챕터 6. Dynamic Programming (동적 계획법) - (1) 연쇄 행렬 곱셈

연쇄 행렬 곱셈 문제 (Chained Matrix Multiplication)

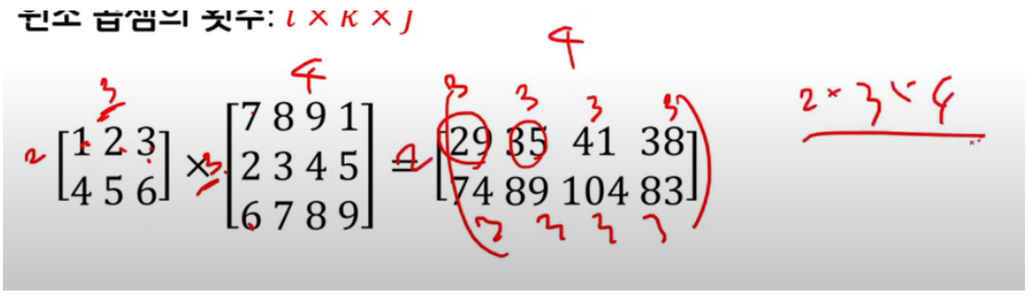
- 주어진 n개의 연쇄 행렬을 곱하는 최적의 순서를 구하시오.
  - n개의 연쇄 행렬 곱셈: A1 * A2 * --- * An
  - 행렬 곱셈은 결합 법칙이 성립: (Ax * Ay) * Az = Ax * (Ay * Az)
  - 하지만, 행렬 곱셈의 순서에 따라서 각 원소의 곱셈 횟수가 달라짐
  - 각 원소의 곱셈 횟수가 가장 작아지도록 하는 곱셈 순서가 최적의 순서

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

연쇄 행렬 곱셈 문제의 이해

- 2 * 3 행렬과 3 * 4 행렬을 곱하면 2 * 4 행렬이 나옴
  - Algorithm 1.4: 원소를 곱하는 횟수는 2 * 3 * 4 = 24

- 일반적으로, i * k 행렬과 k * j 행렬을 곱하면 i * j 행렬이 나옴
  - 원소 곱셈의 횟수: i * k * j



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) =

$$C(n) = \frac{1}{n+1}\binom{2n}{n} \sim \frac{4^n}{n^{3/2}\sqrt{\pi}}$$

- 연쇄 행렬 곱셈이 가지는 경우의 수 = C (n - 1)
    - n개의 항에 괄호를 씌우는 모든 경우의 수 (n = 1, 2, 3, --)

카탈랑 수가 등장하는 조합 문제:

- **괄호 문제: 괄호를 올바르게 짝을 맞추는 경우의 수 문제**
- **BST 문제: 이진 검색 트리의 경우의 수 문제**
- **스택 순열 문제: 스택을 통과하여 만들 수 있는 순열의 경우의 수 문제**
- **연쇄 행렬 곱셈 문제: 행렬의 곱셈 순서를 만들 수 있는 경우의 수 문제**
- **삼각형 분할 문제: $n+2$개의 정점으로 이루어진 다각형을 삼각형으로 분할하는 방법의 수**

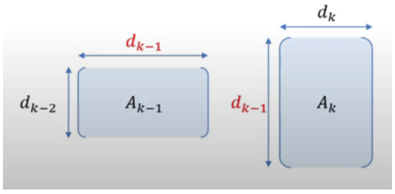
$$\underset{(20\times 2)}{A} \times \underset{(2\times 30)}{B} \times \underset{(30\times 12)}{C} \times \underset{(12\times 8)}{D}$$

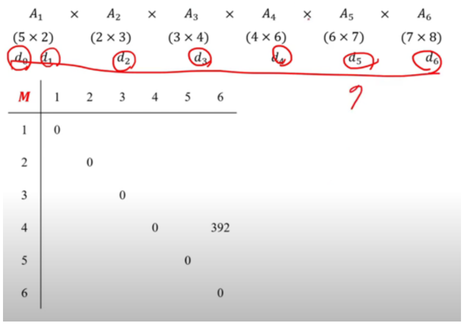
- 연쇄 행렬이 4개일 경우 다섯가지 경우의 수가 존재
    - A(B(CD)) = 3,680
    - (AB)(CD) = 8,880
    - A((BC)D) = 1,232
    - ((AB)C)D = 10,320
    - (A(BC))D = 3,120

