Chapter 5-1

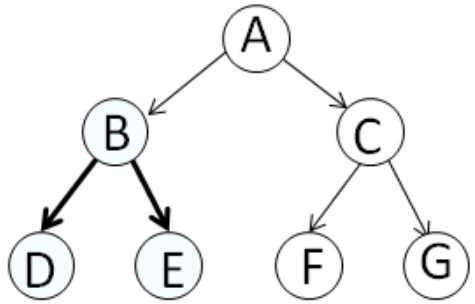
Dynamic Programming

Dynamic Programming

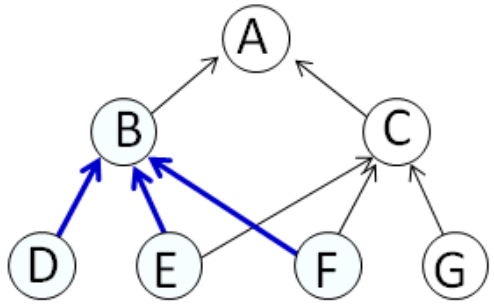
- Dynamic Programming (DP) Algorithm
  - 입력 크기가 작은 부분 문제들을 해결한 후에,
  - 그 해들을 이용하여 보다 큰 크기의 부분 문제들을 해결하여,
  - 최종적으로 원래 주어진 입력의 문제를 해결

DP vs Divide and Conquer

- Divide and conquer 알고리즘과 DP 알고리즘의 전형적인 부분 문제들 사이의 관계



분할 정복 알고리즘

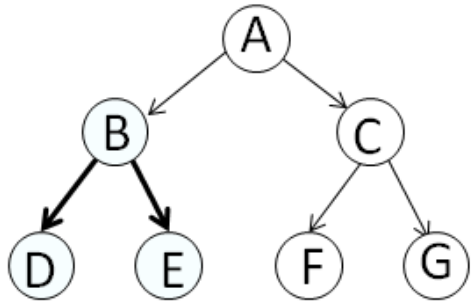


동적 계획 알고리즘

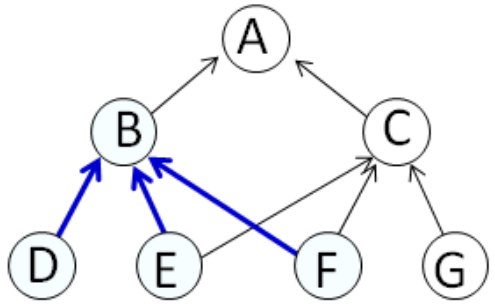
- Divide and conquer 알고리즘
  - A는 B와 C로 분할되고, B는 D와 E로 분할되는데, D와 E의 해를 취합하여 B의 해를 구함
    - 단, D, E, F, G는 각각 더 이상 분할할 수 없는 (또는 가장 작은 크기의) 부분 문제들임
  - 마찬가지로 F와 G의 해를 취합하여 C의 해를 구하고, 마지막으로 B와 C의 해를 취합하여 A의 해를 구함

DP vs Divide and Conquer

- Divide and conquer 알고리즘과 DP 알고리즘의 전형적인 부분 문제들 사이의 관계



분할 정복 알고리즘



동적 계획 알고리즘

- DP 알고리즘
  - 먼저 최소 단위의 부분 문제 D, E, F, G의 해를 각각 구함
  - 그 다음 D, E, F의 해를 이용하여 B의 해를 구함
  - E, F, G의 해를 이용하여 C의 해를 구함
  - B와 C의 해를 계산하는데 E와 F의 해 모두를 이용함

Dynamic Programming Algorithm

- DP 알고리즘에는 부분 문제들 사이에 의존적 관계가 존재
  - 작은 부분 문제의 해가 보다 큰 부분 문제를 해결하는데 사용되는 관계가 있음
  - 이러한 관계는 문제 또는 입력에 따라 다르고, 대부분의 경우 뚜렷이 보이지 않아서 '함축적 순서 ' (implicit order)라고 함

- Divide and conquer 알고리즘은 부분 문제의 해를 중복 사용하지 않음

All Pairs Shortest Paths 문제

- 모든 쌍 최단 경로 (All Pairs Shortest Paths) 문제
  - 각 쌍의 점 사이의 최단 경로를 찾는 문제

	서 울 Seoul	인 천 Incheon	수 원 Suwon	대 전 Daejeon	전 주 Jeonju	광 주 Gwangju	대 구 Daegu	울 산 Ulsan	부 산 Busan
서 울 Seoul		40.2	41.3	154	232.1	320.4	297	407.5	432
인 천 Incheon			54.5	174	253.3	351.6	317.6	447	453
수 원 Suwon				132.6	189.4	299.6	268.1	356	390.7
대 전 Daejeon					96.9	185,2	148.7	259.1	283.4
전 주 Jeonju						105.9	219.7	331.1	322.9
광 주 Gwangju							219.3	329.9	268
대 구 Daegu								111.1	135.5
울 산 Ulsan									52.9
부 산 Busan									

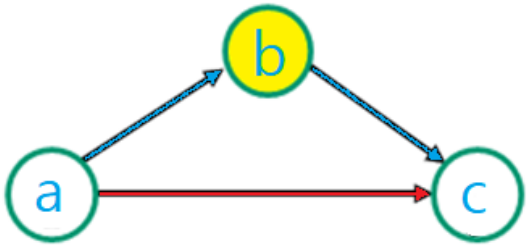
- 다익스트라의 최단 경로 알고리즘을 사용하면
  - 각 점을 시작점으로 정하여 다익스트라 알고리즘 수행
  - 시간 복잡도는 n X $O(n^2)$ = $O(n^3)$, 단, n은 점의 수

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm
  - Floyd 알고리즘의 시간 복잡도는 $O(n^3)$으로 Dijkstra 알고리즘을 n번 사용할 때의 시간 복잡도와 동일
  - Floyd 알고리즘은 매우 간단하여 Dijkstra 알고리즘보다 효율적

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm의 아이디어
  - 작은 그래프에서의 부분 문제
    - 3개의 점이 있는 경우, a에서 c까지의 최단 경로를 찾으려면 2가지 경로, 즉, a에서 c로 직접 가는 경로와 점 b를 경유하는 경로 중에서 짧은 것을 선택



  - 경유 가능한 점
    - 점 1인 경우,
    - 점 1, 2인 경우,
    - 점 1, 2, 3인 경우
    - …
    - 점 1, 2, …, n, 즉, 모든 점을 경유 가능한 점들로 고려하면서 모든 쌍의 최단 경로의 거리를 계산

All Pairs Shortest Paths 문제

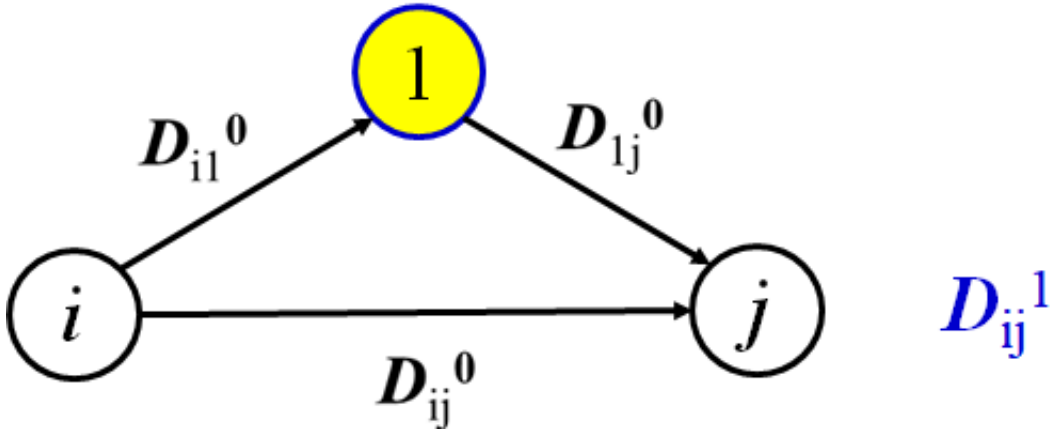
- Floyd-Warshall Algorithm의 부분 문제 정의
  - 그래프의 점이 1, 2, 3, …, n일 때

