BJ_17070 파이프 옮기기

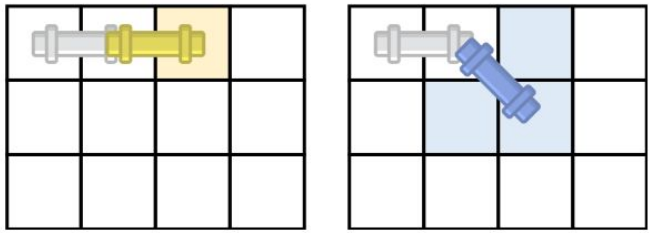
문제 제한 조건 : 1sec, 512MB

입력 조건 : 집 크기 N (3 <= N <= 16)
집 상태 (0 : 빈칸, 1 : 벽)

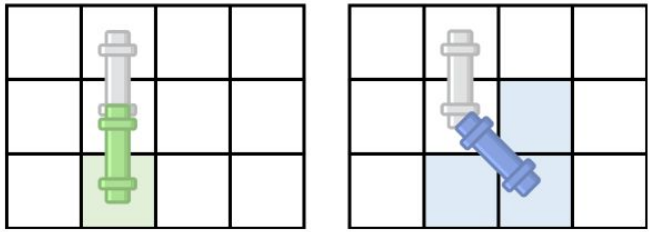
출력 조건 : 이동시킬 수 없을 경우에는 0 출력, 방법의 수는 1,000,000 이하

목표 : (1, 1)에서 N, N까지 가는데 드는 방법 수

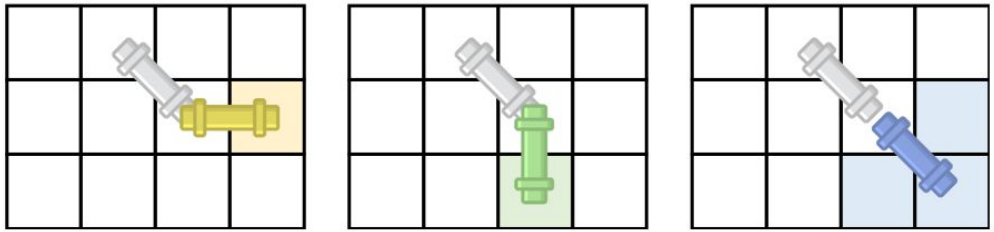
파이프를 어떻게 옮길 수 있나?



가로



세로



대각선

꼬리물기 방식

0, 45, 90도 방향

0도 이동시 (노랑색만 꼭 빈칸)

45도 이동시 (파란색만 꼭 빈칸)

90도 이동시 (초록색만 꼭 빈칸)

DFS로 풀어볼까?

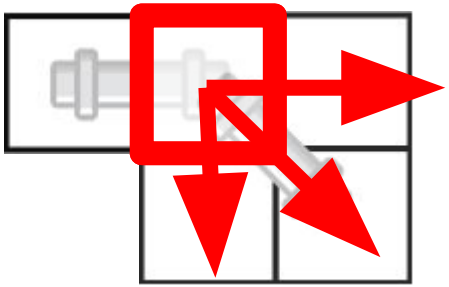
맵의 크기가 **최대 16까지,** 파이프는 **꼬리물기**로 되어있음

메모리는 512MB로 넉넉함

경우의 수가 한정적으로 보임

DFS로 풀어볼까?

꼬리물기 방식



시작점과 **다음점**을 반복해 끝점이 도착지점 => 경로 + 1

```
public static void movePipe(int y, int x, int endY, int endX) {
        // System.out.println(y + " " + x + " " + endY + " " + endX);
        if(endY == N - 1 && endX == N - 1) {
                answer += 1;
                return;
        }
```

다음점이 장애물일 경우엔 **진행을 하지 않는다**.

직전 파이프의 상태에 따라 다음점까지 **진행방법**이 달라진다.

