문제. 회전 방어

한 명의 주인공 캐릭터가 존재한다. 주인공 캐릭터는 처음에 위쪽(상)을 바라보고 있다. 이후에 1초부터 N초에 걸쳐서 매 초마다 상, 하, 좌, 우 위치에 적이 나타난다. 주인공 캐릭터는 매 초마다 방향을 바꾸지 않거나(가만히 있기), 좌로 90도 회전(좌회전), 우로 90도 회전(우회전)할 수 있다. 주인공 캐릭터의 위치는 고정되어 위치를 옮기는 것은 불가능하다.

매 초마다 ① 먼저 캐릭터가 회전 방향을 결정한 뒤에 ② 상, 하, 좌, 우 위치에서 적군이 한 칸씩 다가와 공격을 수행한다.

적군으로부터 공격을 받은 주인공 캐릭터의 점수는 적군의 공격력에 따라 차감된다. 구체적으로 캐릭터의 앞(정면)에 존재하는 적은 공격력의 1배수만큼, 옆에 존재하는 적은 공격력의 2배수만큼, 뒤(후면)에 존재하는 적은 공격력의 3배수만큼 점수가 차감된다.

앞서 언급했듯이 총 N초에 걸쳐서 매 초마다 상, 하, 좌, 우 위치에 각각 적군이 1명씩 등장한다. 따라서 입력은 $4 \times N$ 크기의 2차원 배열로 주어지며, 이 배열은 총 N초에 걸쳐서 등장하는 각 적군들의 공격력 정보를 포함한다. 이때 각 행의 의미를 주인공의 위치를 중심으로 살펴 보면, 첫 번째 행은 위쪽(상), 두 번째 행은 오른쪽(우), 세 번째 행은 아래쪽(하), 네 번째 행은 왼쪽(좌) 위치를 의미한다.

캐릭터가 N초간 적절하게 행동하도록 하여 최소한의 데미지를 받도록 하는(최소한의 점수가 차감되도록 하는) 프로그램을 작성하여라. 예를 들어 $N = 4$이고, 다음과 같이 적군의 공격력이 주어진다고 가정하자.

시간	1초	2초	3초	4초
상	2	2	3	7
우	1	3	8	6
하	3	5	6	5
좌	4	4	9	3

만약 주인공 캐릭터가 다음과 같이 (1) 가만히, (2) 좌회전, (3) 우회전, (4) 우회전 순서대로 행동한다면, 총 (21 + 27 + 55 + 39) = 142점이 감점된다.

시간	1초	2초	3초	4초
행동	가만히	좌회전	우회전	우회전
현재 보는 곳	상	좌	상	우
차감되는 점수	21	27	55	39

하지만 주인공 캐릭터가 다음과 같이 (1) 좌회전, (2) 좌회전, (3) 가만히, (4) 좌회전 순서대로 행동한다면, 총 (17 + 25 + 49 + 39) = 130점이 감점된다. 이 점수는 주인공이 최선의 행동을 했을 때 차감되는 점수 합으로, 최솟값이다.

시간	1초	2초	3초	4초
행동	좌회전	좌회전	가만히	좌회전
현재 보는 곳	좌	하	하	우
차감되는 점수	17	25	49	39

주인공이 N초 동안 최선의 행동을 함으로써 만들 수 있는, 차감되는 점수의 합의 최솟값을 반환하는 프로그램을 작성하여라.

입력 조건

가장 먼저 적군이 등장하는 횟수(시간) N이 자연수로 주어진다. N은 2 이상 9 이하의 자연수다.
이어서 $4 \times N$ 크기의 2차원 배열 $enemies$가 주어지며, 이 배열은 총 N초에 걸쳐서 등장하는 각 적군들의 공격력 정보를 포함한다. 모든 적군의 공격력의 값은 1 이상 100 이하의 자연수다.

출력 조건

주인공이 N초 동안 최선의 행동을 함으로써 만들 수 있는 차감되는 점수의 합의 최솟값을 자연수 형태로 반환한다.

입출력 예시

N	$enemies$	정답
4	[[2, 2, 3, 7], [1, 3, 8, 6], [3, 5, 6, 5], [4, 4, 9, 3]]	130
4	[[1, 2, 3, 4], [6, 2, 5, 3], [4, 7, 1, 2], [8, 6, 3, 2]]	107

해설 3. 회전 방어

본 문제는 백트래킹(backtracking) 알고리즘을 이용하여 모든 경우의 수를 탐색하는 방식으로 해결할 수 있다. 1초부터 N초에 걸쳐서 주인공 캐릭터는 매 초마다 방향을 바꾸지 않거나(가만히 있기), 좌로 90도 회전(좌회전), 우로 90도 회전(우회전)할 수 있다. 따라서 매 초마다 3가지 선택지가 존재하며, 동일한 선택지를 반복적으로 고를 수 있다는 점에서 중복 순열을 계산하는 것과 유사하다. 그러므로 전체 3^N가지 모든 경우의 수를 고려하는 프로그램을 깊이 우선 탐색(DFS)을 이용하여 작성하여 해결할 수 있다.