    D_{ij}^{k} = 점 {1, 2, …, k}를 경유 가능한 점으로 고려하여, 점 i에서 점 j까지의 모든 경로 중에서 가장 짧은 경로의 거리

    - [주의] 점 1에서 점 k까지의 모든 점을 반드시 경유하는 경로를 의미하는 것이 아님
    - D_{ij}^{k} 는 {1, 2, …, k}을 하나도 경유하지 않으면서 점 i에서 직접 점 j에 도달하는 간선 (i, j)가 가장 짧은 거리일 수도 있음
    - 단, k≠i, k≠j이고 k=0인 경우, 점 0은 그래프에 없으므로 어떤 점도 경유하지 않는다는 것을 의미
    - 즉, D_{ij}^{0} = 간선 (i, j)의 가중치

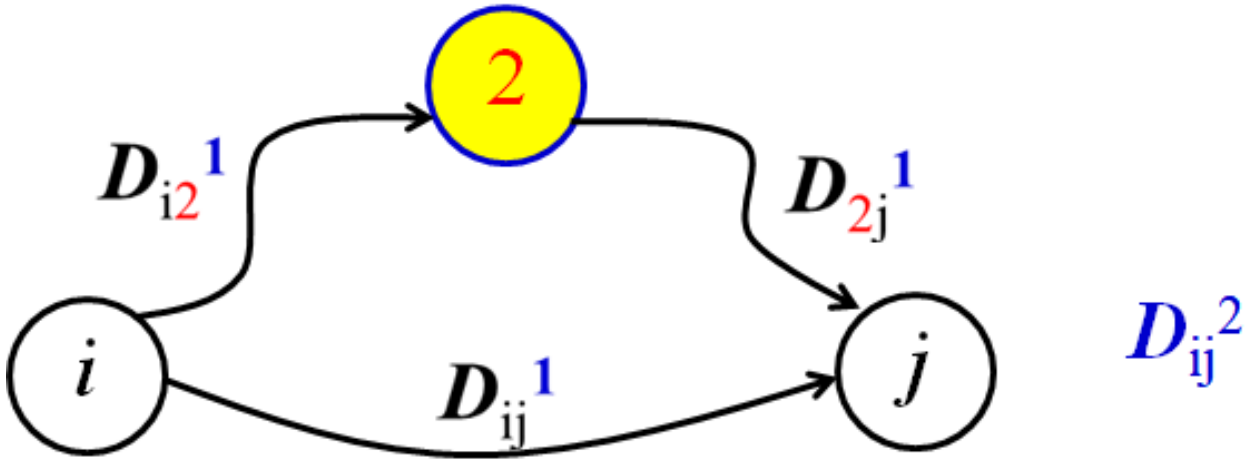
All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm의 부분 문제 (D_{ij}^{1})
  - 점 i에서 점 j까지 점 1을 경유하는 경우와 직접 가는 경로 중에서 짧은 경로의 거리
  - 모든 점 i와 j에 대해 D_{ij}^{1} 를 계산하는 것이 가장 작은 부분 문제
  - i≠1, j≠1



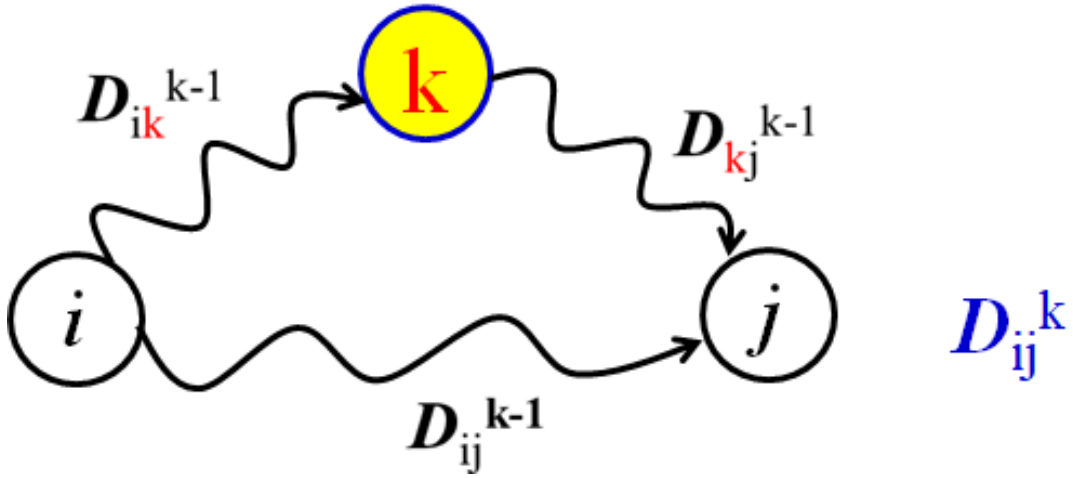
All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm의 부분 문제 (D_{ij}^{2})
  - 점 i에서 점 j까지 점 2를 경유하는 경로와 D_{ij}^{1} 중에서 짧은 경로의 거리
  - 단, 점 2를 경유하는 경로의 거리는 $D_{i2}^{1} + D_{2j}^{1}$
  - i≠2, j≠2



All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm의 부분 문제 (D_{ij}^{k})
  - 점 i에서 점 j까지 점 k를 경유하는 경로와 D_{ij}^{k-1} 중에서 짧은 경로의 거리
  - 점 k를 경유하는 경로의 거리는 $D_{ik}^{k-1} + D_{kj}^{k-1}$
  - i≠k, j≠k



All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm의 부분 문제 (D_{ij}^{n})
  - 모든 점을 경유 가능한 점들로 고려한 모든 쌍 i와 j의 최단 경로의 거리
  - Floyd의 모든 쌍 최단 경로 알고리즘은 k가 1에서 n이 될 때까지 D_{ij}^{k} 를 계산해서 D_{ij}^{n} 를 찾음

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm

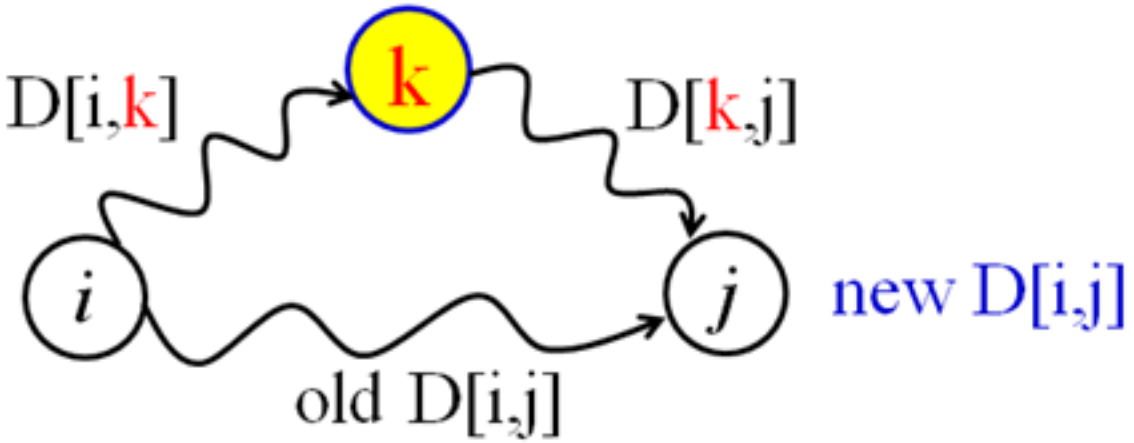
AllPairsShortest

- 입력: 2차원 배열 D, 단, D[i, j] = 간선 (i, j)의 가중치, 만일 간선 (i, j)가 없으면 D[i, j] = ∞, 모든 i에 대하여 D[i, i] = 0
- 출력: 모든 쌍 최단 경로의 거리를 저장한 2차원 배열 D

```
1.  for k = 1 to n
2.      for i = 1 to n (i≠k)
3.          for j = 1 to n (j≠k, j≠i)
4.              D[i, j] = min{ D[i, k] + D[k, j], D[i, j] }
```