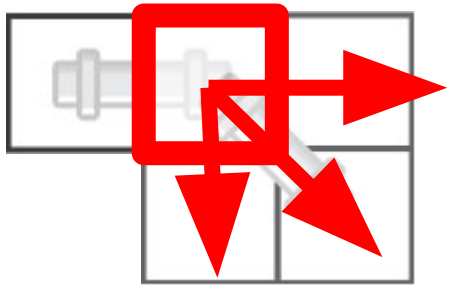
```
// 파이프의 상태를 알아봅니다.
public static int getPosition(int y, int x, int endY, int endX) {
        if(x == endX && y == endY - 1) {
                // 90도
                return 3;
        } else if(endX - 1 == x && y == endY) {
                // 0도
                return 1;
        } else if(endX - 1 == x && endY - 1 == y) {
                // 45도
                return 2;
        }
        return 0;
}
```

```
int position = getPosition(y, x, endY, endX);

switch(position) {
        case 1:
                if(!isOver(endY, endX + 1) && map[endY][endX + 1] == '0') movePipe(endY, endX, endY, endX + 1);
                if(!isOver(endY + 1, endX + 1) && map[endY + 1][endX + 1] == '0' &&  map[endY + 1][endX] == '0' &&  map[endY][endX + 1] == '
                break;
        case 2:
                if(!isOver(endY, endX + 1) && map[endY][endX + 1] == '0') movePipe(endY, endX, endY, endX + 1);
                if(!isOver(endY + 1, endX) && map[endY + 1][endX] == '0') movePipe(endY, endX, endY + 1, endX);
                if(!isOver(endY + 1, endX + 1) && map[endY + 1][endX + 1] == '0' &&  map[endY + 1][endX] == '0' &&  map[endY][endX + 1] == '
                break;
        case 3:
                if(!isOver(endY + 1, endX) && map[endY + 1][endX] == '0') movePipe(endY, endX, endY + 1, endX);
                if(!isOver(endY + 1, endX + 1) && map[endY + 1][endX + 1] == '0' &&  map[endY + 1][endX] == '0' &&  map[endY][endX + 1] == '
                break;
        default:
                break;
}
```

DFS로 풀어볼까?

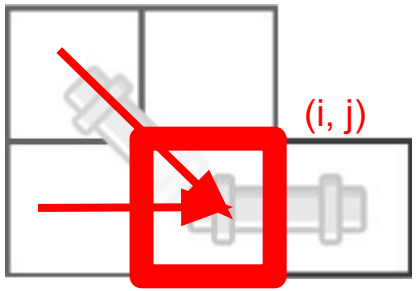
꼬리물기 방식으로 접근하면



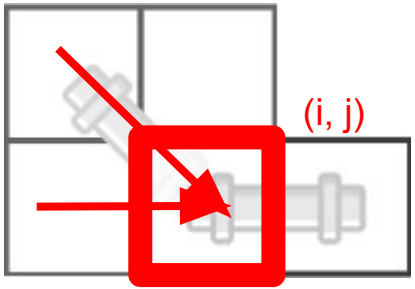
꼬리물기 방식으로 **다음지점이 범위를 벗어났는지 체크 필요**

49892933	djunnni	17070	맞았습니다!!	13332 KB	200 ms	Java 8 / 수정	2694 B	2시간 전

Q. 접근 방식을 바꿔서 (i, j)지점에서 받는 경로의 수로 바라보면 어떨까?



어떻게 (i, j)를 계산할까?



dp[i][j] 는 (i, j)에 도달하는데 가능한 방법 수

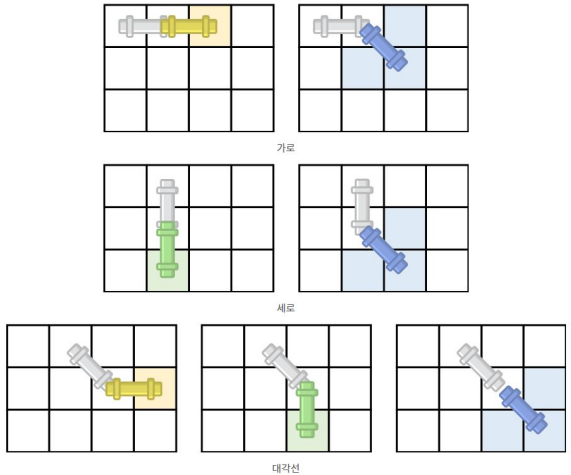
dp[i][j] = dp[i-1][j-1] () dp[i-1][j] () dp[i][j-1]

()에 무엇이 들어갈진 모르겠지만 연관관계를 찾아봐야겠다.

