Problem 935: Knight Dialer

Problem Information

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

Problem Description

The chess knight has a

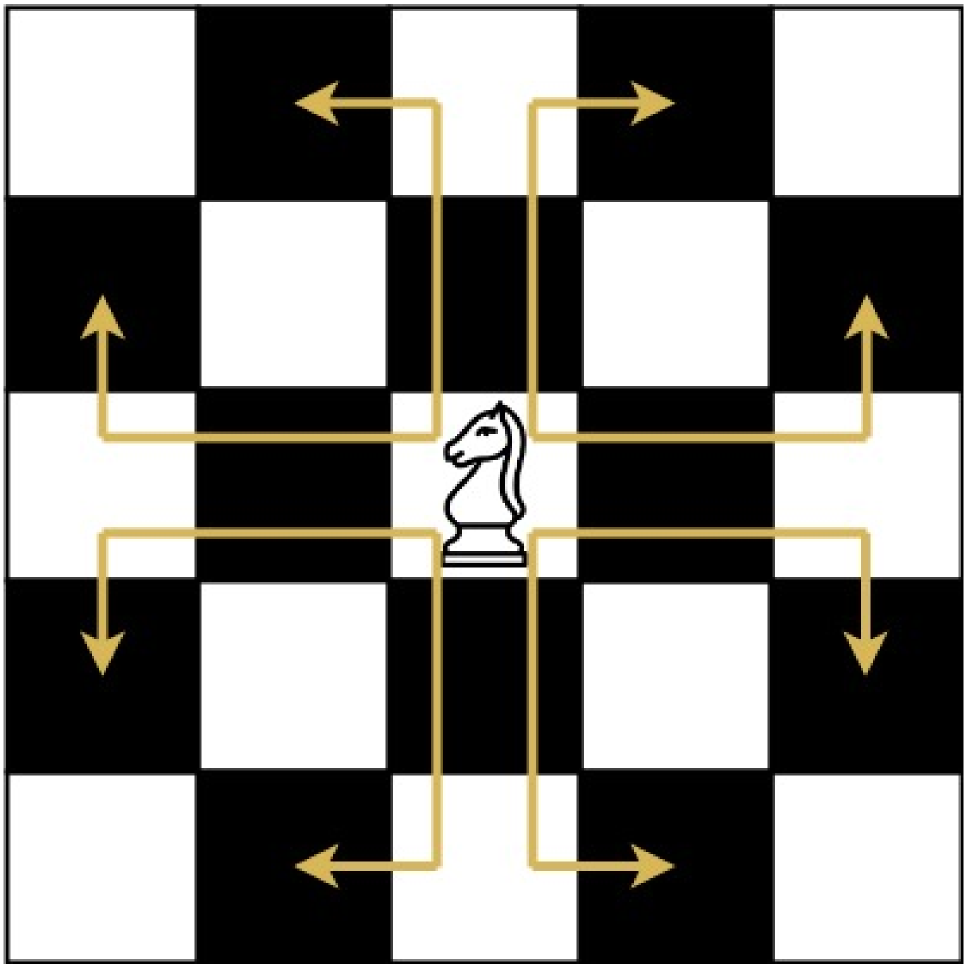
unique movement

, it may move two squares vertically and one square horizontally, or two squares horizontally and one square vertically (with both forming the shape of an

L

). The possible movements of chess knight are shown in this diagram:

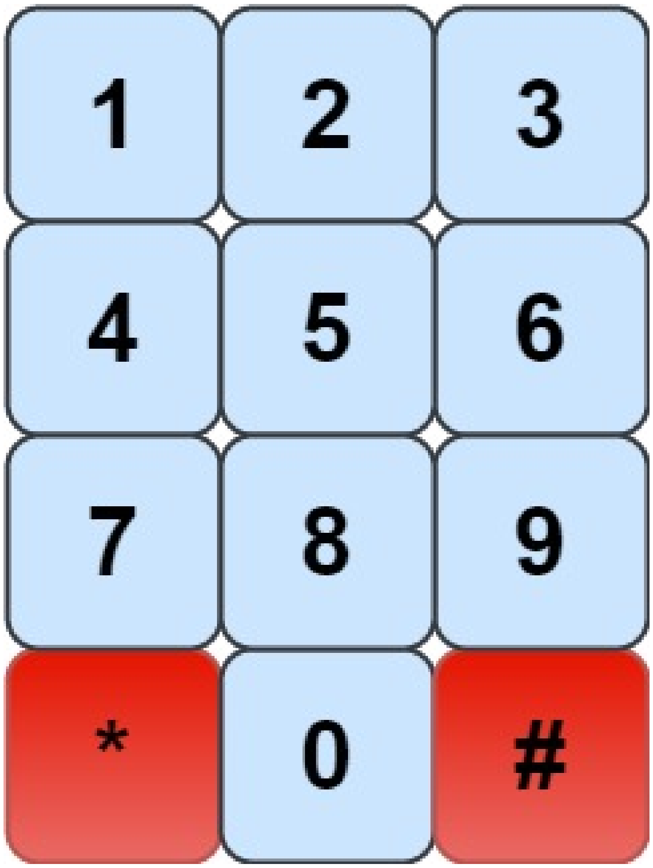
A chess knight can move as indicated in the chess diagram below:



