파이썬으로 배우는 알고리즘 기초

Chap 3. 동적계획

3.4

연쇄 행렬 곱셈

3.4 연쇄 행렬 곱셈

연쇄 행렬 곱셈 문제

- 주어진 n개의 연쇄 행렬을 곱하는 최적의 순서를 구하시오.
  - n개의 연쇄 행렬 곱셈: $A_1 \times A_2 \times \cdots \times A_n$
  - 행렬 곱셈은 결합 법칙이 성립: $(A_x \times A_y) \times A_z = A_x \times (A_y \times A_z)$
  - 하지만, 행렬 곱셈의 순서에 따라서 각 원소의 곱셈 횟수가 달라짐
  - 각 원소의 곱셈 횟수가 가장 작아지도록 하는 곱셈 순서가 최적의 순서

- 연쇄 행렬 곱셈 문제는 최적화 문제
  - 원소의 곱셈 횟수를 최소화하는 행렬 곱셈의 순서 찾기

3.4 연쇄 행렬 곱셈

- **연쇄 행렬 곱셈 문제의 이해**
  - 2×3 **행렬과** 3×4 **행렬을 곱하면** 2×4 **행렬이 나옴**
    - Algorithm 1.4: **원소를 곱하는 횟수는** $2 \times 3 \times 4 = 24$
  - **일반적으로,** $i \times k$ **행렬과** $k \times j$ **행렬을 곱하면** $i \times j$ **행렬이 나옴**
    - **원소 곱셈의 횟수:** $i \times k \times j$

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

3.4 연쇄 행렬 곱셈

- **연쇄 행렬 곱셈**: 단순무식하게 풀기(Brute-Force Approach)
  - 모든 경우의 수에 대해서 계산해 보고 최적의 순서를 선택
  - 연쇄 행렬 곱셈에서 가능한 경우의 수는?
    - **카탈란 수**: $C(n) = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$
    - 연쇄 행렬 곱셈이 가지는 경우의 수 = $C(n-1)$
      - n개의 항에 괄호를 씌우는 모든 경우의 수 $(n = 1, 2, 3, \cdots)$

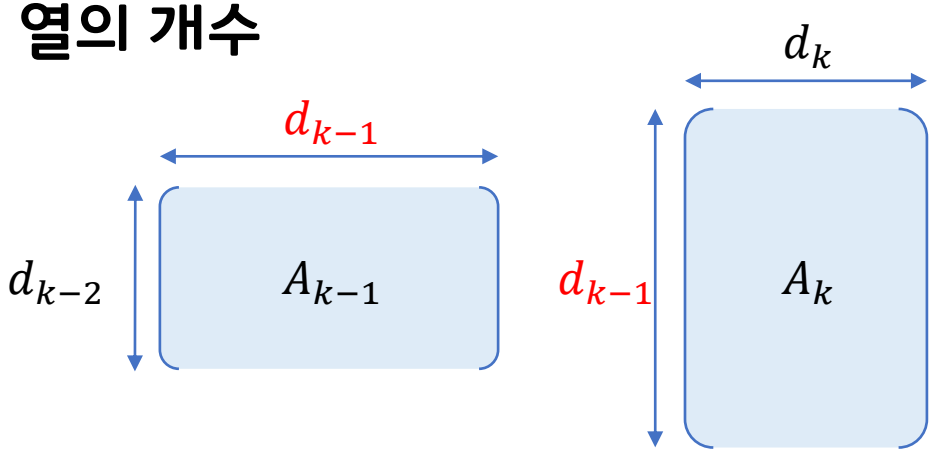
3.4 연쇄 행렬 곱셈

$$\underset{(20\times 2)}{A} \times \underset{(2\times 30)}{B} \times \underset{(30\times 12)}{C} \times \underset{(12\times 8)}{D}$$

- **연쇄 행렬이 4개일 경우 다섯가지 경우의 수가 존재**
  - $A(B(CD)) = 3{,}680$
  - $(AB)(CD) = 8{,}880$
  - $A((BC)D) = 1{,}232$
  - $((AB)C)D = 10{,}320$
  - $(A(BC))D = 3{,}120$