- **Python3 정답 코드 예시**

```
def dfs(N, enemies, depth):
    # 모든 중복 순열을 확인하는 부분
    if depth == N:  # 3개의 선택지 중에서 N초에 걸쳐서 하나씩 선택한 상황
        view = 0  # 처음에 상(위쪽) 방향을 보고 있음
        result = 0
        for i in range(N):  # 1초부터 N초까지 확인
            # 선택지에 따라서 주인공의 방향 변경
            if selected[i] == 0:  # 가만히 있기
```

```
                pass
            elif selected[i] == 1:  # 좌회전
                view = (view - 1) % 4
            elif selected[i] == 2:  # 우회전
                view = (view + 1) % 4
            # 상(0), 우(1), 하(2), 좌(3)의 위치를 확인하며
            for j in range(4):
                power = enemies[j][i]  # 해당 위치의 적군의 공격력
                # 앞에 존재하는 적은 공격력의 1배수
                if view == j:
                    result += power
                # 오른쪽에 존재하는 적은 공격력의 2배수
                elif (view + 1) % 4 == j:
                    result += power * 2
                # 왼쪽에 존재하는 적은 공격력의 2배수
                elif (view - 1) % 4 == j:
                    result += power * 2
                # 뒤쪽에 존재하는 적은 공격력의 3배수
                elif (view + 2) % 4 == j:
                    result += power * 3
        # 차감되는 점수의 합의 최솟값 계산
        global answer
        answer = min(answer, result)
        return
    # 0번, 1번, 2번의 선택지를 하나씩 확인하며
    for i in range(3):
        # 가만히 있기(0), 좌회전(1), 우회전(2)
        selected.append(i)  # 현재 원소 선택
        dfs(N, enemies, depth + 1)  # 재귀 함수 호출
        selected.pop()  # 현재 원소 선택 취소

selected = []  # 현재 중복순열에 포함된 원소의 인덱스(index)
answer = int(1e9)

# 적군이 등장하는 횟수(N)와 적군의 공격력 정보 배열(enemies) 입력받기
def solution(N, enemies):
    dfs(N, enemies, 0)  # DFS를 활용한 모든 경우의 수 탐색
    return answer
```

- **Java 정답 코드 예시**

```
class Solution {
    // 현재 중복순열에 포함된 원소의 인덱스(index)
    public static int[] selected = new int[9];
    public static int answer = (int) 1e9;

    public static void dfs(int N, int[][] enemies, int depth) {
        // 모든 중복 순열을 확인하는 부분
        if (depth == N) {
            // 3개의 선택지 중에서 N초에 걸쳐서 하나씩 선택한 상황
            int view = 0; // 처음에 상(위쪽) 방향을 보고 있음
            int result = 0;
            for (int i = 0; i < N; i++) { // 1초부터 N초까지 확인
                // 선택지에 따라서 주인공의 방향 변경
                if (selected[i] == 0) {
                    // 가만히 있기
                }
                else if (selected[i] == 1) { // 좌회전
                    view = view - 1;
                    if (view == -1) view = 3;
                }
                else if (selected[i] == 2) { // 우회전
                    view = (view + 1) % 4;
                }
                // 상(0), 우(1), 하(2), 좌(3)의 위치를 확인하며
                for (int j = 0; j < 4; j++) {
                    int power = enemies[j][i]; // 해당 위치의 적군의 공격력
                    // 앞에 존재하는 적은 공격력의 1배수
                    if (view == j) {
                        result += power;
                    }
                    // 오른쪽에 존재하는 적은 공격력의 2배수
                    else if ((view + 1) % 4 == j) {
                        result += power * 2;
                    }
                    // 뒤쪽에 존재하는 적은 공격력의 3배수
                    else if ((view + 2) % 4 == j) {
                        result += power * 3;
                    }
                    // 왼쪽에 존재하는 적은 공격력의 2배수
                    else {
                        result += power * 2;
                    }
```

```
                }
            }
            // 차감되는 점수의 합의 최솟값 계산
            answer = Math.min(answer, result);
            return;
        }
        // 0번, 1번, 2번의 선택지를 하나씩 확인하며
        for (int i = 0; i < 3; i++) {
            // 가만히 있기(0), 좌회전(1), 우회전(2)
            selected[depth] = i; // 현재 원소 선택
            dfs(N, enemies, depth + 1); // 재귀 함수 호출
            selected[depth] = -1; // 현재 원소 선택 취소
        }
    }

    // 적군이 등장하는 횟수(N)와 적군의 공격력 정보 배열(enemies) 입력받기
    public static int solution(int N, int[][] enemies) {
        dfs(N, enemies, 0); // DFS를 활용한 모든 경우의 수 탐색
        return answer;
    }
}
```