고급소프트웨어실습1 9주차 과제

20191574 김예진

**1. 실습 3이 Greedy algorithm과 Dynamic programming 중 어디에 해당된다고 할 수 있는가? 이유를 설명하시오.**

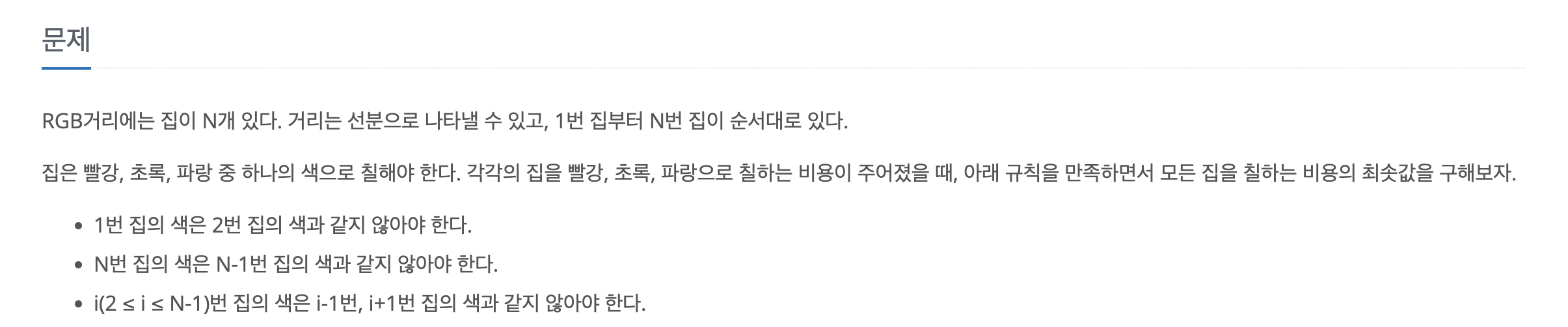
Greedy algorithm은 현재 step에서 최적의 해를 찾아가는 방식으로 문제를 푸는 방법이다. 예를 들어 Greedy algorithm의 대표적 문제인 회의 배정 문제의 경우, 가능한 회의들 중 끝나는 시간이 가장 빠른 순으로 선택해 나가는 rule을 가지고 문제를 해결할 수 있다.

이 문제는 어떤 rule을 가지고 매 step에서 원소 채택 여부를 정하기 어렵다. 여기서 rule이란 서브태스크의 영향을 받지 않은 독립적인 rule을 의미한다. 실습 3의 경우에는 매 원소에 대해 채택 여부를 결정할 때 직전 원소, 즉 서브태스크의 값이 무엇인지를 확인해야만 현재 step에서 결정을 내릴 수 있다.

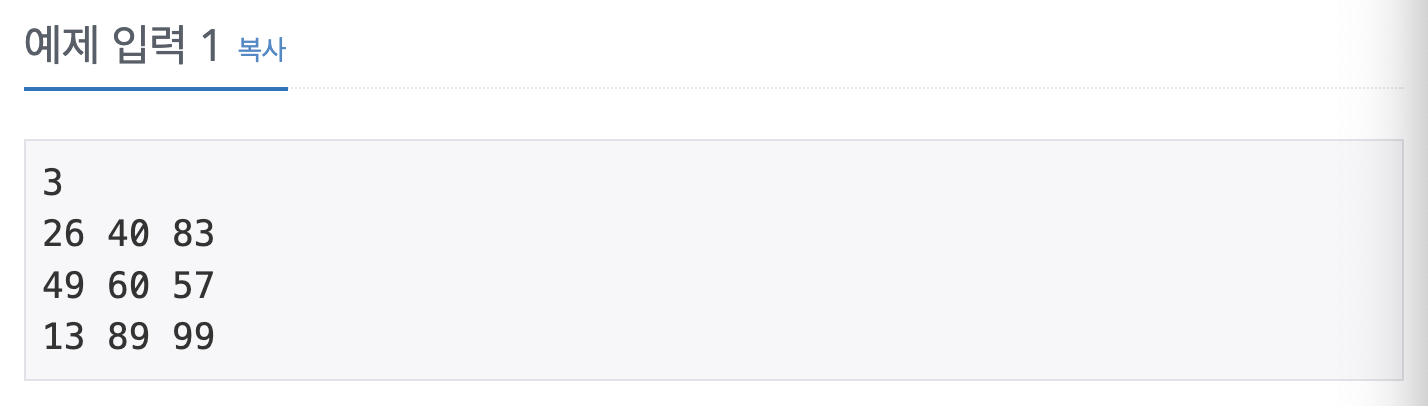
따라서 실습 3은 Dynamic Programming에 해당한다.