3.4 연쇄 행렬 곱셈

- **연쇄 행렬 곱셈 문제의 엄밀한 정의**
  - n개의 연쇄 행렬 곱셈: $A_1 \times A_2 \times \cdots \times A_n$
  - A_{k-1}의 행의 개수와 A_k의 열의 개수가 같아야 함
  - d_k를 행렬 A_k의 행의 개수로 정함 $(1 \le k \le n)$
  - d_{k-1}은 행렬 A_k의 열의 개수, A_{k-1}의 행의 개수임
  - d_0는A_1의 열의 개수



3.4 연쇄 행렬 곱셈

- **연쇄 행렬 곱셈: 동적계획(Dynamic Programming)**
  - 1단계: **재귀 관계식**을 찾는다.
    - M: 연쇄 행렬을 곱하는데 필요한 곱셈의 최소 회수 행렬
    - $M[i][j]$: A_i에서 A_j까지 행렬을 곱하는데 필요한 곱셈의 최소 회수 $(1 \le i \le j \le n)$
    - 목표: $A_i \cdots A_j$ 행렬을 $(A_i \cdots A_k)(A_{k+1} \cdots A_j)$로 분할하는 재귀 관계식 찾기
  - 2단계: **상향식 방법**으로 해답을 구한다.
    - 초기화: $M[i][i] = 0$ (주대각선을 0으로)
    - 최종 목표:$M[1][n]$.
    - 상향식 계산: 대각선 1번, 대각선 2번, $\cdots$, 대각선 $n-1$번

3.4 연쇄 행렬 곱셈

- **연쇄 행렬 곱셈의 재귀 관계식 구하기**
  - **분할정복**(Divide-and-Conquer)
    - n**개의 행렬을 두 개의 최적 부분행렬의 곱으로 분할**
  - **예를 들어,** $A_1A_2A_3A_4A_5A_6$ **은 다음과 같이 분할 가능**
  - **각 분할의 곱셈 횟수:**
    - **각 부분행렬의 곱셈 횟수 + 두 행렬의 곱셈 횟수**
    - $M[1][k] + M[k+1][6] + d_0d_kd_6$
  - **최적 분할:**
    - $M[1][6] = \underset{i \le k \le j-1}{\text{minimum}}(M[1][k] + M[k+1][6] + d_0d_kd_6)$

$$(A_1)(A_2A_3A_4A_5A_6): k = 1$$
$$(A_1A_2)(A_3A_4A_5A_6): k = 2$$
$$(A_1A_2A_3)(A_4A_5A_6): k = 3$$
$$(A_1A_2A_3A_4)(A_5A_6): k = 4$$
$$(A_1A_2A_3A_4A_5)(A_6): k = 5$$

3.4 연쇄 행렬 곱셈

$$\underset{(5\times 2)}{A_1} \times \underset{(2\times 3)}{A_2} \times \underset{(3\times 4)}{A_3} \times \underset{(4\times 6)}{A_4} \times \underset{(6\times 7)}{A_5} \times \underset{(7\times 8)}{A_6}$$

