{n-1} = 2\, t_{n-1}$이고,
- 따라서, $t_n \geq 2t_{n-1} \geq 2^2 t_{n-2} \geq \ldots \geq 2^{n-2} t_2 = 2^{n-2}$ 이다.

행렬 곱셈 순서 문제

◈ 이 문제는 **최적 부분 구조**를 지니고 있는가?

- n개의 행렬을 곱하는 최적의 순서는 n개의 행렬 중 일부 부분집합에 속하는 행렬을 곱하는 최적의 순서를 항상 포함한다.
  - 예를 들어, 6개의 행렬($A_1, A_2, \ldots, A_6$)을 곱한다고 가정할 때 최적해가 다음과 같다면,

$$\boxed{A_1\ ((((A_2\ A_3)\ A_4)\ A_5)\ A_6)}$$

  - 3개의 행렬(A_2, A_3, A_4)을 곱하는 최적의 순서는 다음과 같다.

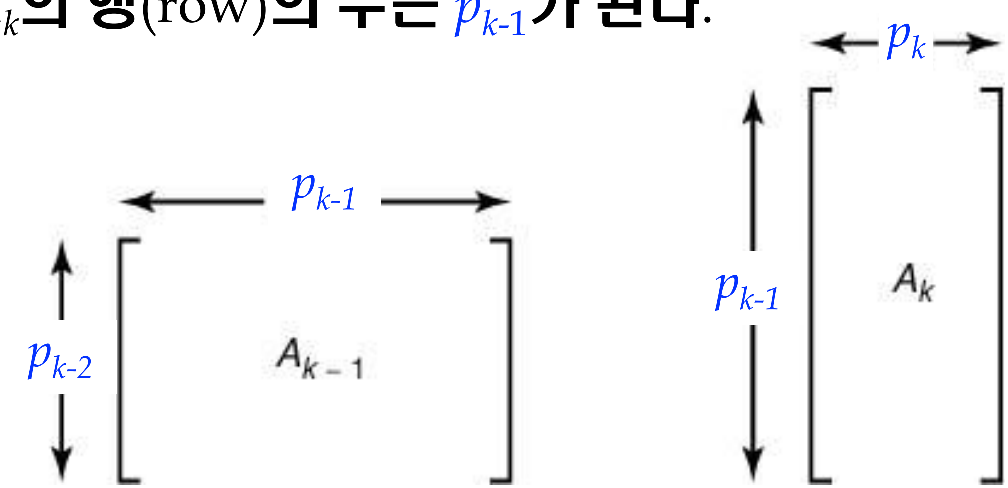
$$\boxed{(A_2\ A_3)\ A_4}$$

- 위와 같은 상황이 모든 경우에 대해 만족
  → "최적 부분 구조"를 지니고 있고, 또한 동적 프로그램 알고리즘 구축 가능

행렬 곱셈 순서 문제

◈ 표기법 정리

- n개의 행렬($A_1, A_2, \ldots, A_n$)을 연쇄적으로 곱할 때, p_k를 행렬 A_k의 열(column)의 수라고 정의하면, p_{k-1}는 행렬 A_{k-1}의 열의 수
- 자연히 A_k의 행(row)의 수는 p_{k-1}가 된다.



- A_1의 행의 수는 p_0라고 하자.
- 그렇다면, 간단하게 $(A_1 A_2) A_3$의 곱셈 연산 횟수는?: $p_0 p_1 p_2 + p_0 p_2 p_3$

행렬 곱셈 순서 문제

◈ 재귀적 관계 도출

- $C_{i,j}$: A_i부터 A_j까지의 행렬을 곱하는데 필요한 곱셈의 최소 횟수
- 예를 들어, $C_{1,6}$을 구하기 위하여 다음과 같이 두 개의 부분으로 나누어서 계산한다고 생각하며, 두 부분으로 나누는 각 경우에 대해 가장 적은 곱셈 횟수를 산출하는 경우만을 선택한다.
  - 6개의 행렬을 곱하는 최적의 순서는 다음 중 하나에 포함된다.
    - $A_1(A_2A_3A_4A_5A_6)$
    - $(A_1A_2)(A_3A_4A_5A_6)$
    - $(A_1A_2A_3)(A_4A_5A_6)$
    - $(A_1A_2A_3A_4)(A_5A_6)$
    - $(A_1A_2A_3A_4A_5)(A_6)$

사실 1. 최적 원칙이 적용되는 문제이다.
사실 2. 옆에 있는 다섯 가지는 크게 두 부분으로 나누어진다.
중요 관찰. 나누어진 두 부분은 사실 1에 의해 최적이다.

