Given an NxN chessboard and a Knight at position (x,y). The Knight has to take exactly K steps, where at each step it chooses any of the 8 directions uniformly at random. What is the probability that the Knight remains in the chessboard after taking K steps, with the condition that it can’t enter the board again once it leaves it? Solve using Dynamic programming.

Nischay Nagar(IIT2019198), Abhishek Bithu(IIT2019199) and Raj Chandra(IIT2019200)
4th Semester B.Tech, Department Of IT, Indian Institute Of Information Technology
Allahabad, Prayagraj, India

Abstract — In this paper we have approached the problem of finding the probability that the knight remains in the chessboard after k number of steps.This paper shows how this problem can be solved using dynamic programming.

Keywords

- Dynamic programming

Algorithm

Step 1:

Define direction vectors for the knight.

```
            Ex:- dx[ ] = {1, 2, 2, 1, -1, -2, -2, -1}
                 dy[ ] = {2, 1, -1, -1, -2, -1, 1, 2}
```

Step 2:

Take an array dp[N, N, steps + 1] which will store the probability of reaching (x,y) after (steps) number of moves.

Step 3:

Base case: if the number of steps is 0, then the probability that the Knight will remain inside the board is 1

Step 4:

Take the position (x, y) after s number of steps.

Step 5:

Take prob = 0.0 then check for each position reachable from (x, y) using the direction vectors and store it in a new position (nx, ny).

Step 6:

Check if this new position (nx, ny) is inside of the chessboard, if yes then add dp1[nx][ny][s - 1] / 8.0 to prob.

Step 7:

Store the prob in dp[x][y][s].

Step 8:

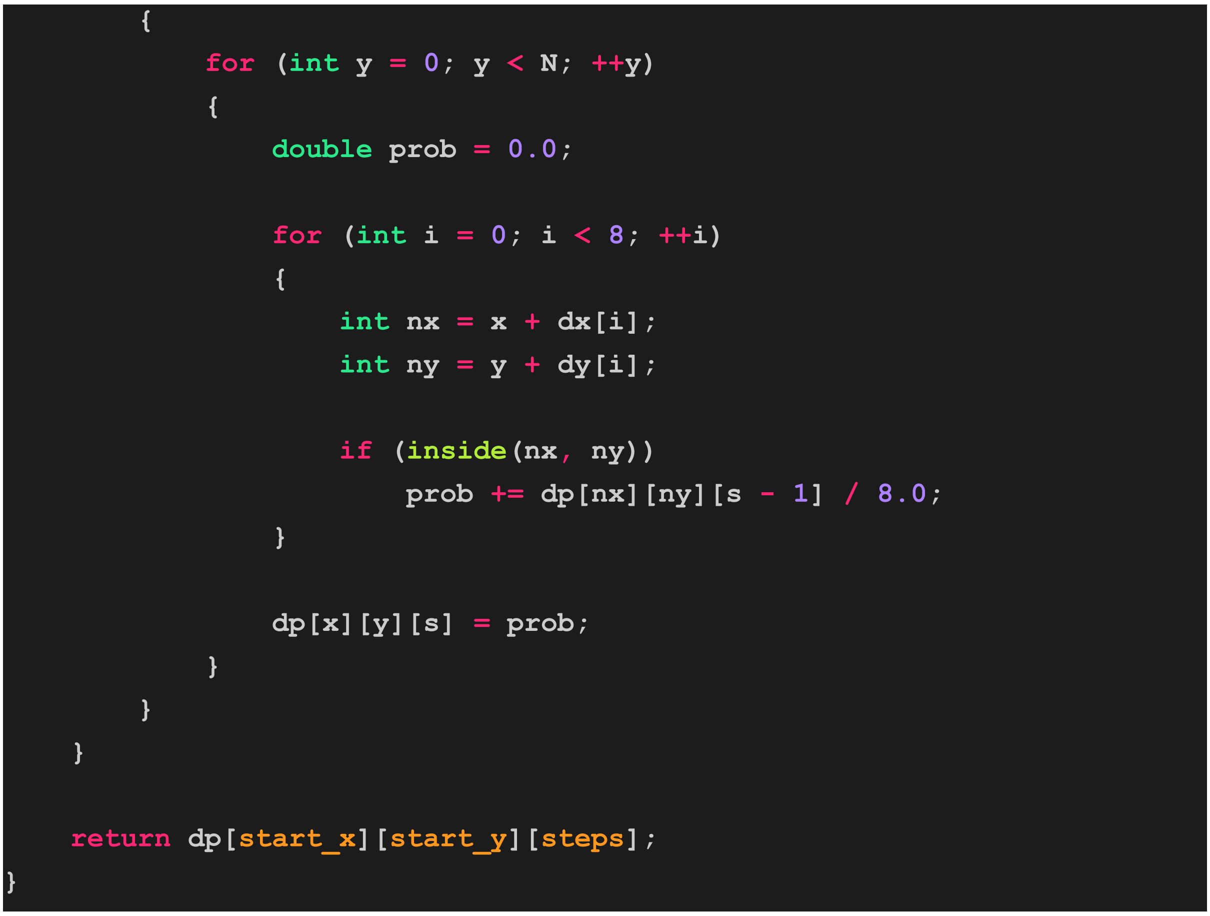
Keep repeating for the given number of steps.

