Problem 688: Knight Probability in Chessboard

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

On an

n x n

chessboard, a knight starts at the cell

(row, column)

and attempts to make exactly

k

moves. The rows and columns are

0-indexed

, so the top-left cell is

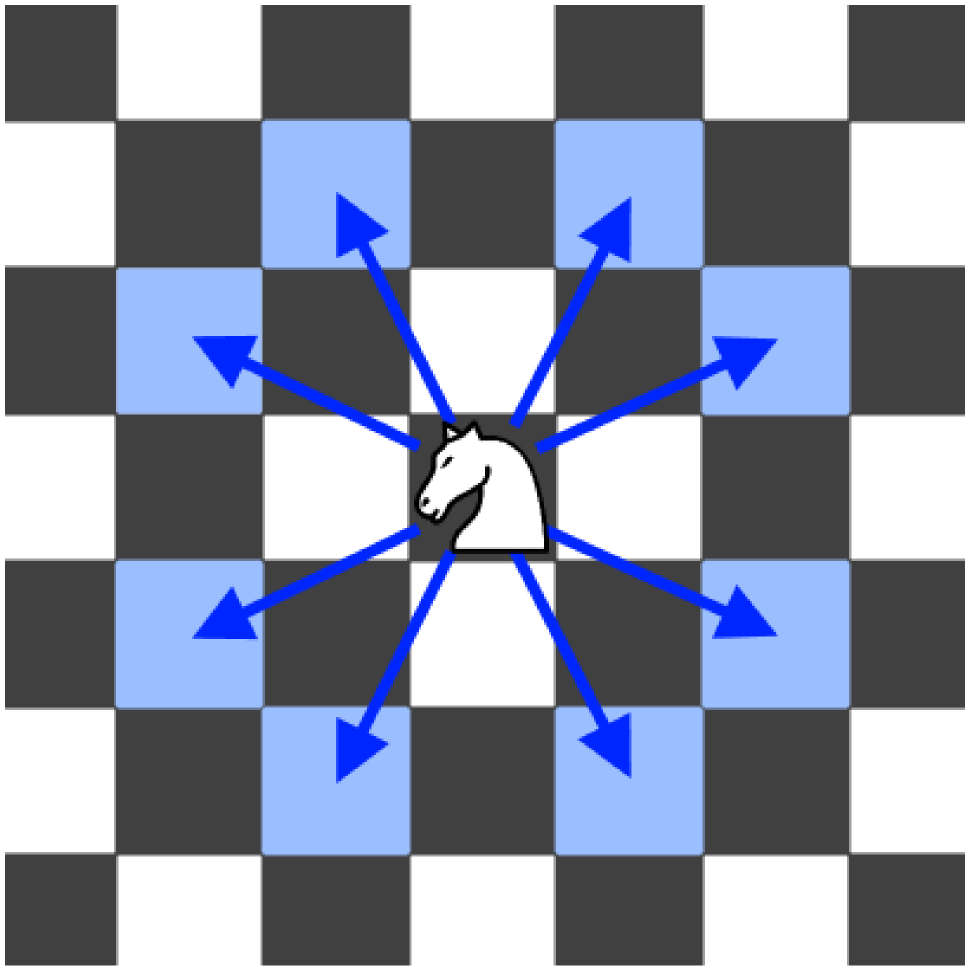
(0, 0)

, and the bottom-right cell is

(n - 1, n - 1)

.

A chess knight has eight possible moves it can make, as illustrated below. Each move is two cells in a cardinal direction, then one cell in an orthogonal direction.



Each time the knight is to move, it chooses one of eight possible moves uniformly at random (even if the piece would go off the chessboard) and moves there.

The knight continues moving until it has made exactly

k

moves or has moved off the chessboard.

Return

the probability that the knight remains on the board after it has stopped moving

.

Example 1:

Input:

n = 3, k = 2, row = 0, column = 0

Output:

0.06250

Explanation:

There are two moves (to (1,2), (2,1)) that will keep the knight on the board. From each of those positions, there are also two moves that will keep the knight on the board. The total probability the knight stays on the board is 0.0625.

Example 2:

Input:

n = 1, k = 0, row = 0, column = 0

Output:

1.00000

Constraints:

1 <= n <= 25

0 <= k <= 100

0 <= row, column <= n - 1

Code Snippets

C++:

```
class Solution {
public:
double knightProbability(int n, int k, int row, int column) {

}
};
```

Java:

```
class Solution {
public double knightProbability(int n, int k, int row, int column) {

}
}
```

Python3:

```
class Solution:
def knightProbability(self, n: int, k: int, row: int, column: int) -> float:
```

Python:

```
class Solution(object):
def knightProbability(self, n, k, row, column):
"""
:type n: int
:type k: int
:type row: int
:type column: int
:rtype: float
"""
```

JavaScript:

```
/**
* @param {number} n
* @param {number} k
* @param {number} row
* @param {number} column
* @return {number}
*/
var knightProbability = function(n, k, row, column) {
};
```