**2. 앞에서 다루지 않은 문제 중에 Dynamic programming을 사용했을 때 더 효율적으로 풀 수 있는 문제를 2개 들고, 알고리즘을 설명하시오. brute force 방식에 비햐여 DP를 사용하는 경우 어느 정도 빨라지는 지도 설명하시오.**

2-1)

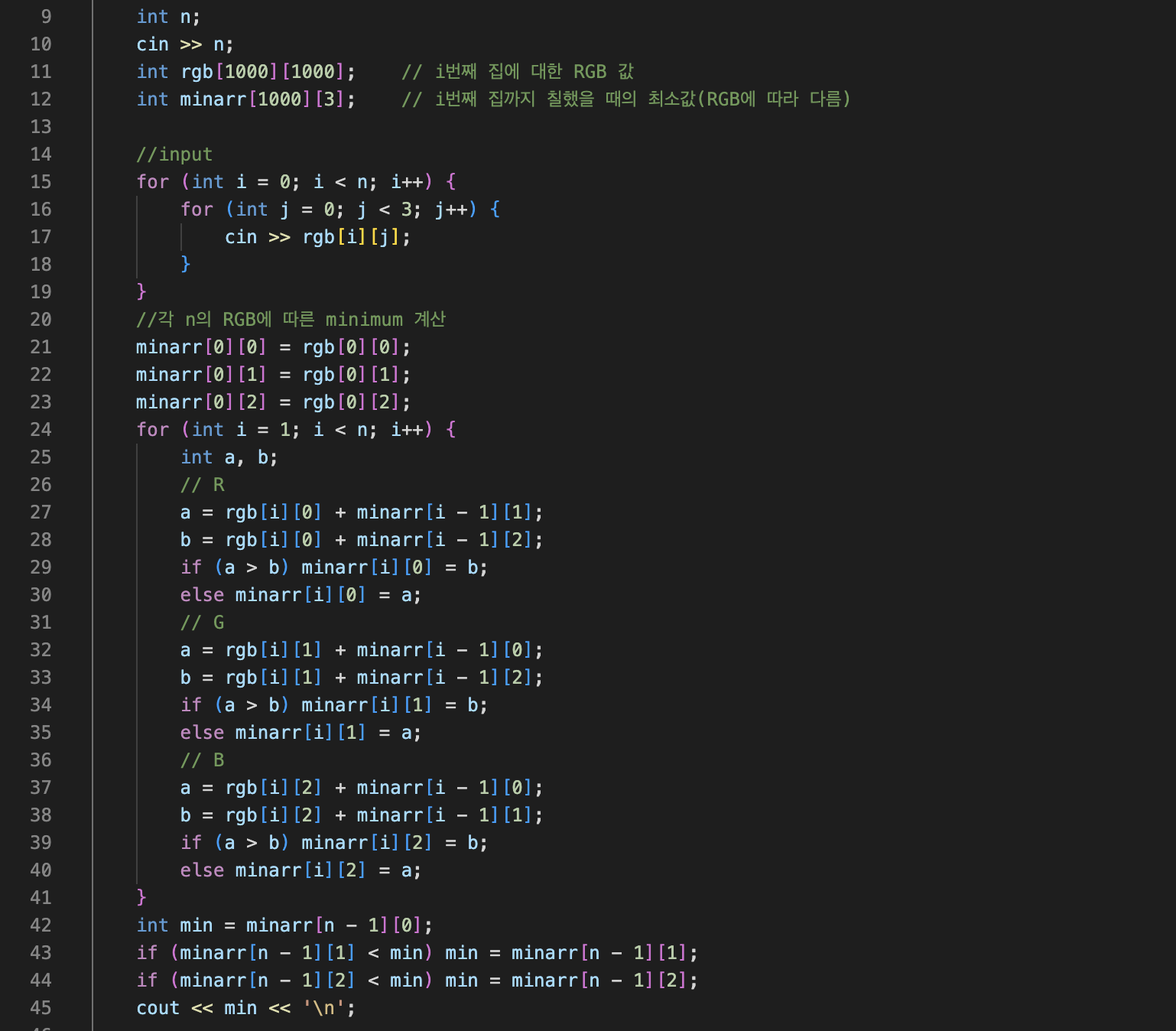


이 문제는 N개의 집을 RGB 색으로 칠할 때 각 이웃한 집의 색이 같지 않도록 칠할 때, 최소 비용을 계산하는 문제이다.



