688. Knight Probability in Chessboard

https://leetcode.com/problems/knight-probability-in-chessboard/

Approach #1: Dynamic Programming [Accepted]

Intuition and Algorithm

Let `f[r][c][steps]` be the probability of being on square `(r, c)` after `steps` steps. Based on how a knight moves, we have the following recursion:

$f[r][c][steps] = \sum_{dr,dc} f[r+dr][c+dc][steps-1]/8.0$

where the sum is taken over the eight (dr, dc) pairs $(2,1), (2,-1), (-2,1), (-2,-1), (1,2), (1,-2), (-1,2), (-1,-2)$.

Instead of using a three-dimensional array `f`, we will use two two-dimensional ones `dp` and `dp2`, storing the result of the two most recent layers we are working on. `dp2` will represent `f[][][steps]`, and `dp` will represent `f[][][steps-1]`.

```
#define inBoard 0 <= cr && cr < N && 0 <= cc && cc < N
class Solution {
public:
    int dX[8] = {2, 2, 1, 1, -1, -1, -2, -2};
    int dY[8] = {1, -1, 2, -2, 2, -2, 1, -1};


    double knightProbability(int N, int K, int r, int c) {
        vector<vector<double>> dp (N, vector<double> (N,0));
        dp[r][c]=1;
        for(;K>0;K--){
            vector<vector<double>> dp2 (N, vector<double> (N,0));
            for (int r = 0; r < N; r++) {
                for (int c = 0; c < N; c++) {
                    for (int k = 0; k < 8; k++) {
                        int cr = r + dX[k];
                        int cc = c + dY[k];
                        if (inBoard) {
                            dp2[cr][cc] += dp[r][c] / 8.0;
                        }
                    }
                }
            }
            dp = dp2;
        }
        double ans = 0.0;
        for ( auto a : dp ){
            for( auto b : a){
                ans += b;
            }
        }
        return ans;
```

```
    }
};
```

Approach 2

Matrix Exponentiation