All Pairs Shortest Paths 문제

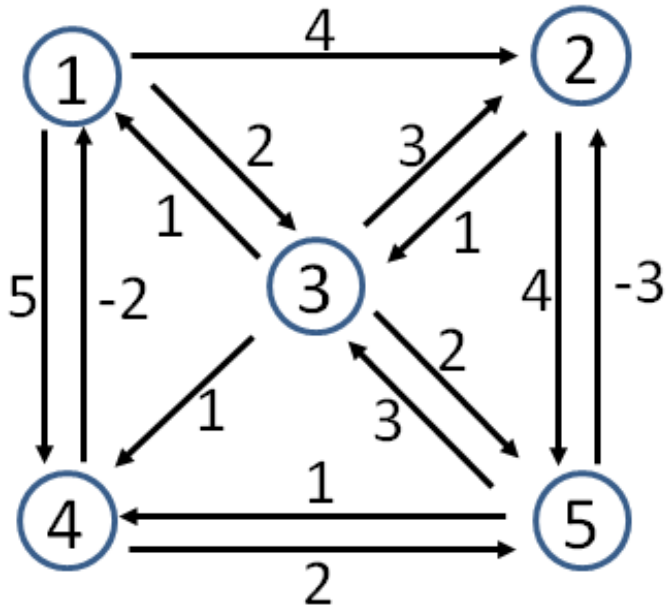
- Floyd-Warshall Algorithm (부분 문제 간의 함축적 순서)



  - D[i, j]를 계산하기 위해서 미리 계산되어 있어야 할 부분 문제는 D[i, k]와 D[k, j]임

All Pairs Shortest Paths 문제

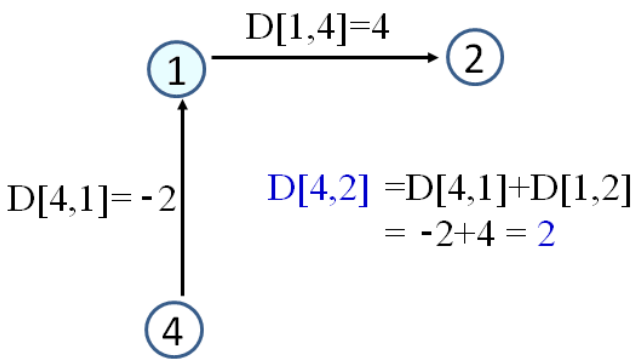
- Floyd-Warshall Algorithm 수행 과정

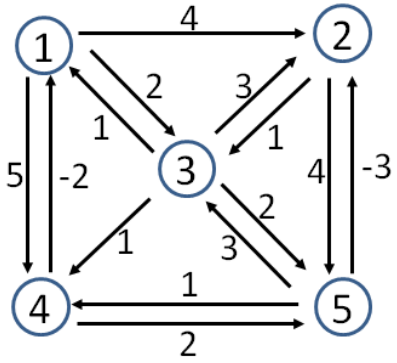


D	1	2	3	4	5
1	0	4	2	5	∞
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	∞	∞	0	2
5	∞	-3	3	1	0

All Pairs Shortest Paths 문제

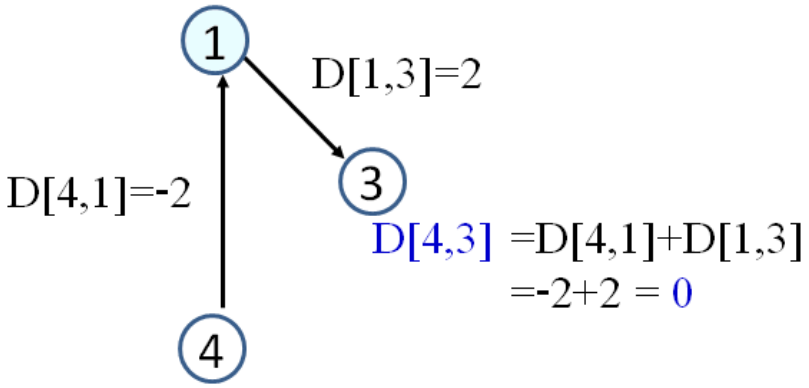
- Floyd-Warshall Algorithm 수행 과정 (k = 1 일 때)
  - D[2, 3] = min{ D[2, 3], D[2, 1]+D[1, 3] } = min{1, ∞+2} = 1
  - D[2, 4] = min{ D[2, 4], D[2, 1]+D[1, 4] } = min{∞, ∞+5} = ∞
  - D[2, 5] = min{ D[2, 5], D[2, 1]+D[1, 5] } = min{4, ∞+ ∞} = 4
  - D[3, 2] = min{ D[3, 2], D[3, 1]+D[1, 2] } = min{3, 1+4} = 3
  - D[3, 4] = min{ D[3, 4], D[3, 1]+D[1, 4] } = min{1, 1+5} = 1
  - D[3, 5] = min{ D[3, 5], D[3, 1]+D[1, 5] } = min{2, 1+∞} = 2
  - D[4, 2] = min{ D[4, 2], D[4, 1]+D[1, 2] } = min{∞, -2+4} = 2 // 갱신됨

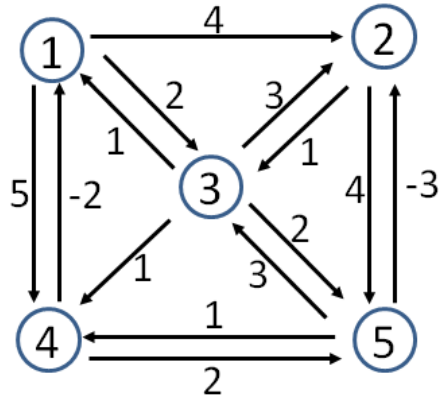




All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 1 일 때)
  - D[4, 3] = min{ D[4, 3], D[4, 1]+D[1, 3] } = min{∞, -2+2} = 0 // 갱신됨



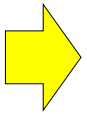


- D[4, 5] = min{ D[4, 5], D[4, 1]+D[1, 5] } = min{2, -2+∞} = 2
- D[5, 2] = min{ D[5, 2], D[5, 1]+D[1, 2] } = min{-3, ∞+4} = -3
- D[5, 3] = min{ D[5, 3], D[5, 1]+D[1, 3] } = min{3, ∞+2} = 3
- D[5, 4] = min{ D[5, 4], D[5, 1]+D[1, 4] } = min{1, ∞+5} = 1

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 결과 (k = 1 일 때)
  - k=1일 때, D[4, 2], D[4, 3]이 각각 2, 0으로 갱신, 다른 원소들은 변하지 않음

D	1	2	3	4	5
1	0	4	2	5	∞
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	∞	∞	0	2
5	∞	-3	3	1	0



D	1	2	3	4	5
1	0	4	2	5	∞
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	2	0	0	2
5	∞	-3	3	1	0

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 2 일 때)
  - D[1, 3] = min{ D[1, 3], D[1, 2]+D[2, 3] } = min{2, 4+1} = 2
  - D[1, 4] = min{ D[1, 4], D[1, 2]+D[2, 4] } = min{5, 4+∞} = 5
  - D[1, 5] = min{ D[1, 5], D[1, 2]+D[2, 5] } = min{∞, 4+4} = 8 // 갱신됨
  - D[3, 1] = min{ D[3, 1], D[3, 2]+D[2, 1] } = min{1, 3+∞} = 1
  - D[3, 4] = min{ D[3, 4], D[3, 2]+D[2, 4] } = min{1, 3+∞} = 1
  - D[3, 5] = min{ D[3, 5], D[3, 2]+D[2, 5] } = min{2, 3+4} = 2
  - D[4, 1] = min{ D[4, 1], D[4, 2]+D[2, 1] } = min{-2, 2+∞} = -2
  - D[4, 3] = min{ D[4, 3], D[4, 2]+D[2, 3] } = min{0, 2+1} = 0
  - D[4, 5] = min{ D[4, 5], D[4, 2]+D[2, 5] } = min{2, 2+4} = 2

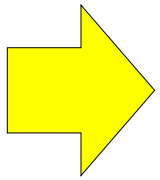
All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 2 일 때)
  - D[5, 1] = min{ D[5, 1], D[5, 2]+D[2, 1] } = min{∞, -3+∞} = ∞
  - D[5, 3] = min{ D[5, 3], D[5, 2]+D[2, 3] } = min{3, -3+1} = -2 // 갱신됨
  - D[5, 4] = min{ D[5, 4], D[5, 2]+D[2, 4] } = min{1, -3+∞} = 1

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 결과 (k = 2 일 때)
  - D[1, 5]가 1 -> 2 -> 5의 거리인 8로 갱신
  - D[5, 3]이 5 -> 2 -> 3의 거리인 -2로 갱신

