연쇄행렬곱셈 알고리즘

보통 행렬을 곱셈할 때는 슈트라센 알고리즘이 아닌이상 $O(n^3)$의 시간복잡도가 나온다. 슈트라센 알고리즘으로 풀면 $O(n^{2.78})$정도까지 줄일 수는 있다고 하는데 너무 복잡해서 보통의 방법으로 하는 경우가 많다. 하지만 행렬이 2개가 아니라 3개 이상인 경우에 대해 곱셈을 할 때 곱셈 순서에 따라서 곱셈 횟수를 다르게 할 수 있다.

DP(Dynamic Programming)을 활용한 알고리즘으로 optimal substructure만 알아낸다면 subproblems의 optimal solution을 통해서 bottom-up 방식으로 해결할 수 있다. 보통 DP 문제를 해결할 때는 Recurrence equation을 구해내면 그나마 알고리즘을 구현하는 것은 쉽다. 점화식을 구해내는게 상당히 어려울 뿐이다. 어쨌든 연쇄행렬곱셈 알고리즘의 점화식을 보도록 하자.

$$m[i,j] = \begin{cases} 0 & \text{if i=j} \\ min_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & \text{if i<j} \end{cases}$$

m[i,j]의 의미는 행렬 A_i부터 A_j까지 곱하는데 드는 최소 곱셈의 횟수를 의미한다.
i=j일 경우는 당연히 행렬이 1개인 경우이므로 곱셈은 0번한다. 다음으로 i<j인 경우의 점화식은 가장 적은 곱셈의 횟수를 구하는 것인데 임의의 k를 기준으로 나누어서 계산한다.

$m[i,k]$ **= 행렬** A_i**부터** A_k**까지의 최소 곱셈 횟수**
$m[k+1,j]$ **= 행렬** A_{k+1}**부터** A_j**까지의 최소 곱셈 횟수** $p_{i-1}p_kp_j$ **= 결국 나눠진 2개의 행렬을 곱하기 때문에 그 비용인** A_i**부터** A_j**까지의 곱셈횟수이다.**

예를 들어, $A_1A_2A_3A_4$라는 행렬의 곱셈이 있다고 하자.
크게 총 3가지의 방법, (1, 2~4), (1~2, 3~4), (1~3,4)가 있다. 하나의 경우만 살펴보면,

- (1, 2~4)

여기서 1은 i=j인 경우이므로 곱셈횟수는 0이다. 2~4는 i<j이므로 (2~3, 4)과 (2, 3~4)로 다시 나누어진다.

- (2~3, 4), (2, 3~4)

여기선 더 이상 나눠지지 않고 단순 곱셈이므로 A_2A_3과 A_3A_4를 계산한다. 그리고 바로 이렇게 계산한 결과를 (1, 2~4)에서 사용하는 것이다. 2개의 곱셈결과 중 더 적은 곱셈 횟수로 곱하는 순서를 고르는 개념이다.

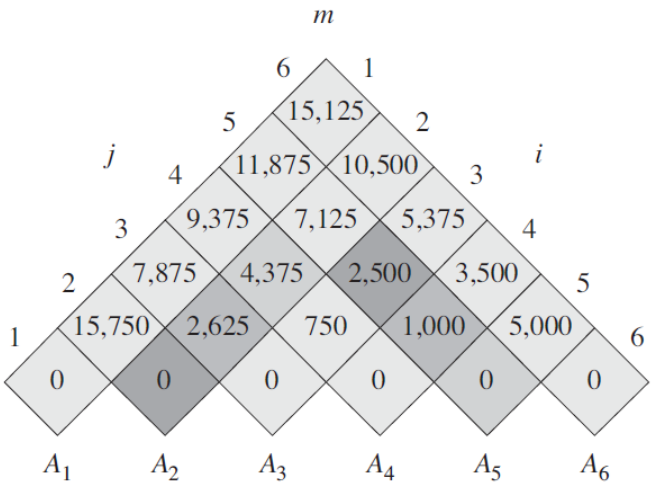


Figure 15.5 The m and s tables computed by MATRIX-CHAIN-ORDER for $n = 6$ and the following matrix dimensions:

matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

다음 그림을 보자. $A_1 \dots A_6$를 곱하는 경우인데 왼쪽부터 보면 A_1A_2는 $30 \times 35 \times 15 = 15,750$이다. 또한 A_2A_3은 $35 \times 15 \times 5 = 2,625$이다. 이 때 $A_1A_2A_3$을 구하는 경우를 보자.

1) 먼저 $p_{i-1}p_kp_j$를 보면 $30 \times 35 \times 5 = 5,250$인 것을 알 수 있다. 2) 이제 $A_1A_2 + 5,250$과 $A_2A_3 + 5,250$ 중에 더 큰 값을 보면 되는 것이다. 3) 각각 $15,750 + 5,250 = 21,000$과 $2,625 + 5,250 = 7,875$이기 때문에 $A_1A_2A_3 = 7,875$가 된다.

이런 방식으로 계속 해결하면 항상 곱셈의 횟수가 최소인 경우로 선택을 하기 때문에 $A_1 \dots A_6$은 최소곱셈횟수가 나오게 된다. 물론 우리의 목적은 최소곱셈횟수가 무엇인지를 알아내는 것이 아니라 그것을 통해서 어떤 순서로 곱셈을 해야 최소로 곱셈을 할 수 있는지를 알아내는 것이다. 따라서 이 알고리즘이 진행될 때 순서를 정해주는 부분을 저장하는 Auxiliary table이 하나 더 있어야 한다. 위의 문제는 $((A_1(A_2A_3))((A_4A_5)A_6))$으로 나눠진다. 위의 그림에서 오른쪽 그림이 나눠지는 기준인 k를 저장하는 Auxiliary table인 것이다.