TypeScript:

```
function knightProbability(n: number, k: number, row: number, column:
number): number {

};
```

C#:

```
public class Solution {
public double KnightProbability(int n, int k, int row, int column) {

}
}
```

C:

```
double knightProbability(int n, int k, int row, int column) {

}
```

Go:

```
func knightProbability(n int, k int, row int, column int) float64 {

}
```

Kotlin:

```
class Solution {
fun knightProbability(n: Int, k: Int, row: Int, column: Int): Double {

}
}
```

Swift:

```
class Solution {
func knightProbability(_ n: Int, _ k: Int, _ row: Int, _ column: Int) ->
Double {
}
}
```

Rust:

```
impl Solution {
pub fn knight_probability(n: i32, k: i32, row: i32, column: i32) -> f64 {

}
}
```

Ruby:

```
# @param {Integer} n
# @param {Integer} k
# @param {Integer} row
# @param {Integer} column
# @return {Float}
def knight_probability(n, k, row, column)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @param Integer $row
* @param Integer $column
* @return Float
*/
function knightProbability($n, $k, $row, $column) {

}
}
```

Dart:

```
class Solution {
double knightProbability(int n, int k, int row, int column) {

}
}
```

Scala:

```
object Solution {
def knightProbability(n: Int, k: Int, row: Int, column: Int): Double = {

}
}
```

Elixir:

```
defmodule Solution do
@spec knight_probability(n :: integer, k :: integer, row :: integer, column
:: integer) :: float
def knight_probability(n, k, row, column) do

end
end
```

Erlang:

```
-spec knight_probability(N :: integer(), K :: integer(), Row :: integer(),
Column :: integer()) -> float().
knight_probability(N, K, Row, Column) ->
.
```

Racket:

```
(define/contract (knight-probability n k row column)
(-> exact-integer? exact-integer? exact-integer? exact-integer? flonum?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Knight Probability in Chessboard
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
double knightProbability(int n, int k, int row, int column) {

}
};
```

Java Solution:

```
/**
* Problem: Knight Probability in Chessboard
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public double knightProbability(int n, int k, int row, int column) {

}
}
```

Python3 Solution:

```
"""
Problem: Knight Probability in Chessboard
Difficulty: Medium
Tags: dp
```

```
Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def knightProbability(self, n: int, k: int, row: int, column: int) -> float:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def knightProbability(self, n, k, row, column):
"""
:type n: int
:type k: int
:type row: int
:type column: int
:rtype: float
"""
```

JavaScript Solution:

```
/**
* Problem: Knight Probability in Chessboard
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} n
* @param {number} k
* @param {number} row
* @param {number} column
* @return {number}
*/
```

```
var knightProbability = function(n, k, row, column) {

};
```

TypeScript Solution:

```
/**
* Problem: Knight Probability in Chessboard
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function knightProbability(n: number, k: number, row: number, column:
number): number {

};
```

C# Solution:

```
/*
* Problem: Knight Probability in Chessboard
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public double KnightProbability(int n, int k, int row, int column) {

}
}
```

C Solution:

```
/*
* Problem: Knight Probability in Chessboard
* Difficulty: Medium
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

double knightProbability(int n, int k, int row, int column) {

}
```

Go Solution:

```
// Problem: Knight Probability in Chessboard
// Difficulty: Medium
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func knightProbability(n int, k int, row int, column int) float64 {

}
```

Kotlin Solution:

```
class Solution {
fun knightProbability(n: Int, k: Int, row: Int, column: Int): Double {

}
}
```

Swift Solution:

```
class Solution {
func knightProbability(_ n: Int, _ k: Int, _ row: Int, _ column: Int) ->
Double {
}
}
```

Rust Solution:

```
// Problem: Knight Probability in Chessboard
// Difficulty: Medium
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn knight_probability(n: i32, k: i32, row: i32, column: i32) -> f64 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @param {Integer} k
# @param {Integer} row
# @param {Integer} column
# @return {Float}
def knight_probability(n, k, row, column)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $n
* @param Integer $k
* @param Integer $row
* @param Integer $column
* @return Float
*/
function knightProbability($n, $k, $row, $column) {

}
}
```

Dart Solution:

```
class Solution {
double knightProbability(int n, int k, int row, int column) {

}
}
```

Scala Solution:

```
object Solution {
def knightProbability(n: Int, k: Int, row: Int, column: Int): Double = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec knight_probability(n :: integer, k :: integer, row :: integer, column
:: integer) :: float
def knight_probability(n, k, row, column) do

end
end
```

Erlang Solution:

```
-spec knight_probability(N :: integer(), K :: integer(), Row :: integer(),
Column :: integer()) -> float().
knight_probability(N, K, Row, Column) ->
.
```

Racket Solution:

```
(define/contract (knight-probability n k row column)
(-> exact-integer? exact-integer? exact-integer? exact-integer? flonum?)
)
```