3

다이나믹 프로그래밍

3.1 파도반 수열 BOJ #9461

점화식을 유도하면 다음과 같습니다.

$$P_n = \begin{cases} 1 & n=1, n=2, n=3 \\ 2 & n=4, n=5 \\ P_{n-1}+P_{n-5} & \text{otherwise} \end{cases}$$

그대로 구현하면 됩니다. 여러 테스트 케이스가 들어오므로 P_{100}까지 전부 구해 두고 테스트 케이스가 들어올 때마다 결과를 출력합니다.

$P_{100} \approx 8.9 \times 10^{11}$이고 이는 `int` 범위에 들어가지 않음에 주의합니다. `int`로 짰다면 테스트 입력으로 `1 100`을 넣었을 때 -203165375가 나오는 것을 확인할 수 있습니다. 테스트 데이터를 잘 구성해 미리 넣어 보고 실수를 방지합시다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```
#include <iostream>

using namespace std;
using ll = long long;

ll dp[101] = {0, 1, 1, 1, 2, 2};

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
```

```cpp
    ios_base::sync_with_stdio(false);

    for (int i = 6; i <= 100; i++) {
        dp[i] = dp[i - 1] + dp[i - 5];
    }

    int t;
    cin >> t;

    while (t--) {
        int x;
        cin >> x;
        cout << dp[x] << '\n';
    }

    return 0;
}
```

3.2 1, 2, 3 더하기 BOJ #9095

어떤 수 n이 있을 때

$$d_n = (n\text{을 } 1, 2, 3\text{의 합으로 나타내는 방법의 수})$$

로 정의합시다. 그러면 우선 $d_1 = 1,\ d_2 = 2,\ d_3 = 4$입니다.

$n > 3$을 1, 2, 3의 합으로 나타내는 방법은

- $n-1$을 1, 2, 3의 합으로 나타내고 마지막에 $+1$
- $n-2$를 1, 2, 3의 합으로 나타내고 마지막에 $+2$
- $n-3$을 1, 2, 3의 합으로 나타내고 마지막에 $+3$

으로 나뉘집니다. 따라서 $n > 3$일 때 $d_n = d_{n-1} + d_{n-2} + d_{n-3}$입니다. 정리하면

$$d_n = \begin{cases} 1 & n = 1 \\ 2 & n = 2 \\ 4 & n = 3 \\ d_{n-1} + d_{n-2} + d_{n-3} & \text{otherwise} \end{cases}$$

이고, 이를 코드로 구현하면 됩니다.

정답 코드

실행 시간 0ms, 메모리 1,988KB

```cpp
#include <iostream>

using namespace std;

int dp[12] = {0, 1, 2, 4};

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    for (int i = 4; i <= 12; i++) {
        dp[i] = dp[i - 1] + dp[i - 2] + dp[i - 3];
    }

    int t;
    cin >> t;

    while (t--) {
        int x;
        cin >> x;
        cout << dp[x] << '\n';
    }

    return 0;
}
```

3.3 동전 1 BOJ #2293

앞의 문제들과 접근은 비슷하나, 순서가 다른 것은 같은 경우이므로 $1+2$원과 $2+1$원을 따로 처리해 줘야 합니다. 어떤 방법을 쓰면 좋을까요?

DP를 갱신해 주는 순서를 바꾸면 됩니다. 일단 첫번째 동전만 사용해 경우의 수를 구하고, 두 번째 동전을 추가해 경우의 수를 구하고, 세 번째 동전을 추가해 경우의 수를 구하고, … 이와 같이 반복하면 순서가 다른 경우는 생기지 않습니다.

정답 코드

실행 시간 0ms, 메모리 2,068KB

```cpp
#include <iostream>

using namespace std;
using ll = long long;

ll dp[101] = {0, 1, 1, 1, 2, 2};