$d_0\ d_1 \quad d_2 \quad d_3 \quad d_4 \quad d_5 \quad d_6$

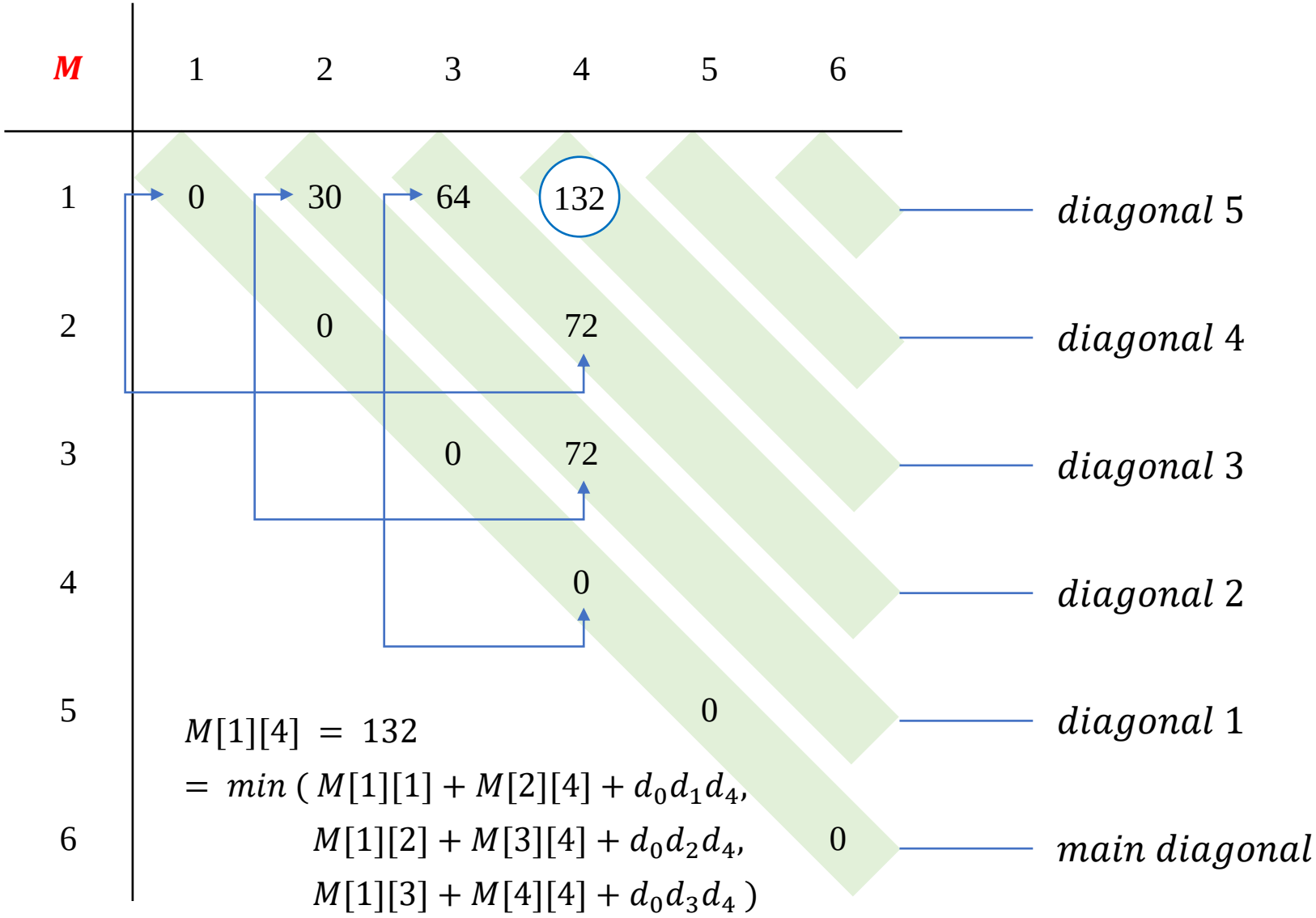
M	1	2	3	4	5	6
1	0					
2		0				
3			0			
4				0		392
5					0	
6						0

- $A_4A_5A_6$의 계산
  - $(A_4A_5)A_6$: $d_3d_4d_5 + d_3d_5d_6 = 392$
  - $A_4(A_5A_6)$: $d_4d_5d_6 + d_3d_4d_6 = 528$
- $M[4][6] = \min(392,528) = 392$

3.4 연쇄 행렬 곱셈

- **연쇄 행렬 곱셈의 재귀 관계식**
  - For $1 \leq i \leq j \leq n$,
    - if $i = j, M[i][j] = 0$
    - if $i < j, M[i][j] = \underset{i \leq k \leq j-1}{\text{minimum}}\left(M[i][k] + M[k+1][j] + d_{i-1}d_k d_j\right)$

3.4 연쇄 행렬 곱셈



$$M[1][4] = 132$$
$$= min\,(\,M[1][1] + M[2][4] + d_0 d_1 d_4,$$
$$M[1][2] + M[3][4] + d_0 d_2 d_4,$$
$$M[1][3] + M[4][4] + d_0 d_3 d_4\,)$$