입력은 위와 같이 집의 수 N과 각 집을 RGB로 칠할 때 드는 비용이 주어진다.

코드는 다음과 같이 작성할 수 있다.



minarr에는 i번째 집을 각각 RGB로 칠했을 때의 minimum 비용을 저장한다.

i-1번째까지의 최소 비용까지 이미 계산이 되었다고 가정하면 i번째 행은 다음과 같이 계산이 가능하다.

1) i번째 집을 R로 칠하는 경우

min(i-1번째 집을 G로 칠했을 때 누적 최소 비용, i-1번째 집을 B로 칠했을 때 누적 최소 비용) + i번째 집을 R로 칠하는 비용

2) i번째 집을 G로 칠하는 경우

min(i-1번째 집을 R로 칠했을 때 누적 최소 비용, i-1번째 집을 B로 칠했을 때 누적 최소 비용) + i번째 집을 G로 칠하는 비용

3) i번째 집을 B로 칠하는 경우

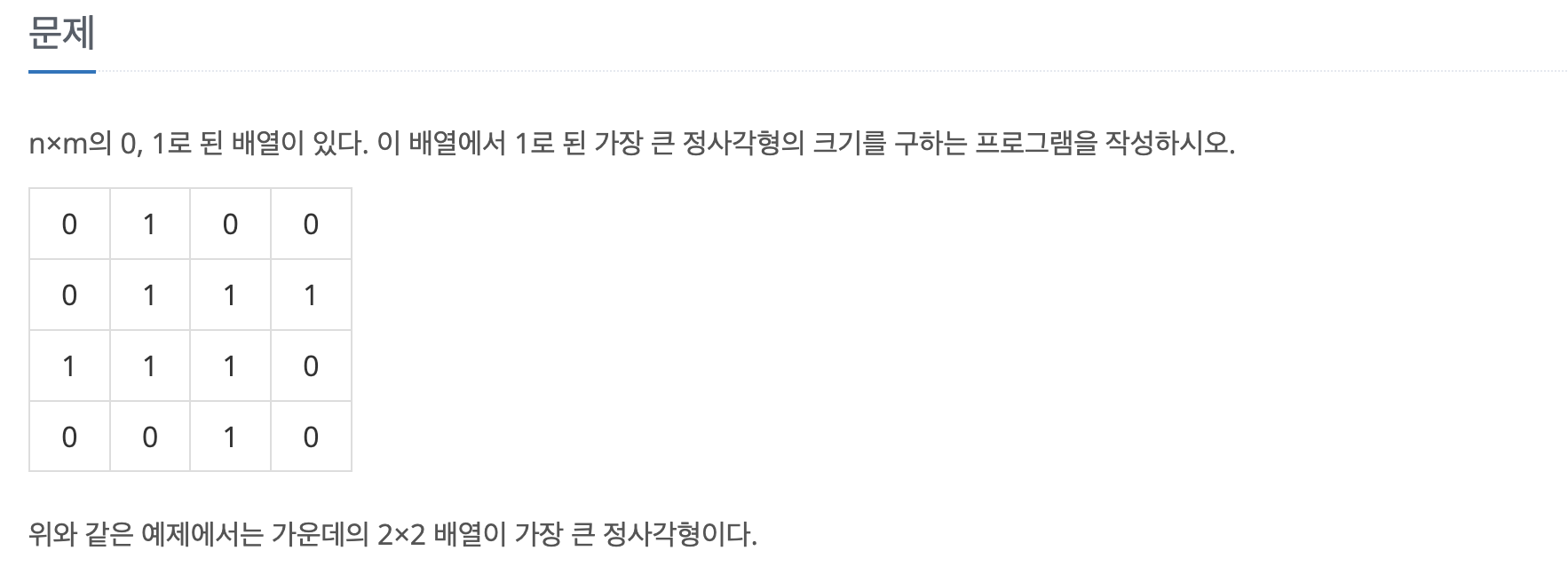
min(i-1번째 집을 R로 칠했을 때 누적 최소 비용, i-1번째 집을 G로 칠했을 때 누적 최소 비용) + i번째 집을 B로 칠하는 비용

이를 1번째 집부터 N번째 집까지 채우고, 최종적으로 N번째 집의 3가지 케이스 중 minimum 값을 채택함으로써 정답을 구할 수 있다.