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

```

```
    for (int i = 6; i <= 100; i++) {
        dp[i] = dp[i - 1] + dp[i - 5];
    }

    int t;
    cin >> t;

    while (t--) {
        int x;
        cin >> x;
        cout << dp[x] << '\n';
    }

    return 0;
}
```

동전 문제에도 많은 버전이 있습니다.

- BOJ #11047 동전 0: 동전을 적절히 사용해서 그 가치의 합을 K로 만드려고 할 때, 필요한 동전 개수의 최솟값을 구하는 문제입니다. 대신 동전의 가치가 배수 관계를 이룹니다. 다이나믹 프로그래밍은 아니지만 조금만 생각하면 쉽게 풀리는 문제입니다.
- BOJ #2294 동전 2: 동전 0과 같은 문제이나 동전의 가치가 배수 관계를 이루지 않습니다. 다이나믹 프로그래밍으로 풀 수 있습니다.

3.4 제곱수의 합 BOJ #1699

자연수 N을 제곱수들의 합으로 표현할 때 그 항의 최소 개수를 구해야 합니다. '가장 긴 증가하는 부분 수열' 문제를 접근하듯이 아래와 같은 점화식을 세워볼 수 있습니다. d_i를

$$d_i = (i\text{를 제곱수들의 합으로 표현할 때, 그 항의 최소 개수})$$

로 정의한다면, $j+k^2=i$가 되는 모든 j들에 대해 j를 제곱수들의 합으로 표현한 것 뒤에 k^2를 더해 주면 i가 되므로 d_i는 결국 i보다 작으면서 $j+k^2=i$가 되는 모든 j에 대해 최솟값을 취한

$$d_i = \begin{cases} 1 & i = 0 \\ \displaystyle\min_{0 \le j < i,\, k \in \mathbb{N},\, j+k^2=i} (d_j + 1) & \text{otherwise} \end{cases}$$

가 됩니다.

$N \le 10^5$이고, N 이하의 제곱수는 $\left\lfloor \sqrt{N} \right\rfloor$개 있으므로 시간 복잡도는 $\mathcal{O}\left(N^{\frac{3}{2}}\right)$이고 이는 2초 안에 충분히 실행 가능합니다.

정답 코드

실행 시간 32ms, 메모리 2,380KB

```cpp
#include <iostream>
#include <algorithm> // min()

using namespace std;

int dp[100001];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    fill(dp, dp + 100001, 987654321);

    int n;
    cin >> n;

    dp[0] = 0;
    for (int i = 0; i <= n; i++) {
        for (int k = 1; i - k * k >= 0; k++) {
            dp[i] = min(dp[i], dp[i - k * k] + 1);
        }
    }

    cout << dp[n];

    return 0;
}
```

3.5 이항 계수 2 BOJ #11051

이항 계수에는 다음과 같은 성질이 있습니다.

$$\binom{n}{k} + \binom{n}{k+1} = \binom{n+1}{k+1}$$

이를 2차원 배열을 사용해 그대로 코드로 옮기면 됩니다.

중간 과정에서 계산 결과가 `int` 범위를 초과할 수 있는데, $\binom{n}{k}$를 전부 구한 후 마지막에 나머지 연산을 하지 말고 아래의 성질을 이용해 더할 때마다 매번 나머지 연산을 하면 오버플로우를 방지할 수 있습니다.

$$(a+b) \bmod m = (a \bmod m + b \bmod m) \bmod m$$

정답 코드

실행 시간 4ms, 메모리 5,900KB

```cpp
#include <iostream>
#include <algorithm> // min()

using namespace std;

int binom[1001][1001];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n, k;
    cin >> n >> k;

    binom[0][0] = 1;
    for (int i = 0; i < n; i++) {
        binom[i + 1][0] = 1;
        for (int j = 0; j < k; j++) {
            binom[i + 1][j + 1] = (binom[i][j] + binom[i][j + 1]) % 10007;
        }
    }

    cout << binom[n][k];