코드 구현

코드는 꽤 복잡하다. 책에서 나온 pseudo code를 통해서 구현이 되었지만 한번에 이해하기는 좀 어렵다.

```
// 1. n개의 행렬을 곱한다고 하면 그 안에서 [1,k]와 [k+1,n]의 임의의 범위로 나눌 수 있고.
// 행렬을 곱하는 개수는 2개부터 n개를 곱하는 경우가 있다.
for (int chain = 2; chain <= n; chain++) {
    /* 2. 몇개의 행렬을 곱하냐에 따라서 다음과 같이 곱할 수 있다.
     EX) 2개의 행렬을 곱한다고 하면 (1,2),(2,3),(3,4)....(n-1,n) 이런 경우의 수들이 나온다.
     따라서 i가 1부터 n-chain+1인 이유가 여기에 있다.
      몇개의 행렬을 곱하냐에 대해 생각해줘야 하기 때문!!*/
    for (int i = 1; i <= n - chain + 1; i++) {
    /* 3. 정해진 i에 따라서 chain길이 만큼 곱해야 하므로 j가 그 역할을 한다.
      chain길이 만큼 끊어주는 것이다.*/

        int j = i + chain - 1;
```

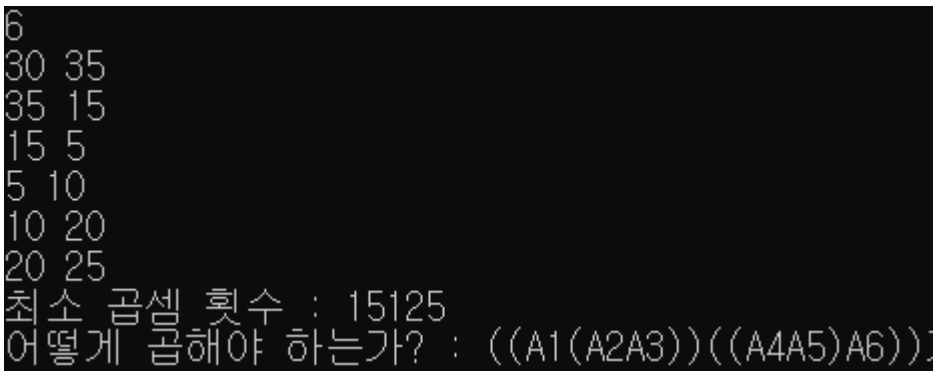
```
            // 처음의 최소 비용은 없기 때문에 최소비용을 찾기 위해서 큰값으로 초기화 해주어야 한다.
            m[i][j] = 2100000000;

        /* i부터 j까지 chain 길이만큼 곱하는데 그 곱하는데도 k를 기준으로 나뉘어질 수 있다.
        EX) 1부터 3까지 곱하는 것은 (1,1),(2,3)과 (1,2),(3,3)으로 나뉜다.
        따라서 k의 범위는 i부터 j-1이 되는 것이다. k+1에 대해서 계산할 때가 있기 때문에 j-1까지..*/
            for (int k = i; k <= j - 1; k++) {
                int temp_min = m[i][k] + m[k + 1][j] + p[i-1] * p[k] * p[j];
                if (temp_min < m[i][j]) {
                    m[i][j] = temp_min;
                    k_position[i][j] = k; // 곱셈이 최소비용을 내는 split하는 위치를 저장!
                }
            }
        }
    }
```

주석에 아주 상세히 설명해서 주석을 차근차근 읽어보면 이해가 된다. 단, 백지 상태에서 이걸 구현해 보라고 하면 아마 못할 것 같다. 어쨌든 위 코드를 컴파일하면 $A_1 \dots A_6$ 의 최소곱셈횟수는 $m[1][6]$이 된다. 하지만 최종 목표가 역시 이것이 아닌 행렬의 곱셈을 어떤 순서로 해야 하는가?가 훨씬 중요한 것이기 때문에 어떻게 나뉘는지 확인하는 코드도 있어야 한다.

```
void PrintOptimalParen(int s[][501], int i, int j)
{
    if (i == j) printf("A%d", i);
    else {
        printf("(");
        PrintOptimalParen(s, i, s[i][j]);
        PrintOptimalParen(s, s[i][j] + 1, j);
        printf(")");
    }
}
```

이 코드도 역시 책에 있는 pseudo code를 보면 바로 구현이 된다. $s[i][j]$에 $A_i \dots A_j$가 나눠지는 기준인 k를 저장했기 때문에 처음 k를 기준으로 계속 하위 k로 쪼개지면서 출력이 된다. 분할정복으로 재귀를 사용한 것이다.



실행하면 예상대로 위 그림에서 주어진 결과가 제대로 나오는 것을 볼 수 있다.
알고리즘 자체의 난이도는 그렇게 높은 편은 아니지만 이해하는데 시간이 꽤 걸렸다. 하나의 알고리즘도 제대로 이해하고 넘어가는 습관이 되게 중요하기 때문에 그런 것 같다. 남에게 설명할 정도의 수준이 되어야 한다!