--> 4개의 노드(A,B,C,D)로 이진 탐색 트리 (BST)를 만드는 경우와 동일

연쇄 행렬 곱셈 문제의 엄밀한 정의

- n개의 연쇄 행렬 곱셈: A_1 * A_2 * --- * A_n
- A_{k-1}의 행의 개수와 A_k의 열의 개수가 같아야 함
- d_k를 행렬 A_k의 행의 개수로 정함 (1 <= k <= n)
- dk-1은 행렬 Ak의 열의 개수, A_{k-1}의 행의 개수임.
- d_0는 A_1의 열의 개수





연쇄 행렬 곱셈: 동적계획(Dynamic Programming)

- 1단계: 재귀 관계식을 찾는다.
  - M : 연쇄 행렬을 곱하는데 필요한 곱셈의 최소 횟수 행렬
  - M[i][j]: Ai에서 Aj까지 행렬을 곱하는데 필요한 곱셈의 최소 횟수

    (1 <= i <= j <= n)
  - 목표: A_i --- A_j 행렬을 (A_i --- A_k)(A_{k+1} --- A_j)로 분할하는 재귀 관계식 찾기

- 2단계: 상향식 방법으로 해답을 구한다.
  - 초기화: M[i][j] = 0 (주대각선을 0으로)
  - 최종 목표: M[1][n].
  - 상향식 계산: 대각선 1번, 대각선 2번, ---, 대각선 n - 1번

    -> 상향식 계산 기법은 많이 사용됨(모양) -> 

연쇄 행렬 곱셈의 재귀 관계식 구하기

- 분할 정복(Divide-and-Conquer)
  - n개의 행렬을 두 개의 최적 부분행렬의 곱으로 분할
- 예를 들어, A_1, A_2, A_3, A_4, A_5, A_6은 다음과 같이 분할 가능
- 각 분할(부분행렬)의 곱셈 횟수:
  - 각 부분행렬의 곱셈 횟수 + 두 행렬의 곱셈 횟수
  - $M[1][k] + M[k+1][6] + d_0 d_k d_6$
- 최적 분할
  - $M[1][6] = \underset{i \le k \le j-1}{\text{minimum}}(M[1][k] + M[k+1][6] + d_0 d_k d_6)$

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

연쇄 행렬 곱셈의 재귀 관계식

- For 1 <= i <= j <= n

$$M[i][j] = \underset{i \le k \le j-1}{\text{minimum}}\left(M[i][k] + M[k+1][j] + d_{i-1} d_k d_j\right), \text{ if } i < j.$$
$$M[i][i] = 0.$$

$M[i][i] = 0$ for $1 \le i \le 6$

$$M[1][2] = \underset{1 \le k \le 1}{\text{minimum}}(M[1][k] + M[k+1][2] + d_0 d_k d_2)$$
$$= M[1][1] + M[2][2] + d_0 d_1 d_2 = 0 + 0 + 5 \times 2 \times 3 = 30.$$

$$M[1][3] = \underset{1 \le k \le 2}{\text{minimum}}(M[1][k] + M[k+1][3] + d_0 d_k d_3)$$
$$= \text{minimum}(M[1][1] + M[2][3] + d_0 d_1 d_3, M[1][2] + M[3][3] + d_0 d_2 d_3)$$
$$= \text{minimum}(0 + 24 + 5 \times 2 \times 4,\ 30 + 0 + 5 \times 3 \times 4) = 64.$$

$$M[1][4] = \underset{1 \le k \le 3}{\text{minimum}}(M[1][k] + M[k+1][4] + d_0 d_k d_4)$$
$$= \text{minimum}(M[1][1] + M[2][4] + d_0 d_1 d_4, M[1][2] + M[3][4] + d_0 d_2 d_4, M[1][3] + M[4][4] + d_0 d_3 d_4)$$
$$= \text{minimum}(0 + 72 + 5 \times 2 \times 6,\ 30 + 72 + 5 \times 3 \times 6,\ 64 + 0 + 5 \times 4 \times 6) = 132.$$