D	1	2	3	4	5
1	0	4	2	5	∞
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	2	0	0	2
5	∞	-3	3	1	0



D	1	2	3	4	5
1	0	4	2	5	8
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	2	0	0	2
5	∞	-3	-2	1	0

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 3 일 때)
  - D[1, 2] = min{ D[1, 2], D[1, 3]+D[3, 2] } = min{4, 2+3} = 4
  - D[1, 4] = min{ D[1, 4], D[1, 3]+D[3, 4] } = min{5, 2+1} = 3 // 갱신됨
  - D[1, 5] = min{ D[1, 5], D[1, 3]+D[3, 5] } = min{8, 2+2} = 4 // 갱신됨
  - D[2, 1] = min{ D[2, 1], D[2, 3]+D[3, 1] } = min{∞, 1+1} = 2 // 갱신됨
  - D[2, 4] = min{ D[2, 4], D[2, 3]+D[3, 4] } = min{∞, 1+1} = 2 // 갱신됨
  - D[2, 5] = min{ D[2, 5], D[2, 3]+D[3, 5] } = min{4, 1+2} = 3 // 갱신됨
  - D[4, 1] = min{ D[4, 1], D[4, 3]+D[3, 1] } = min{-2, 0+1} = -2
  - D[4, 2] = min{ D[4, 2], D[4, 3]+D[3, 2] } = min{2, 0+3} = 2
  - D[4, 5] = min{ D[4, 5], D[4, 3]+D[3, 5] } = min{2, 0+2} = 2

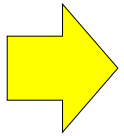
All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 3 일 때)
  - D[5, 1] = min{ D[5, 1], D[5, 3]+D[3, 1] } = min{∞, -2+1} = -1 // 갱신됨
  - D[5, 2] = min{ D[5, 2], D[5, 3]+D[3, 2] } = min{-3, -2+3} = -3
  - D[5, 4] = min{ D[5, 4], D[5, 3]+D[3, 4] } = min{1, -2+1} = -1 // 갱신됨

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 결과 (k = 3 일 때)
  - 총 7개의 원소가 갱신

D	1	2	3	4	5
1	0	4	2	5	8
2	∞	0	1	∞	4
3	1	3	0	1	2
4	-2	2	0	0	2
5	∞	-3	-2	1	0



D	1	2	3	4	5
1	0	4	2	3	4
2	2	0	1	2	3
3	1	3	0	1	2
4	-2	2	0	0	2
5	-1	-3	-2	-1	0

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 4 일 때)
  - D[1, 2] = min{ D[1, 2], D[1, 4]+D[4, 2] } = min{4, 3+2} = 4
  - D[1, 3] = min{ D[1, 3], D[1, 4]+D[4, 3] } = min{2, 3+0} = 2
  - D[1, 5] = min{ D[1, 5], D[1, 4]+D[4, 5] } = min{4, 3+2} = 4
  - D[2, 1] = min{ D[2, 1], D[2, 4]+D[4, 1] } = min{2, 2-2} = 0 // 갱신됨
  - D[2, 3] = min{ D[2, 3], D[2, 4]+D[4, 3] } = min{1, 2+0} = 1
  - D[2, 5] = min{ D[2, 5], D[2, 4]+D[4, 5] } = min{3, 2+2} = 3
  - D[3, 1] = min{ D[3, 1], D[3, 4]+D[4, 1] } = min{1, 1-2} = -1 // 갱신됨
  - D[3, 2] = min{ D[3, 2], D[3, 4]+D[4, 2] } = min{3, 1+2} = 3
  - D[3, 5] = min{ D[3, 5], D[3, 4]+D[4, 5] } = min{2, 1+2} = 2

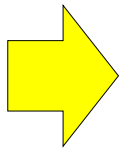
All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 4 일 때)
  - D[5, 1] = min{ D[5, 1], D[5, 4]+D[4, 1] } = min{-1, -1-2} = -3 // 갱신됨
  - D[5, 2] = min{ D[5, 2], D[5, 4]+D[4, 2] } = min{-3, -1+2} = -3
  - D[5, 3] = min{ D[5, 3], D[5, 4]+D[4, 3] } = min{-2, -1+0} = -2

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 결과 (k = 4 일 때)
  - 총 3개의 원소가 갱신

D	1	2	3	4	5
1	0	4	2	3	4
2	2	0	1	2	3
3	1	3	0	1	2
4	-2	2	0	0	2
5	-1	-3	-2	-1	0



D	1	2	3	4	5
1	0	4	2	3	4
2	0	0	1	2	3
3	-1	3	0	1	2
4	-2	2	0	0	2
5	-3	-3	-2	-1	0

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 5 일 때)
  - D[1, 2] = min{ D[1, 2], D[1, 5]+D[5, 2] } = min{4, 4-3} = 1 // 갱신됨
  - D[1, 3] = min{ D[1, 3], D[1, 5]+D[5, 3] } = min{2, 4-2} = 2
  - D[1, 4] = min{ D[1, 4], D[1, 5]+D[5, 4] } = min{3, 4-1} = 3
  - D[2, 1] = min{ D[2, 1], D[2, 5]+D[5, 1] } = min{0, 3-3} = 0
  - D[2, 3] = min{ D[2, 3], D[2, 5]+D[5, 3] } = min{1, 3-2} = 1
  - D[2, 4] = min{ D[2, 4], D[2, 5]+D[5, 4] } = min{2, 3-1} = 2
  - D[3, 1] = min{ D[3, 1], D[3, 5]+D[5, 1] } = min{-1, 2-3} = -1
  - D[3, 2] = min{ D[3, 2], D[3, 5]+D[5, 2] } = min{3, 2-3} = -1 // 갱신됨
  - D[3, 4] = min{ D[3, 4], D[3, 5]+D[5, 4] } = min{1, 2-1} = 1

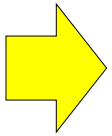
All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 과정 (k = 5 일 때)
  - D[4, 1] = min{ D[4, 1], D[4, 5]+D[5, 1] } = min{-2, 2-3} = -2
  - D[4, 2] = min{ D[4, 2], D[4, 5]+D[5, 2] } = min{2, 2-3} = -1 // 갱신됨
  - D[4, 3] = min{ D[4, 3], D[4, 5]+D[5, 3] } = min{0, 2-2} = 0

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 수행 결과 (k = 5 일 때)
  - 총 3개의 원소가 갱신되고, 이것이 주어진 입력에 대한 최종 해

D	1	2	3	4	5
1	0	4	2	3	4
2	0	0	1	2	3
3	-1	3	0	1	2
4	-2	2	0	0	2
5	-3	-3	-2	-1	0



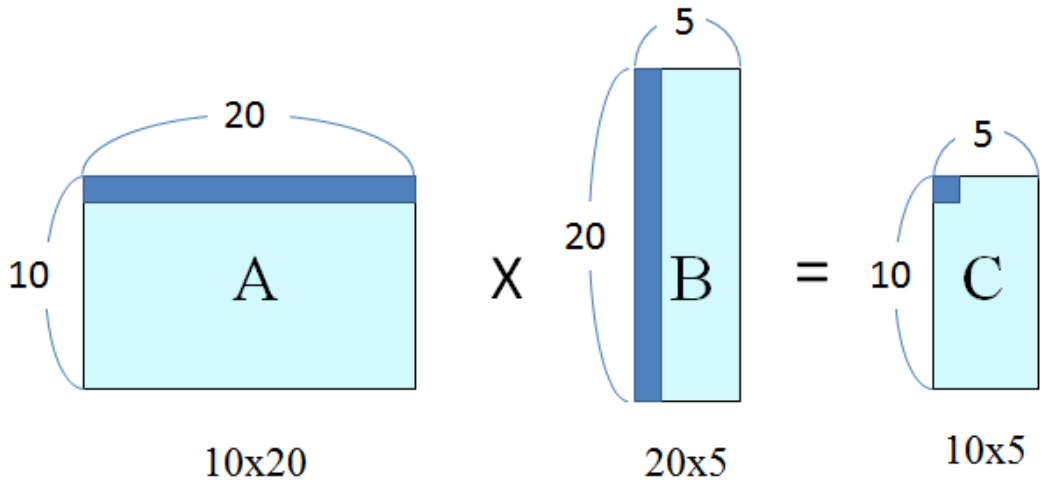
D	1	2	3	4	5
1	0	1	2	3	4
2	0	0	1	2	3
3	-1	-1	0	1	2
4	-2	-1	0	0	2
5	-3	-3	-2	-1	0