Algorithm 3.6: Chained Matrix Multiplication

```
def minmult (d):
    n = len(d) - 1
    M = [[-1] * (n + 1) for _ in range(n + 1)]
    P = [[-1] * (n + 1) for _ in range(n + 1)]
    for i in range(1, n + 1):
        M[i][i] = 0
    for diagonal in range(1, n):
        for i in range(1, n - diagonal + 1):
            j = i + diagonal
            M[i][j], P[i][j] = minimum(M, d, i, j)
    return M, P
```

Algorithm 3.6: Chained Matrix Multiplication

```
def minimum (M, d, i, j):
    minValue = INF
    minK = 0
    for k in range(i, j):
        value = M[i][k] + M[k + 1][j]
        value += d[i - 1] * d[k] * d[j]
        if (minValue > value):
            minValue = value
            minK = k
    return minValue, minK
```

3.4 연쇄 행렬 곱셈

- **그러면... 곱셈 순서는 어떻게 출력하지?**
  - $P[i][j] = k$ **이면** $(A_i \cdots A_k)(A_{k+1} \cdots A_j)$**로 분할**
  - **재귀 호출: 분할정복!**

P	1	2	3	4	5	6
1		1	1	1	1	1
2			2	3	4	5
3				3	4	5
4					4	5
5						5
6						

- $P[1][6] = 1$: $P[1][1]$ & $P[2][6]$
- $P[2][6] = 5$: $P[2][5]$ & $P[6][6]$
- $P[2][5] = 4$: $P[2][4]$ & $P[5][5]$
- $P[2][4] = 3$: $P[2][3]$ & $P[4][4]$
- $P[2][3] = 2$: $P[2][2]$ & $P[3][3]$

- $(A_1A_2A_3A_4A_5A_6)$
- $(A_1)(A_2A_3A_4A_5A_6)$
- $(A_1)\big((A_2A_3A_4A_5)A_6\big)$
- $(A_1)\Big(\big((A_2A_3A_4)A_5\big)A_6\Big)$
- $(A_1)\Big(\big(((A_2A_3)A_4)A_5\big)A_6\Big)$

Algorithm 3.7: Print Optimal Order

```
def order (P, i, j):
    if (i == j):
        print('A%d'%(i), end='')
    else:
        k = P[i][j]
        print('(', end = '')
        order(P, i, k)
        order(P, k + 1, j)
        print(')', end = '')
```

3.4 연쇄 행렬 곱셈


```
INF = 999
d = [5, 2, 3, 4, 6, 7, 8]
M, P = minmult(d)
print('M = ')
for i in range(1, len(M)):
    print(M[i][1:])
print('P = ')
for i in range(1, len(P)):
    print(P[i][1:])

print('minimum order: ', end = '')
order(P, 1, len(d) - 1)
```

3.4 연쇄 행렬 곱셈


```
M =
[0, 30, 64, 132, 226, 348]
[-1, 0, 24, 72, 156, 268]
[-1, -1, 0, 72, 198, 366]
[-1, -1, -1, 0, 168, 392]
[-1, -1, -1, -1, 0, 336]
[-1, -1, -1, -1, -1, 0]

P =
[-1, 1, 1, 1, 1, 1]
[-1, -1, 2, 3, 4, 5]
[-1, -1, -1, 3, 4, 5]
[-1, -1, -1, -1, 4, 5]
[-1, -1, -1, -1, -1, 5]
[-1, -1, -1, -1, -1, -1]

minimum order: (A1((((A2A3)A4)A5)A6))
```

주니온TV@Youtube

자세히 보면 유익한 코딩 채널

https://bit.ly/2JXXGqz

- 여러분의 **구독**과 **좋아요**는 강의제작에 큰 힘이 됩니다.
- 강의자료 및 소스코드: **구글 드라이브**에서 다운로드
  (다운로드 주소는 영상 하단 설명란 참고)

  https://bit.ly/3fN0q8t