$M[1][6] = 348.$

A_1	×	A_2	×	A_3	×	A_4	×	A_5	×	A_6
(5×2)		(2×3)		(3×4)		(4×6)		(6×7)		(7×8)
d_0 d_1		d_2		d_3		d_4		d_5		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**
  - $(A_4A_5)A_6$: $d_3 d_4 d_5 + d_3 d_5 d_6 = 392$
  - $A_4(A_5A_6)$: $d_4 d_5 d_6 + d_3 d_4 d_6 = 528$
- $M[4][6] = \min(392,528) = 392$

연쇄행렬곱셈 알고리즘 연습 칸 (다 채워보기) :

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

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

1. i*i는 main diagonal 주 대각선, 주 차원 이라고도 한다.

* 참고 계산: M[1][3] = 132

= min (M[1][1] + M[2][4] + $d_0d_1d_4$,

M[1][2] + M[3][4] + $d_0d_2d_4$,

M[1][3] + M[4][4] + $d_0d_3d_4$)

2. M[1][6], 다시 말해서 A_1 * --- * A_6까지 곱한 것의 최소값을 M[1][6]에 저장한 것이다.

```cpp
// Chained Matrix Multiplication

#include <iostream>
#include <vector>
#include <string>
#include <limits.h>

using namespace std;

typedef vector<vector<int>> matrix_t;

int minimum(int i, int j, int &mink, vector<int> &d, matrix_t &M)
{
    int minValue = INT_MAX, value;
    for (int k = i; k <= j - 1; k++)
    {
        value = M[i][k] + M[k + 1][j] + d[i - 1] * d[k] * d[j];
        if (minValue > value)
        {
            minValue = value;
            mink = k;
        }
    }
    return minValue;
}

void minmult(int n, vector<int> &d, matrix_t &M, matrix_t &P)
{
    for (int i = 1; i <= n; i++)
    {
        M[i][i] = 0;
    }

    for (int diagonal = 1; diagonal <= n - 1; diagonal++)
    {
        for (int i = 1; i <= n - diagonal; i++)
        {
            int j = i + diagonal;
            int k;
            M[i][j] = minimum(i, j, k, d, M);
            P[i][j] = k;
        }
    }
}

void order(int i, int j, matrix_t &P, string &s)
{
    if (i == j)
    {
        s += "(A" + to_string(i) + ")";
    }
    else
    {
        int k = P[i][j];
        s += "(";
        order(i, k, P, s);
        order(k + 1, j, P, s);
        s += ")";
    }
}
```

```cpp
int main()
{
    // 행렬 크기 입력
    int n;
    cin >> n;

    // 행렬 크기에 맞는 차원 배열 입력
    vector<int> d(n + 1);
    for (int i = 0; i <= n; i++)
    {
        cin >> d[i];
    }

    matrix_t M(n + 1, vector<int>(n + 1, 0)); // 곱셈 결과 테이블
    matrix_t P(n + 1, vector<int>(n + 1, 0)); // 분할 지점 테이블

    // 행렬 곱셈 최소 비용 계산 시작
    minmult(n, d, M, P);

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (j == 1)
            {
                cout << M[i][j];
                if (i != n)
                    cout << " ";
            }
            if (M[i][j] != 0)
            {
                cout << M[i][j];
                if (j != n)
                    cout << " ";
            }
        }
        cout << endl;
    }

    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= n; j++)
        {
            if (j == 1)
            {
                cout << P[i][j];
                if (i != n)
                    cout << " ";
            }
            if (P[i][j] != 0)
            {
                cout << P[i][j];
                if (j != n)
                    cout << " ";
            }
        }
        cout << endl;
    }

    // 최적 곱셈 횟수 출력
    cout << M[1][n] << endl;

    // 최적 곱셈 순서 출력
    string optimal_order;
    order(1, n, P, optimal_order);
    cout << optimal_order << endl;

    return 0;
}
```