All Pairs Shortest Paths 문제

- Floyd-Warshall Algorithm 시간 복잡도
  - 각 k에 대해서 모든 i, j 쌍에 대해 계산되므로, 총 n X n X n = n^3회 계산이 이루어지고, 각 계산은 O(1) 시간 소요
  - 시간 복잡도는 $O(n^3)$

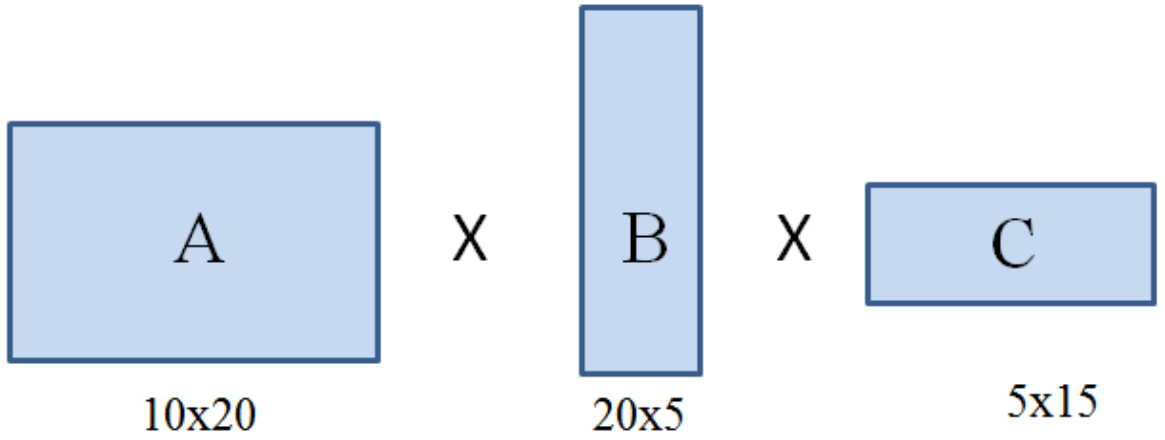
Chained Matrix Multiplications 문제

- 연속 행렬 곱셈 (Chained Matrix Multiplications) 문제
  - 연속된 행렬들의 곱셈에 필요한 원소 간의 최소 곱셈 횟수를 찾는 문제
  - 10 X 20 행렬 A와 20 X 5 행렬 B를 곱하는데 원소 간의 곱셈 횟수는 10 X 20 X 5 = 1,000
  - 두 행렬을 곱한 결과, 행렬 C는 10 X 5



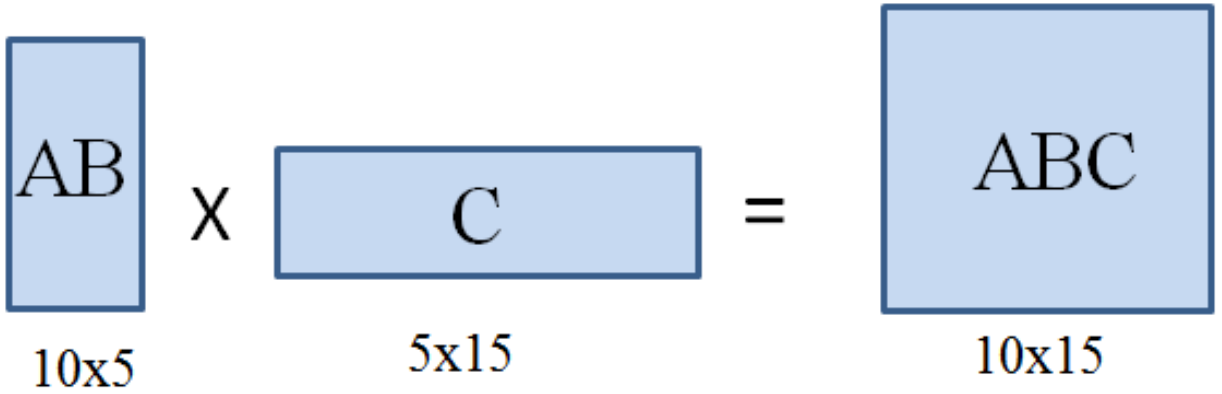
Chained Matrix Multiplications 문제

- 행렬 곱셈
  - 3개의 행렬을 곱해야 하는 경우
  - 연속된 행렬의 곱셈에는 결합 법칙 허용
  - A X B X C = (A X B) X C = A X (B X C)



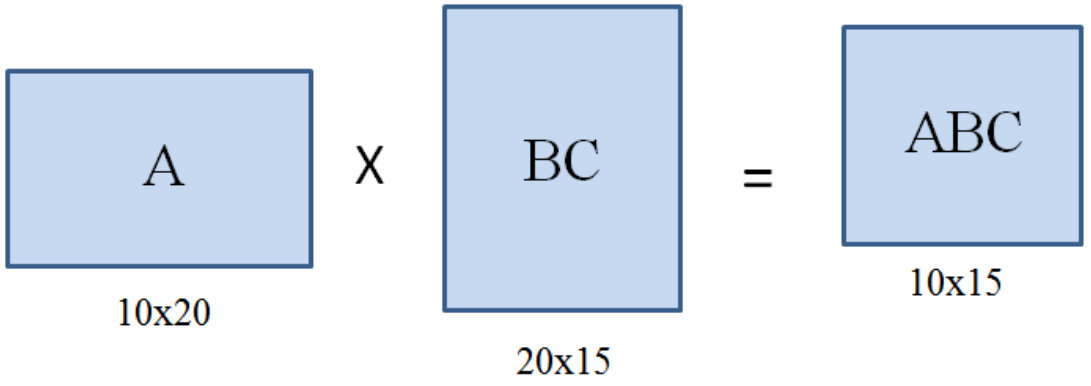
Chained Matrix Multiplications 문제

- A X B를 계산한 후에 C를 곱하기
  - A X B를 계산하는 데, 10 X 20 X 5 = 1,000 번
  - 결과 행렬의 크기가 10 X 5이고, 이에 행렬 C를 곱하면 10 X 5 X 15 = 750 번
  - 1,000 + 750 = 1,750 회의 원소의 곱셈 필요



Chained Matrix Multiplications 문제

- B X C를 계산한 후에 A를 곱하기
  - B X C를 계산하는 데, 20 X 5 X 15 = 1,500 번
  - 그 결과 20 X 15 행렬이 만들어지고, 이를 행렬 A와 곱하면 10 X 20 X 15 = 3,000 번
  - 1,500 + 3,000 = 4,500 회의 곱셈 필요
  - 곱셈의 순서에 따라 결과가 달라짐 => 곱셈 횟수 최소화를 위한 곱셈 순서 발견 필요



Chained Matrix Multiplications 문제

- [주의] 주어진 행렬의 순서를 지켜서 반드시 이웃하는 행렬끼리 곱해야 함
  - A X B X C X D X E 일 때,
  - A X C를 수행하거나, A X D를 먼저 수행할 수 없음

Chained Matrix Multiplications 문제

- 부분 문제

부분 문제 크기		부분 문제 개수
1	A B C D E	5개
2	AxB BxC CxD DxE	4개
3	AxBxC BxCxD CxDxE	3개
4	AxBxCxD BxCxDxE	2개
5	A x B x C x D x E	1개

Chained Matrix Multiplications 문제

- 알고리즘

입력: 연속된 행렬 $A_1 \times A_2 \times \cdots \times A_n$, 단, A_1은 $d_0 \times d_1$, A_2는 $d_1 \times d_2$, ⋯, A_n은 $d_{n-1} \times d_n$ 임