    return 0;
}
```

3.6 스티커 BOJ #9465

i행에서 윗쪽 스티커의 점수를 s_{i0}, 아랫쪽 스티커의 점수를 s_{i1} 이라고 두고, d_{ij}를 다음과 같이 정의합시다.

$$d_{ij} = \begin{cases} (i\text{번째 행에서 아무 스티커도 안 골랐을 때의 점수의 최댓값}) & j = 0 \\ (i\text{번째 행에서 윗쪽 스티커를 골랐을 때의 점수의 최댓값}) & j = 1 \\ (i\text{번째 행에서 아랫쪽 스티커를 골랐을 때의 점수의 최댓값}) & j = 2 \end{cases}$$

그러면 각각의 경우 점화식은 다음과 같습니다.

$$d_{ij} = \begin{cases} \max\left\{d_{(i-1)0}, d_{(i-1)1}, d_{(i-1)2}\right\} & j = 0 \\ \max\left\{d_{(i-1)0}, d_{(i-1)2}\right\} + s_{i0} & j = 1 \\ \max\left\{d_{(i-1)0}, d_{(i-1)1}\right\} + s_{i1} & j = 2 \end{cases}$$

- d_{i0}의 경우 이번 열에서 아무것도 고르지 않으므로, 바로 전 열에서의 점수의 최댓값을 가져옵니다.
- d_{i1}의 경우 이번 열에서 윗쪽 스티커를 고르므로, 바로 전 열에서 윗쪽 스티커를 고른 경우만 제외하고 최댓값을 가져옵니다. 그리고 윗쪽 스티커의 점수를 더합니다.

- d_{i2}의 경우 이번 열에서 윗쪽 스티커를 고르므로, 바로 전 열에서 아랫쪽 스티커를 고른 경우만 제외하고 최댓값을 가져옵니다. 그리고 아랫쪽 스티커의 점수를 더합니다.

이를 코드로 구현하면 정답입니다.

정답 코드

실행 시간 92ms, 메모리 3,940KB

```cpp
#include <iostream>
#include <cstring> // memset()
#include <algorithm> // min()

using namespace std;

int s[100000][2], dp[100000][3];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;

        for (int j = 0; j < 2; j++) {
            for (int i = 0; i < n; i++) {
                cin >> s[i][j];
            }
        }

        dp[0][0] = 0;
        dp[0][1] = s[0][0];
        dp[0][2] = s[0][1];

        for (int i = 1; i < n; i++) {
            dp[i][0] = max({dp[i - 1][0], dp[i - 1][1], dp[i - 1][2]});
            dp[i][1] = max(dp[i - 1][0], dp[i - 1][2]) + s[i][0];
            dp[i][2] = max(dp[i - 1][0], dp[i - 1][1]) + s[i][1];
        }

        cout << max({dp[n - 1][0], dp[n - 1][1], dp[n - 1][2]}) << '\n';
    }

    return 0;
}
```

3.7 상자넣기 BOJ #1965

문제를 읽어 보면 결국 가장 긴 증가하는 부분 수열을 구하는 것과 동일한 문제입니다. 강의자료에 나온 방법 그대로 코딩하면 정답입니다.

정답 코드

실행 시간 0ms, 메모리 1,996KB

```
#include <iostream>
#include <algorithm>
using namespace std;

int a[1000], d[1000];
int main() {
    int n;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
        d[i] = 1;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] >= a[i]) continue;
            d[i] = max(d[i], d[j] + 1);
        }
    }

    int mx = 0;
    for (int i = 0; i < n; i++) {
        mx = max(mx, d[i]);
    }
    cout << mx;

