Problem of the Week – Knight's Survival Probability

Company: Two Sigma
Difficulty: Medium / Hard
Topic: Dynamic Programming, Probability, Recursion

📌 Scenario

A knight on a chessboard moves with its usual L-shaped moves. If it moves off the board at any step it disappears (it cannot come back). We move the knight **k** times; each move is chosen uniformly at random among the 8 possible knight moves (even if some of those land off-board). Compute the probability that after **k** moves the knight still remains on the board.

This models random-walk survival probability with absorbing (off-board) states.

📌 Problem Statement

Given:

- An 8 × 8 chessboard (rows and columns indexed from 0..7).
- A starting square `(r, c)`.
- An integer `k` (number of moves).

At each move the knight chooses uniformly from its 8 possible knight moves; if the chosen move lands off the board the knight is lost and cannot return. Return the probability that after exactly `k` moves the knight is still on the board.

🔷 Input Format

- Integers `r`, `c`, `k` where `0 ≤ r, c ≤ 7` and `k ≥ 0`.

🔷 Output Format

- A floating point number: the probability ($0 \le p \le 1$) that the knight stays on the board after `k` moves.

🔷 Examples

1. **Example 1**
   - Input: `r = 0, c = 0, k = 1`
   - Explanation: From a corner the knight has 2 legal on-board moves out of 8 total choices → probability = `2/8 = 0.25`.
   - Output: `0.25`
2. **Example 2**
   - Input: `r = 0, c = 0, k = 2`
   - Output: `0.0625` (example value — see DP below for computation)

(Note: example 2 value depends on accounting for off-board choices at each step; the DP code below computes exact values.)

🔑 Approaches

1. Dynamic Programming (bottom-up) — recommended

Let `dp[t][i][j]` = probability that the knight is at cell `(i,j)` after `t` moves and still on the board.

- Initialize `dp[0][r][c] = 1.0`.
- For each move `t` from `0` to `k-1`, distribute `dp[t][i][j]` equally among the 8 knight moves:
- `dp[t+1][ni][nj] += dp[t][i][j] * (1/8)`

  if `(ni,nj)` is inside the board; moves that go off-board are not added (their probability mass is lost).

- After `k` moves, probability = sum over all `i,j` of `dp[k][i][j]`.

Time: `O(k * 64 * 8)` = `O(k)`, constant factor small.
Space: `O(64)` if you roll the DP over time (only keep current and next layer).

2. Recursion + Memoization (top-down)

Define `P(t, i, j)` = probability to be on board after `t` remaining moves starting from `(i,j)`:

```
P(0, i, j) = 1  if (i,j) on board
P(t, i, j) = (1/8) * sum_{(ni,nj) in knightMoves(i,j)} P(t-1, ni, nj)
```

Memoize on `(t, i, j)` to avoid recomputation. Equivalent complexity to DP.

3. Matrix exponentiation (advanced)

Treat the 64×64 transition matrix `T` (each off-board move leads to nowhere); probability vector after `k` steps is `v0 * T^k`. This is overkill for small fixed board sizes but possible.

✅ Complexity

- Time: `O(k * 64 * 8)` ≈ `O(k)` with small constant (64 squares).
- Space: `O(64)` if using two 8×8 layers (current/next).

🔗 Practice Link / Related Problem

- LeetCode: **688. Knight Probability in Chessboard** — same problem (change board size as parameter): https://leetcode.com/problems/knight-probability-in-chessboard/

🔎 Clarification Note (common variants)

Two variants appear in interview / online problems:

- **Variant A (standard / LeetCode 688):** Each step the knight chooses uniformly among the 8 possible moves; if that chosen move is off-board the knight disappears. (Use divisor = 8.)
- **Variant B (alternate):** Each step the knight chooses uniformly among the *legal* on-board moves only. (Use divisor = number of legal moves for that cell.)

Make sure to confirm which variant the interviewer expects. The DP code above implements **Variant A** (uniform over 8 possible moves). If you want Variant B, change the distribution accordingly.