출력: 입력의 행렬 곱셈에 필요한 원소의 최소 곱셈 횟수

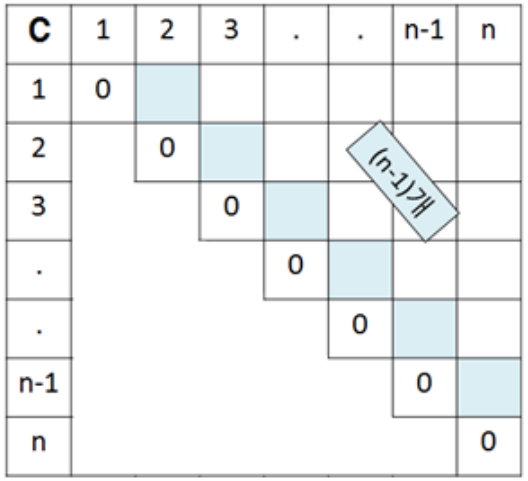
```
1.  for i = 1 to n
2.      C[i, i] = 0
3.  for L = 1 to n-1        // L은 부분 문제의 크기를 조절하는 인덱스
4.      for i = 1 to n-L
5.          j = i + L
6.          C[i, j] = ∞
7.          for k = i to j-1
8.              temp = C[i, k] + C[k+1, j] + d_{i-1} d_k d_j
9.              if (temp < C[i, j])
10.                 C[i, j] = temp
11. Return C[1, n]
```

C[i, j]: $A_i \times A_{i+1} \times \cdots \times A_j$ 에 필요한 원소 간의 최소 곱셈 횟수

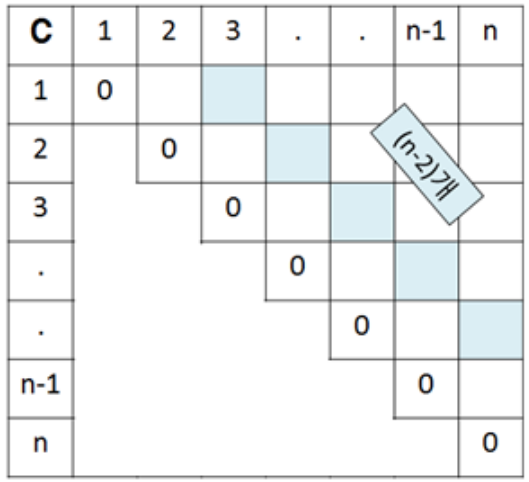
Chained Matrix Multiplications 문제

- Line 3의 for-loop

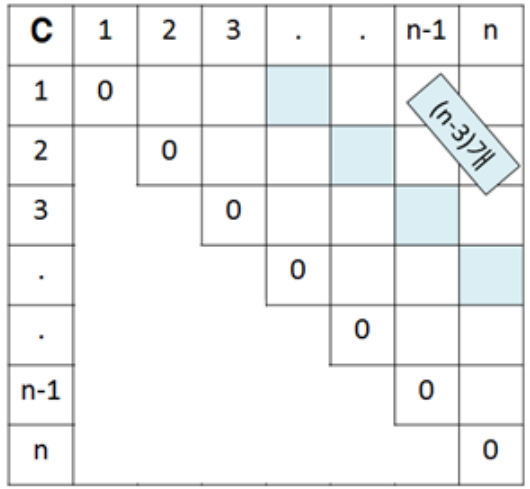
L = 1



L = 2

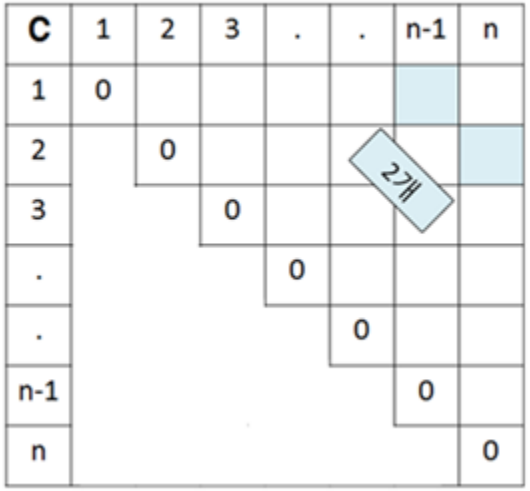


L = 3

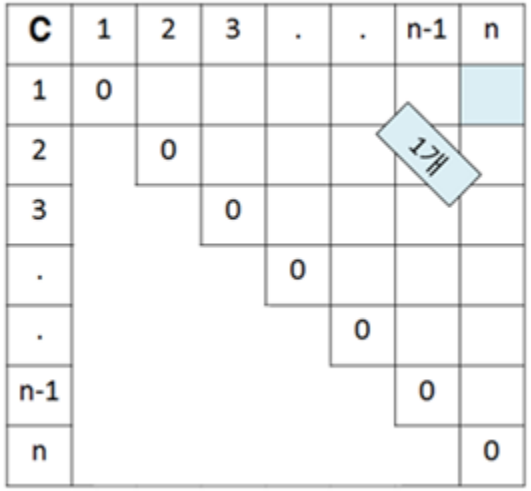


...

L = n-2



L = n-1



Chained Matrix Multiplications 문제

- Line 7의 for-loop

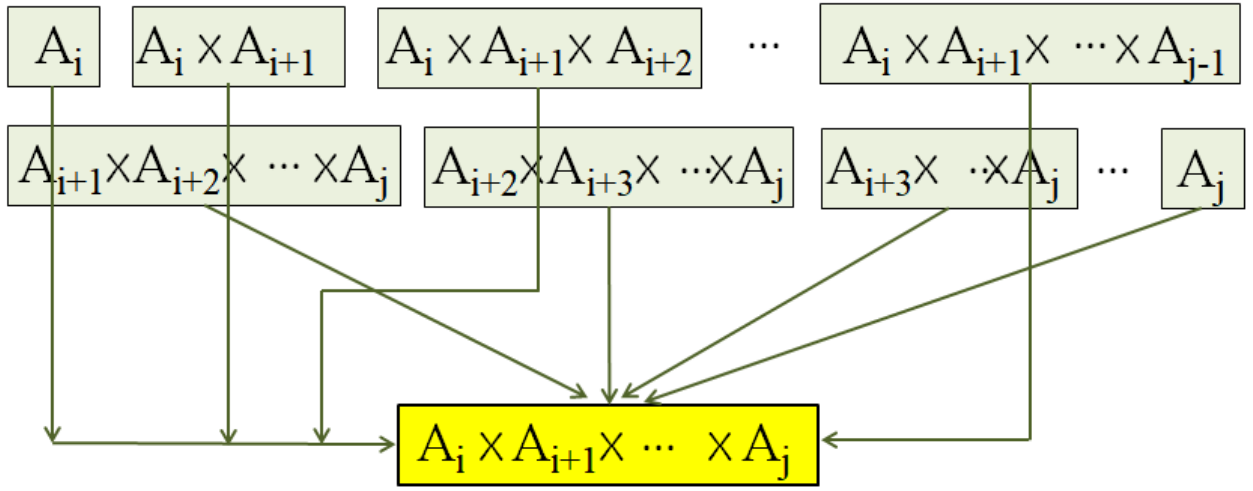
$(A_i) \times (A_{i+1} \times A_{i+2} \times \cdots \times A_j)$ k=i 일때

$(A_i \times A_{i+1}) \times (A_{i+2} \times A_{i+3} \times \cdots \times A_j)$ k=i+1 일때

$(A_i \times A_{i+1} \times A_{i+2}) \times (A_{i+3} \times \cdots \times A_j)$ k=i+2 일때

: :