    return 0;
}
```

3.8 RGB거리 BOJ #1149

'스티커' 문제와 비슷하게 접근하면 됩니다. 집 i를 칠하는 비용을 s_{ij}라고 두고, d_{ij}를 다음과 같이 정의합니다.

$$d_{ij} = \begin{cases} (i\text{번째 집을 빨강으로 칠했을 때의 비용의 최솟값}) & j = 0 \\ (i\text{번째 집을 초록으로 칠했을 때의 비용의 최솟값}) & j = 1 \\ (i\text{번째 집을 파랑으로 칠했을 때의 비용의 최솟값}) & j = 2 \end{cases}$$

그러면 각각의 경우 점화식은 역시 '스티커' 문제와 비슷하게 다음과 같이 세울 수 있습니다.

$$d_{ij} = \begin{cases} \min\left\{d_{(i-1)1}, d_{(i-1)2}\right\} + s_{i0} & j = 0 \\ \min\left\{d_{(i-1)0}, d_{(i-1)2}\right\} + s_{i1} & j = 1 \\ \min\left\{d_{(i-1)0}, d_{(i-1)1}\right\} + s_{i2} & j = 2 \end{cases}$$

- d_{i0}의 경우 바로 전의 집을 빨강으로 칠한 경우를 제외하고 최솟값을 가져옵니다. 그리고 i번째 집을 빨강으로 칠합니다.
- d_{i1}의 경우 바로 전의 집을 초록으로 칠한 경우를 제외하고 최솟값을 가져옵니다. 그리고 i번째 집을 초록으로 칠합니다.
- d_{i2}의 경우 바로 전의 집을 파랑으로 칠한 경우를 제외하고 최솟값을 가져옵니다. 그리고 i번째 집을 파랑으로 칠합니다.

이를 구현하면 됩니다. 입력 방식이 '스티커' 문제와 다름에 주의하세요.

정답 코드

실행 시간 0ms, 메모리 2,012KB

```cpp
#include <iostream>
#include <cstring> // memset()
#include <algorithm> // min()

using namespace std;

int s[1000][3], dp[1000][3];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n;
    cin >> n;

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < 3; j++) {
            cin >> s[i][j];
        }
    }

    dp[0][0] = s[0][0];
    dp[0][1] = s[0][1];
    dp[0][2] = s[0][2];

    for (int i = 1; i < n; i++) {
        dp[i][0] = min(dp[i - 1][1], dp[i - 1][2]) + s[i][0];
        dp[i][1] = min(dp[i - 1][0], dp[i - 1][2]) + s[i][1];
        dp[i][2] = min(dp[i - 1][0], dp[i - 1][1]) + s[i][2];
    }

    cout << min({dp[n - 1][0], dp[n - 1][1], dp[n - 1][2]}) << '\n';
```

```
    return 0;
}
```

3.9 기타리스트 BOJ #1495

d_{ij}를 다음과 같이 정의합시다.

$$d_{ij} = (i\text{번째 곡을 볼륨 } j\text{로 연주할 수 있는가의 여부})$$

맞습니다. DP 배열을 `int`가 아닌 `bool`로 선언할 것입니다.

DP 배열을 이런 식으로 생각할 수 있다는 아이디어가 있다면 점화식은 비교적 간단히 생각할 수 있습니다. 바로 전 곡을 볼륨 j로 연주할 수 있었을 경우 이번 곡은 볼륨 $j-V_i$, $j+V_i$로 연주할 수 있습니다.

$$d_{ij} = \begin{cases} \text{true} & d_{(i-1)(j-V_i)} \text{ is true} \\ \text{true} & d_{(i-1)(j+V_i)} \text{ is true} \\ \text{false} & \text{otherwise} \end{cases}$$

이를 각 곡의 각 볼륨마다 반복하면 됩니다.

정답 코드

실행 시간 0ms, 메모리 2,088KB

```cpp
#include <iostream>

using namespace std;

bool dp[101][1002];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n, s, m;
    cin >> n >> s >> m;

    dp[0][s] = true;

    for (int i = 1; i <= n; i++) {
        int x;
        cin >> x;

        for (int j = 0; j <= m + 1; j++) {
            if (!dp[i - 1][j]) continue;
            if (j + x <= m) dp[i][j + x] = true;
            if (0 <= j - x) dp[i][j - x] = true;
```