0, 45, 90도에 대해서 어떻게 처리를 하지? + 어떻게 파이프가 놓일 조건을 만족하지?

아까처럼 1,2,3으로 경우를 나눈 방식을 활용,

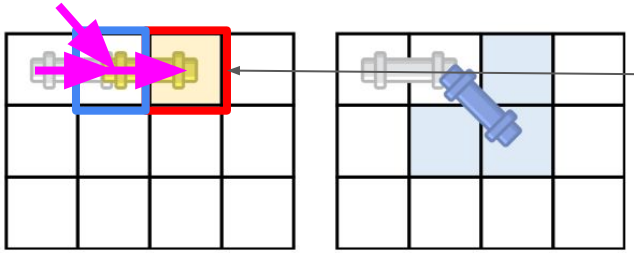
직접 경우를 더하는 방식 사용



dp[i][j][k] 는 (i, j)에 도달하는데 k가지 방향에 따른 경우를 계산

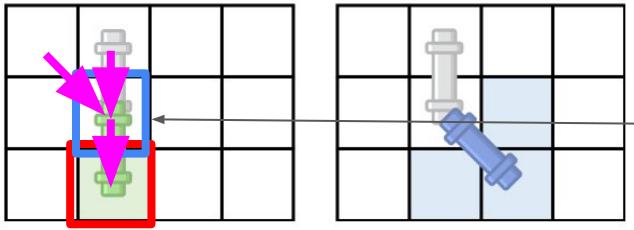
0 -> k : 0
45 -> k : 1
90 -> k : 2

어떻게 (i, j)를 계산할까?



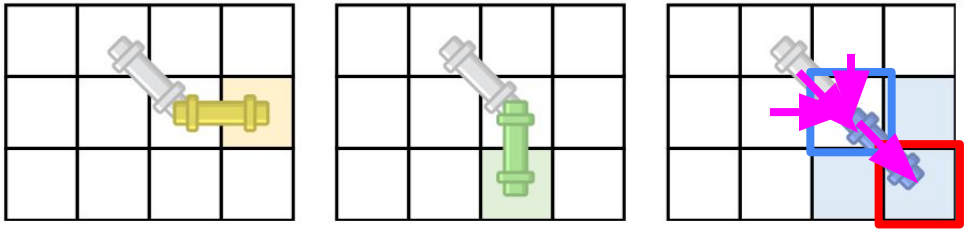
가로

dp[i][j][0] = dp[i][j - 1][0] + dp[i][j - 1][1]



세로

dp[i][j][2] = dp[i-1][j][1] + dp[i-1][j][2]



대각선

dp[i][j][2] =

dp[i - 1][j-1][0] + dp[i - 1][j-1][1] + dp[i - 1][j -1][2];

어떻게 (i, j)를 계산할까?

System.out.println(dp[N-1][N-1][0] + dp[N-1][N-1][1] + dp[N-1][N-1][2]);

```
dp[0][1][0] = 1; // (1,0)으로 1가지 가는 방법이 있다.

for(int i = 0; i < N; i++) {
        for(int j = 2; j < N; j++) {
                if(map[i][j] == '1') continue;
                if(!isOver(i, j - 1) && map[i][j-1] == '0')
                        dp[i][j][0] = dp[i][j - 1][0] + dp[i][j - 1][1];
                if(!isOver(i - 1, j) && map[i-1][j] == '0')
                        dp[i][j][2] = dp[i - 1][j][1] + dp[i - 1][j][2];
                if(!isOver(i - 1, j -1) && map[i - 1][j] == '0' && map[i][j - 1] == '0' && map[i - 1][j - 1] == '0') {
                        dp[i][j][1] = dp[i - 1][j-1][0] + dp[i - 1][j-1][1] + dp[i - 1][j -1 ][2];
                }
        }
}
```

DFS에 비해 불필요한 접근이 없어 더 빠르게 동작한다. 2 ^ N vs N ^ 2

49897078	djunnni	17070	맞았습니다!!	11764 KB	80 ms	Java 8 / 수정	1862 B	1시간 전
49892933	djunnni	17070	맞았습니다!!	13332 KB	200 ms	Java 8 / 수정	2694 B	2시간 전