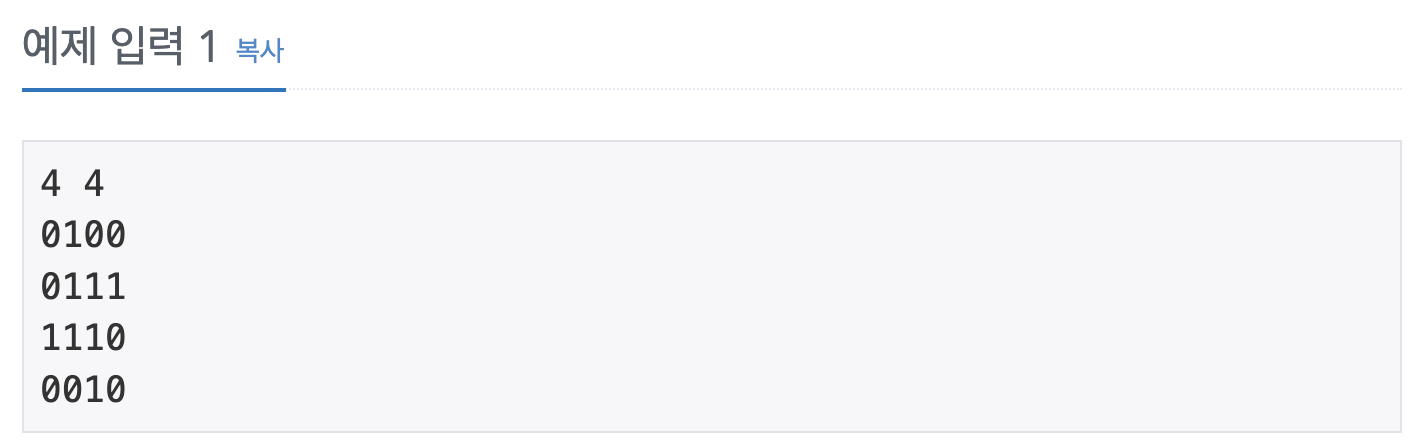
만약 Brute force 방식으로 이 문제를 해결한다면 1번 집에는 3가지의 선택 가능 경우가 있고, 나머지 집의 경우 2가지의 선택 가능 경우가 있다. 따라서 3\*2(n-1) 개의 경우를 모두 살펴봐야 최소 비용이 계산 가능하다.

이 문제를 Dynamic programming으로 풂으로써 3\*n의 경우만 고려해도 최적의 해를 계산이 가능하다. 따라서 DP를 사용함으로써 시간복잡도를 O(**2n**)에서 O(**n**)까지 줄일 수 있다.