```cpp
        }
    }

    int mx = -1;
    for (int i = 0; i <= m + 1; i++) {
        if (dp[n][i]) mx = i;
    }

    cout << mx;

    return 0;
}
```

3.10 2차원 배열의 합 BOJ #1495

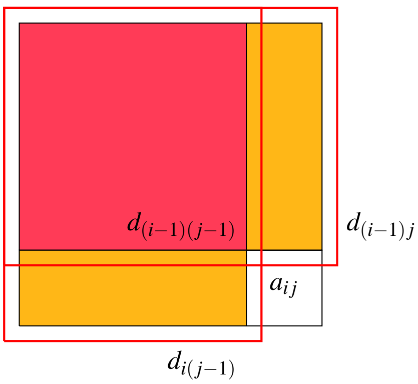
원래 배열을 a라고 두고, d_{ij}를 다음과 같이 정의합시다.

$$d_{ij} = ((0,0) \text{ 위치부터 } (i,j) \text{ 위치까지에 저장된 수의 합})$$

그러면 점화식은 다음과 같습니다.

$$d_{ij} = d_{(i-1)j} + d_{i(j-1)} - d_{(i-1)(j-1)} + a_{ij}$$

아래의 그림을 보면 조금 더 이해하기 쉬울 것입니다.



이제 위와 같은 방식으로 들어오는 부분 $(i,j)\cdots(x,y)$마다 다음을 계산해 주면 됩니다.

$$d_{xy} - d_{x(j-1)} - d_{(i-1)y} + d_{(i-1)(j-1)}$$

정답 코드

실행 시간 12ms, 메모리 2,696KB

```cpp
#include <iostream>

using namespace std;

int a[301][301], dp[301][301];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> a[i][j];
            dp[i][j] = dp[i - 1][j] + dp[i][j - 1] - dp[i - 1][j - 1] + a[i][j];
        }
    }

    int k;
    cin >> k;
    while (k--) {
        int i, j, x, y;
        cin >> i >> j >> x >> y;
        cout << dp[x][y] - dp[x][j - 1] - dp[i - 1][y] + dp[i - 1][j - 1] << '\n';
    }

    return 0;
}
```

계산의 편의를 위해 1-based 인덱스를 사용했습니다.

3.11 1학년 BOJ #5557

‘기타리스트’ 문제와 같은 방식으로 풀면 됩니다. 단 이번엔 `dp` 배열에 경우의 수를 누적해줘야 합니다. 값이 `long long` 범위임에 주의하세요.

정답 코드

실행 시간 0ms, 메모리 2,004KB

```cpp
#include <iostream>

using namespace std;
using ll = long long;

ll dp[21][100];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);
```

```cpp

    int n, s;
    cin >> n;

    for (int i = 0; i < n - 1; i++) {
        int x;
        cin >> x;
        if (i == 0) {
            dp[x][i] = 1;
            continue;
        }
        for (int j = 0; j <= 20; j++) {
            int u = j + x, v = j - x;
            if (0 <= u && u <= 20) dp[u][i] += dp[j][i - 1];
            if (0 <= v && v <= 20) dp[v][i] += dp[j][i - 1];
        }
    }

    cin >> s;
    cout << dp[s][n - 2];

    return 0;
}
```

3.12 신나는 함수 실행 BOJ #9184

3차원 배열을 만들고, 위에서 아래로 내려가는 식으로 문제에서 요구하는 그대로 DP를 구현하면 됩니다. 줄 17이 핵심입니다.

정답 코드

실행 시간 0ms, 메모리 2,060KB

```cpp
1  #include <iostream>
2  
3  using namespace std;
4  using ll = long long;
5  
6  ll dp[21][21][21];
7  
8  ll w(int a, int b, int c) {
9      if (a <= 0 || b <= 0 || c <= 0) {
10         return 1;
11     }
12 
13     if (a > 20 || b > 20 || c > 20) {
14         return w(20, 20, 20);
15     }
16 
17     if (dp[a][b][c]) return dp[a][b][c];
18 
19     if (a < b && b < c) {
20         return dp[a][b][c] = w(a, b, c - 1)
21                            + w(a, b - 1, c - 1)
22                            - w(a, b - 1, c);
```