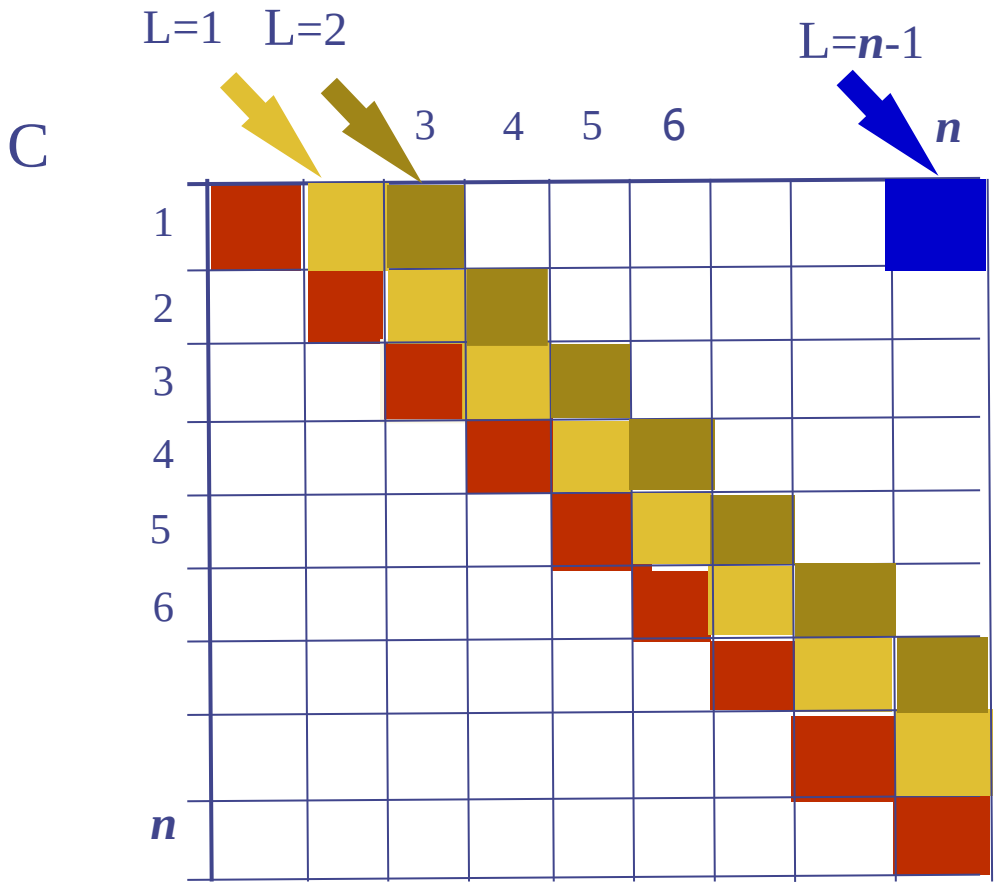
$(A_i \times A_{i+1} \times \cdots \times A_{j-1}) \times (A_j)$ k=j-1 일때



함축적 순서

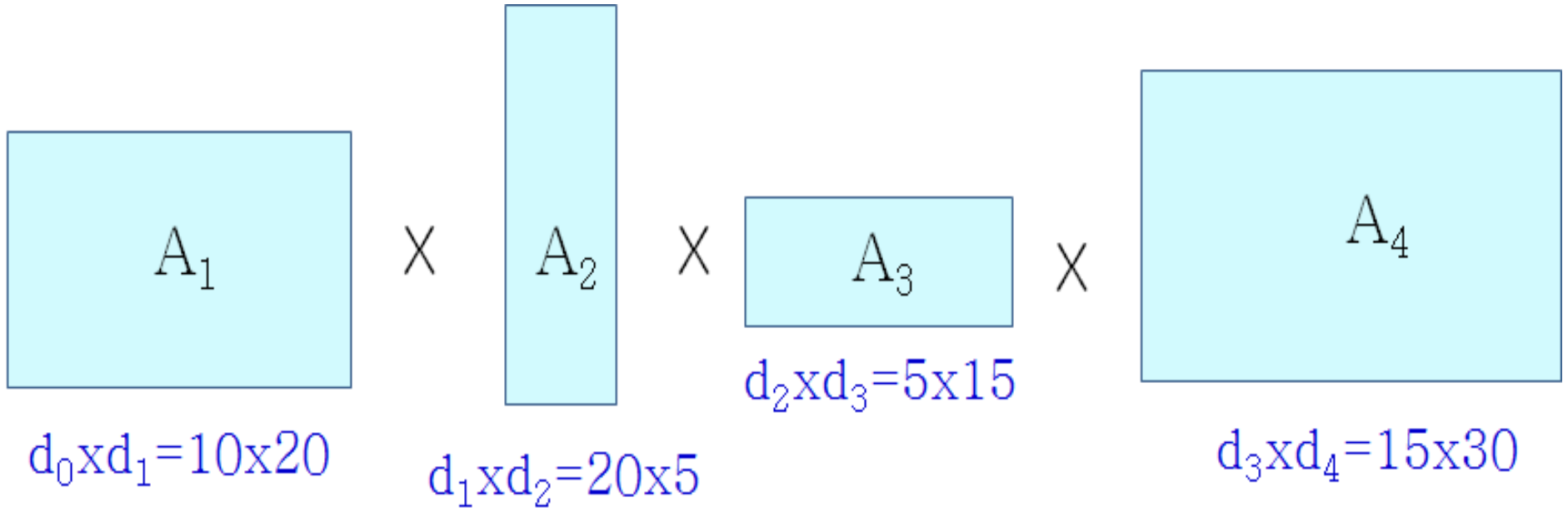
Chained Matrix Multiplications 문제

- 알고리즘 수행 순서



Chained Matrix Multiplications 문제

- 알고리즘 수행 과정



Chained Matrix Multiplications 문제

- 알고리즘 수행 과정 (L = 1 일 때)
  - C[1, 2] = $d_0d_1d_2$ = 10 X 20 X 5 = 1,000
  - C[2, 3] = 20 X 5 X 15 = 1,500
  - C[3, 4] = 5 X 15 X 30 = 2,250