Step 9:

The required probability will be stored in dp[start_x][start_y][k], where (start_x, start_y) are the given initial position of the knight and k is the number of steps.

Pseudo Code

```
int dx[] = {1, 2, 2, 1, -1, -2, -2, -1};
int dy[] = {2, 1, -1, -2, -2, -1, 1, 2};

bool inside(int x, int y)
{
    return (x >= 0 and x < N and y >= 0 and y < N);
}

double findProb(int start_x, int start_y, int steps)
{
    double dp[N][N][steps + 1];

    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            dp[i][j][0] = 1;

    for (int s = 1; s <= steps; ++s)
    {

        for (int x = 0; x < N; ++x)
```



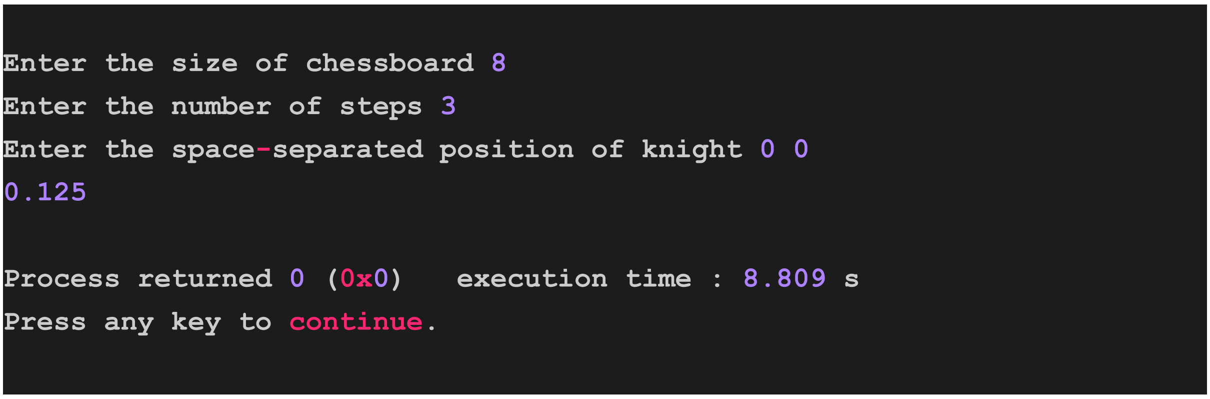
```
        {
            for (int y = 0; y < N; ++y)
            {
                double prob = 0.0;

                for (int i = 0; i < 8; ++i)
                {
                    int nx = x + dx[i];
                    int ny = y + dy[i];

                    if (inside(nx, ny))
                        prob += dp[nx][ny][s - 1] / 8.0;
                }

                dp[x][y][s] = prob;
            }
        }
    }

    return dp[start_x][start_y][steps];
}
```

Complexity analysis

Time Complexity: O(NxNxK), where N is the size of the board and K is the number of steps.

Space Complexity: O(NxNxK), where N is the size of the board and K is the number of steps.

Output



```
Enter the size of chessboard 8
Enter the number of steps 3
Enter the space-separated position of knight 0 0
0.125

Process returned 0 (0x0)   execution time : 8.809 s
Press any key to continue.
```

Conclusion

The method discussed above shows the mentioned problem can be solved using dynamic programming with a time and space complexity of O(NxNxK), where N is the size of the chessboard and K is the number of steps.

References

- https://www.geeksforgeeks.org/dynamic-programming/
- https://www.geeksforgeeks.org/understanding-time-complexity-simple-examples/