```cpp
    }

    return dp[a][b][c] = w(a - 1, b, c)
                       + w(a - 1, b - 1, c)
                       + w(a - 1, b, c - 1)
                       - w(a - 1, b - 1, c - 1);
}

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int a, b, c;
    while (cin >> a >> b >> c) {
        if (a == -1 && b == -1 && c == -1) break;
        cout << "w(" << a << ", " << b << ", " << c << ") = ";
        cout << w(a, b, c) << '\n';
    }

    return 0;
}
```

3.13 팰린드롬? BOJ #10942

배열 a에 대해 $d_{i,j}$를 다음과 같이 정의합시다.

$d_{i,j} =$ (i번째 문자부터 j번째 문자까지로 구성된 부분 문자열이 팰린드롬)

그러면 점화식을 아래와 같이 세울 수 있습니다.

$$d_{i,j} = \begin{cases} \text{true} & i = j \\ a_i = a_j & i+1 = j \\ d_{i+1,j-1} \text{ is true and } a_i = a_j & \text{otherwise} \end{cases}$$

- 길이가 1인 부분 배열은 무조건 팰린드롬입니다.
- 길이가 2인 부분 배열은 두 수가 같으면 팰린드롬입니다.
- 길이가 3 이상인 부분 배열이 팰린드롬이려면, 맨 앞 수와 맨 뒤 수가 같고, 그 사이의 부분 배열이 팰린드롬이면 됩니다.

이처럼 DP 배열의 인덱스에 구간의 시작과 끝을 넣는 식으로 활용할 수도 있습니다.

다만 이 문제의 경우 DP 배열을 업데이트하는 순서도 신경써줘야 합니다. 길이가 i인 부분 배열에 대한 DP가 전부 처리되어야 길이가 $i+2$인 부분 배열에 대해서도 처리할 수 있기 때문에, 무식하게 i 순서대로 j 순서대로 돌리면 틀립니다.

대신 문자열의 길이 d를 순서대로 돌리면서 내부에서 문자열의 시작 인덱스 i를 순서대로 돌리면 됩니다. 아래의 코드 줄 20–22에서 확인할 수 있습니다.

정답 코드

실행 시간 224ms, 메모리 5,900KB

```
1  #include <iostream>
2  
3  using namespace std;
4  
5  int a[2000];
6  bool dp[2000][2000];
7  
8  int main() {
9      cin.tie(nullptr);
10     cout.tie(nullptr);
11     ios_base::sync_with_stdio(false);
12 
13     int n;
14     cin >> n;
15 
16     for (int i = 0; i < n; i++) {
17         cin >> a[i];
18     }
19 
20     for (int d = 0; d < n; d++) {
21         for (int i = 0; i < n; i++) {
22             int j = i + d;
23             if (j >= n) break;
24 
25             if (i == j) {
26                 dp[i][j] = true;
27             } else if (i + 1 == j) {
28                 if (a[i] == a[j]) {
29                     dp[i][j] = true;
30                 }
31             } else {
32                 if (a[i] == a[j] && dp[i + 1][j - 1]) {
33                     dp[i][j] = true;
34                 }
35             }
36         }
37     }
38 
39     int m;
40     cin >> m;
41 
42     while (m--) {
43         int u, v;
44         cin >> u >> v;
45         u--, v--;
46         cout << dp[u][v] << '\n';
47     }
48 
49     return 0;
50 }
```

3.14 Dance Dance Revolution BOJ #2342

두 발의 위치를 인덱스로 활용하기로 합시다. $d_{i,j,k}$를 다음과 같이 정의합시다.

$d_{i,j,k}$
$=$ (i개의 지시 사항을 완료한 후 왼쪽 발이 j, 오른쪽 발이 k에 있을 때 든 최소의 힘)

발을 i에서 j로 움직이는 비용을 $\mathrm{mov}_{i,j}$라고 하고, i번째 지시 사항을 x_i라 할 때 점화식은 다음과 같습니다.

$$d_{i,x,k} = \min_{0 \le j \le 4, 0 \le k \le 4, x \ne k} \left(d_{i,j,k} + \mathrm{mov}_{j,x} \right)$$
$$d_{i,j,x} = \min_{0 \le j \le 4, 0 \le k \le 4, x \ne j} \left(d_{i,j,k} + \mathrm{mov}_{k,x} \right)$$

- 왼발이 j에 있었을 때 이를 x로 옮김
- 오른발이 k에 있었을 때 이를 x로 옮김

두 가지 상황을 고려해 구현하면 됩니다.

정답 코드

실행 시간 16ms, 메모리 12,144KB

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int a[100000], dp[100001][5][5];
int inf = 98765432;
int mov[5][5] = {
        {1, 2, 2, 2, 2},
        {0, 1, 3, 4, 3},
        {0, 3, 1, 3, 4},
        {0, 4, 3, 1, 3},
        {0, 3, 4, 3, 1}
};

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    for (int i = 0; i <= 100000; i++) {
        for (int j = 0; j < 5; j++) {
            for (int k = 0; k < 5; k++) {
                dp[i][j][k] = inf;
            }
        }
    }

    dp[0][0][0] = 0;