  - 따라서 최적의 순서는 다음과 같이 정의된다.

$$C_{1,6} = \min_{1 \le k \le 5}\{C_{1,k} + C_{k+1,6} + p_0p_kp_6\}$$

행렬 곱셈 순서 문제

$$C_{1,6} = \min_{1 \le k \le 5} \{C_{1,k} + C_{k+1,6} + p_0 p_k p_6\}$$

일반화

◈ 재귀적 관계 도출

– $C_{i,j}$: A_i부터 A_j까지의 행렬을 곱하는데 필요한 곱셈의 <u>최소</u> 횟수

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

$k = i,\ i+1,\ \ldots,\ j\text{-}2,\ j\text{-}1$

$$\underbrace{(A_i \cdots A_k)}_{p_{i-1} \times p_k}\underbrace{(A_{k+1} \cdots A_j)}_{p_k \times p_j}$$

행렬 곱셈 순서 문제

◈ 행렬 곱셈 순서 문제 예

- $\underbrace{A_1}_{5\times2} \times \underbrace{A_2}_{2\times3} \times \underbrace{A_3}_{3\times4} \times \underbrace{A_4}_{4\times6} \times \underbrace{A_5}_{6\times7} \times \underbrace{A_6}_{7\times8}$

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

$C_{i,j} = M[i][j]$

	1	2	3	4	5	6
1	0	30	64	132	226	348
2		0	24	72	156	268
3			0	72	198	366
4				0	168	392
5					0	336
6						0

M[i][i] = 0
M[1][2] = M[1][1] + M[2][2] + 5×2×3 = 30
M[2][3] = M[2][2] + M[3][3] + 2×3×4 = 24
…
M[5][6] = M[5][5] + M[6][6] + 6×7×8 = 336
M[1][3] = min(M[1][1] + M[2][3] + 5×2×4,
M[1][2] + M[3][3] + 5×3×4)
= min(24+40,30+60) = 64
…
M[1][4] = min(M[1][1] + M[2][4] + 5×2×6,
M[1][2] + M[3][4] + 5×3×6,
M[1][3] + M[4][4] + 5×4×6)
= min(72+60,30+72+90,64+120) = 132

즉, 행렬 M은 M[i][i]을 모두 0으로 세팅한 이후
대각선 1, 대각선 2, 대각선 3, 대각선 4, 대각선 5 순으로 각 원소를 계산한다.

행렬 곱셈 순서 문제

◈ 의사 코드 (재귀호출)

알고리즘 9-8 | 행렬 곱셈 순서 문제(재귀호출)

```
rMatrixChain(i, j)        ▷ 행렬곱 A_i...A_j을 구하는 최소 비용을 구한다.
{
    if (i=j) then return 0;        ▷ 행렬이 하나뿐인 경우의 비용은 0
    min ← ∞;
    for k ← i to j–1 {
     ❶ q ← rMatrixChain(i, k) + rMatrixChain(k+1, j) + p_{i-1}p_k p_j;
        if (q < min) then min ← q;
    }
    return min;
}
```

k: 대각선 1, 대각선 2, 대각선 3, 대각선 4, 대각선 5

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

- p_i 값은 글로벌 변수로 미리 주어짐
- main() 내부에서 rMatrixChain(1, 6) 호출
- 시간 복잡도: $\Omega(2^n)$

행렬 곱셈 순서 문제

	1	2	3	4	5	6
1	0	30	64	132	226	348
2		0	24	72	156	268
3			0	72	198	366
4				0	168	392
5					0	336
6						0

◈ 의사 코드 (동적프로그래밍)

알고리즘 9-9 | 행렬 곱셈 순서 문제(동적 프로그래밍)

r: 대각선 1, 대각선 2, 대각선 3, 대각선 4, 대각선 5

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

$$\underbrace{A_1}_{5\times2} \times \underbrace{A_2}_{2\times3} \times \underbrace{A_3}_{3\times4} \times \underbrace{A_4}_{4\times6} \times \underbrace{A_5}_{6\times7} \times \underbrace{A_6}_{7\times8}$$

r	i	j
1	1	2
1	2	3
1	3	4
1	4	5
1	5	6
2	1	3
2	2	4
2	3	5
2	4	6
3	1	4
3	2	5
3	3	6
4	1	5
4	2	6
5	1	6

– p_i 값은 글로벌 변수로 미리 주어짐
– main() 내부에서 matrixChain(6) 호출
– 시간 복잡도: $\Theta(n^3)$

05. 최장 공통 부분 순서 (생략)

Questions & Answers