i = 1

i = 2

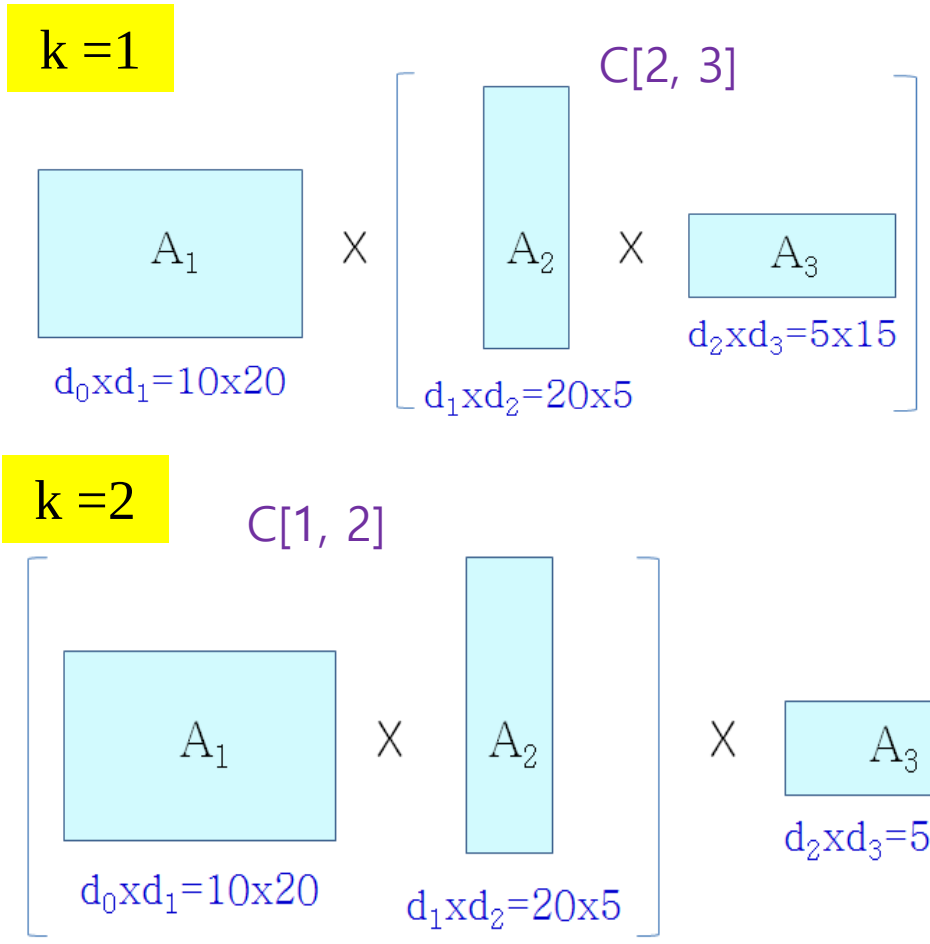
i = 3

L = 1

C	1	2	3	4
1	0			
2		0		
3			0	
4				0

Chained Matrix Multiplications 문제

- 알고리즘 수행 과정 (L = 2, i = 1 일 때)
  - $A_1A_2A_3$ 을 계산한다. C[1, 3] = 1,750



L = 2

C	1	2	3	4
1	0			
2		0		
3			0	
4				0

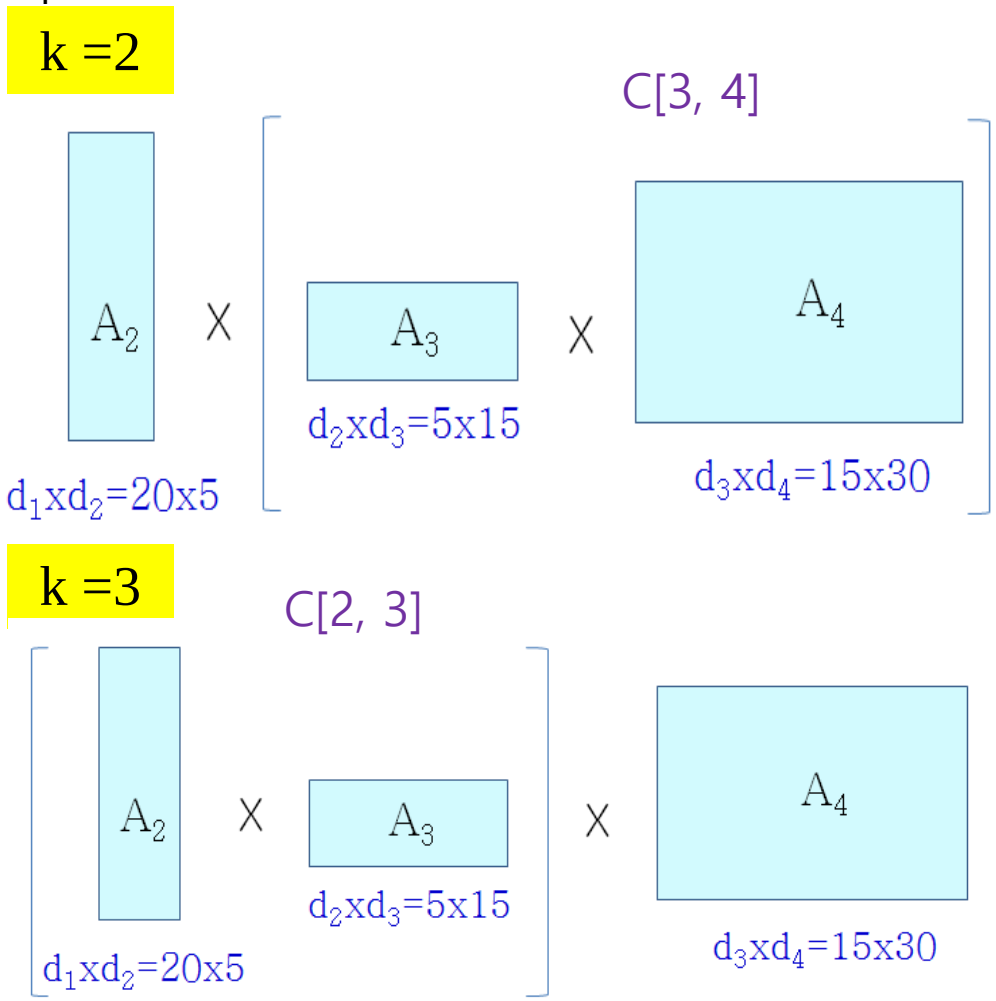
4,500

1,750

1,750이 4,500 보다 작으므로

Chained Matrix Multiplications 문제

- 알고리즘 수행 과정 (L = 2, i = 2 일 때)
  - $A_2A_3A_4$ 를 계산한다. C[2, 4] = 5,250



L = 2

C	1	2	3	4
1	0			
2		0		
3			0	
4				0

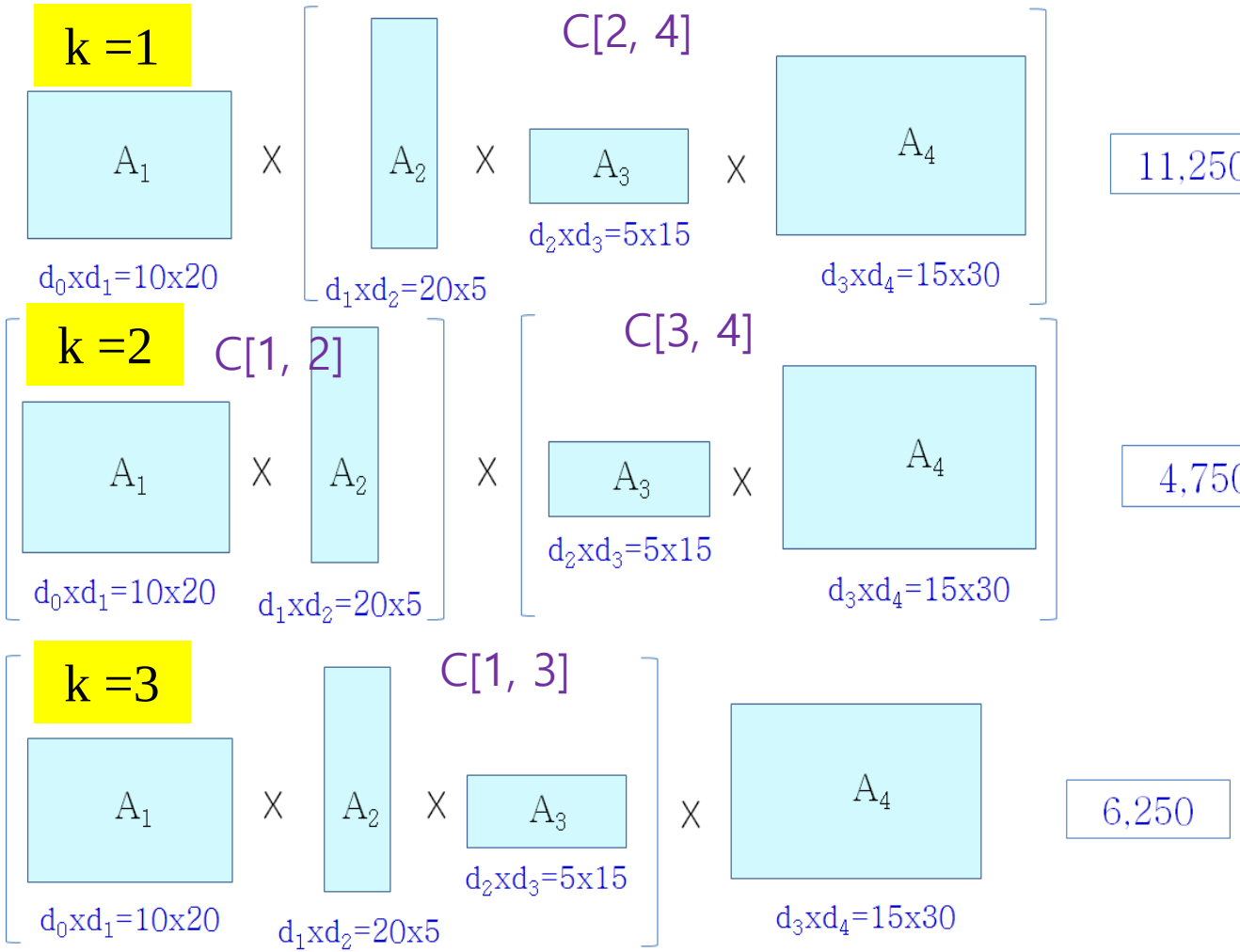
5,250

5,250이 10,500 보다 작으므로

10,500

Chained Matrix Multiplications 문제

- 알고리즘 수행 과정 (L = 3 일 때)
  - A_1 X A_2 X A_3 X A_4 를 계산한다. C[1, 4] = 4,750



L = 3

C	1	2	3	4
1	0			
2		0		
3			0	
4				0

가장 작음

Chained Matrix Multiplications 문제

- 알고리즘 수행 결과

C	1	2	3	4
1	0	1,000	1,750	4,750
2		0	1,500	5,250
3			0	2,250
4				0

Chained Matrix Multiplications 문제

- Time Complexity
  - 총 부분 문제 수: (n-1)+(n-2)+…+2+1 = n(n-1)/2
  - 하나의 부분 문제는 k-loop가 최대 (n-1)번 수행
  - Time Complexity: $O(n^2)$ X O(n) = $O(n^3)$