```

```cpp
    int n = 0;
    while (true) {
        cin >> a[n];
        if (!a[n]) break;
        n++;
    }

    for (int i = 0; i < n; i++) {
        int x = a[i];
        for (int j = 0; j < 5; j++) {
            for (int k = 0; k < 5; k++) {
                if (x != k) dp[i + 1][x][k] = min(
                        dp[i + 1][x][k], dp[i][j][k] + mov[j][x]
                        );
                if (j != x) dp[i + 1][j][x] = min(
                        dp[i + 1][j][x], dp[i][j][k] + mov[k][x]
                        );
            }
        }
    }

    int mn = inf;
    for (int j = 0; j < 5; j++) {
        for (int k = 0; k < 5; k++) {
            mn = min(mn, dp[n][j][k]);
        }
    }
    cout << mn;

    return 0;
}
```

3.15 행렬 곱셈 순서 BOJ #11049

'팰린드롬?' 문제처럼 시작과 끝을 인덱스로 사용하기로 합시다. $d_{i,j}$를 다음과 같이 정의합시다.

$$d_{i,j} = (i\text{번째 행렬부터 } j\text{번째 행렬까지를 곱했을 때의 연산의 수의 최솟값})$$

그러면 점화식을 다음과 같이 세울 수 있습니다.

$$d_{i,j} = \min_{i \le k, k+1 \le j} \left(d_{i,k} + d_{k+1,j} + (\text{행렬 곱셈에 필요한 연산 수}) \right)$$

i, j 사이의 수 k에 대해 $i \cdots k$번째 행렬을 곱한 것과 $(k+1) \cdots j$번째 행렬을 곱한 것을 곱했을 때의 연산의 수를 최소화하면 됩니다. 행렬 곱셈에 필요한 연산 수는 줄 36에서 구했습니다.

정답 코드

실행 시간 56ms, 메모리 3,952KB

```cpp
#include <iostream>
#include <algorithm>

using namespace std;
using ll = long long;
using pii = pair<int, int>;

ll inf = 1ll << 60;
pii a[501];
ll dp[501][501];

int main() {
    cin.tie(nullptr);
    cout.tie(nullptr);
    ios_base::sync_with_stdio(false);

    int n;
    cin >> n;
    for (int i = 1; i <= n; i++) {
        cin >> a[i].first >> a[i].second;
        fill(dp[i], dp[i] + n + 1, inf);
    }

    for (int d = 0; d < n; d++) {
        for (int i = 1; i <= n; i++) {
            int j = i + d;
            if (j > n) break;

            if (d == 0) {
                dp[i][j] = 0;
            } else {
                for (int k = i; k < j; k++) {
                    dp[i][j] = min(
                            dp[i][j],
                            dp[i][k] + dp[k + 1][j]
                            + a[i].first * a[k].second * a[j].second
                            );
                }
            }
        }
    }

    cout << dp[1][n] << '\n';

    return 0;
}
```