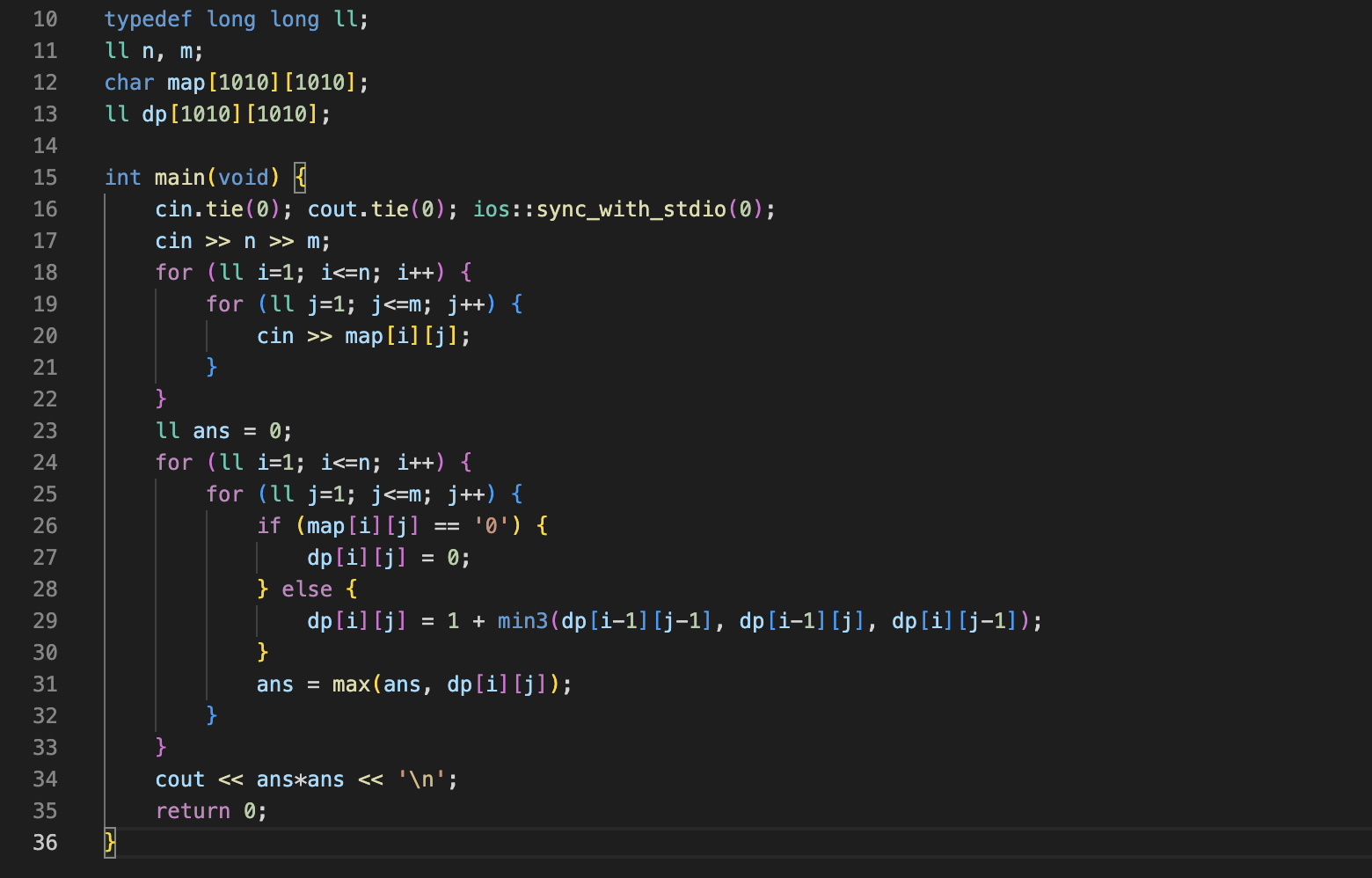
2-2)



이 문제는 배열에서 1로 이루어진 가장 큰 정사각형의 크기를 계산하는 문제이다. 입력은 배열의 크기 N \* M에 해당하는 N, M과 배열의 0, 1 구성이 들어온다.



코드는 다음과 같이 작성이 가능하다.



dp배열은 map의 (i, j)를 정사각형의 오른쪽 아래 꼭짓점이라고 생각했을 때 만들 수 있는 최대 정사각형의 변의 길이를 저장하는 배열이다.

따라서 map[i][j]가 0이면 이 cell을 포함하면서 만들 수 있는 정사각형이 만들어질 수 없으므로 0을 넣고, 1이라면 해당 cell의 왼쪽 방향, 대각선 방향, 윗 방향으로 뻗어나갈 수 있는 최대 변의 최솟값에 1을 더해 dp[i][j]에 저장한다.

이 dp 배열의 원소 중 가장 큰 값을 찾아 제곱한 값이 가장 큰 정사각형의 크기가 된다.

만약 Brute force로 문제를 해결한다면 다음과 같이 시간 복잡도를 계산할 수 있다.

1\*1 size 의 정사각형을 window로 옮기면서 해당 window 안의 모든 원소가 1인지 check한다고 하면 (1\*1) \* (n\*n) 만큼의 탐색이 필요하다. (1\*1)은 window 내부의 원소가 1인지 check하는 비용, (n\*n)는 window의 가짓수이다.

2\*2 size의 경우에도 (2\*2) \* ( \* )로 총 n\*m 만큼의 경우가 생기므로 총 탐색에 필요한 비용은 min(n, m) \* n \* m 이 된다.

Dynamic programming 기법을 활용하여 단순히 n \* m 크기의 배열을 차례로 채워넣고 최적 해를 계산할 수 있으므로 NM만큼의 비용이 든다.

따라서 DP를 사용함으로써 시간 복잡도를 O(**NM(N + M)**)에서 O(**NM**)까지 줄일 수 있다.