We have a chess knight and a phone pad as shown below, the knight

can only stand on a numeric cell

(i.e. blue cell).



Given an integer

n

, return how many distinct phone numbers of length

n

we can dial.

You are allowed to place the knight

on any numeric cell

initially and then you should perform

n - 1

jumps to dial a number of length

n

. All jumps should be

valid

knight jumps.

As the answer may be very large,

return the answer modulo

10

9

+ 7

.

Example 1:

Input:

n = 1

Output:

10

Explanation:

We need to dial a number of length 1, so placing the knight over any numeric cell of the 10 cells is sufficient.

Example 2:

Input:

n = 2

Output:

20

Explanation:

All the valid number we can dial are [04, 06, 16, 18, 27, 29, 34, 38, 40, 43, 49, 60, 61, 67, 72, 76, 81, 83, 92, 94]

Example 3:

Input:

n = 3131

Output:

136006598

Explanation:

Please take care of the mod.

Constraints:

1 <= n <= 5000

Code Snippets

C++:

```
class Solution {
public:
int knightDialer(int n) {

}
};
```

Java:

```
class Solution {
public int knightDialer(int n) {

}
}
```

Python3:

```
class Solution:
def knightDialer(self, n: int) -> int:
```

Python:

```
class Solution(object):
def knightDialer(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} n
* @return {number}
*/
var knightDialer = function(n) {

};
```

TypeScript:

```
function knightDialer(n: number): number {

};
```

C#:

```
public class Solution {
public int KnightDialer(int n) {
}
}
```

C:

```
int knightDialer(int n) {

}
```

Go:

```
func knightDialer(n int) int {

}
```

Kotlin:

```
class Solution {
fun knightDialer(n: Int): Int {

}
}
```

Swift:

```
class Solution {
func knightDialer(_ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn knight_dialer(n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} n
# @return {Integer}
def knight_dialer(n)

end
```

PHP:

```
class Solution {

/**
* @param Integer $n
* @return Integer
*/
function knightDialer($n) {

}
}
```

Dart:

```
class Solution {
int knightDialer(int n) {

}
}
```

Scala:

```
object Solution {
def knightDialer(n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec knight_dialer(n :: integer) :: integer
def knight_dialer(n) do

end
end
```

Erlang:

```
-spec knight_dialer(N :: integer()) -> integer().
knight_dialer(N) ->
.
```

Racket:

```
(define/contract (knight-dialer n)
(-> exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Knight Dialer
* 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:
int knightDialer(int n) {

}
};
```

Java Solution:

```
/**
* Problem: Knight Dialer
* 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 int knightDialer(int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Knight Dialer
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 knightDialer(self, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def knightDialer(self, n):
"""
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Knight Dialer
* 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
* @return {number}
*/
var knightDialer = function(n) {

};
```

TypeScript Solution:

```
/**
* Problem: Knight Dialer
* 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 knightDialer(n: number): number {

};
```

C# Solution:

```
/*
* Problem: Knight Dialer
* 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 int KnightDialer(int n) {

}
}
```

C Solution:

```
/*
* Problem: Knight Dialer
* 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
*/

int knightDialer(int n) {

}
```

Go Solution:

```
// Problem: Knight Dialer
// 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 knightDialer(n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun knightDialer(n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func knightDialer(_ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Knight Dialer
// 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_dialer(n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} n
# @return {Integer}
def knight_dialer(n)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer $n
* @return Integer
*/
function knightDialer($n) {

}
}
```

Dart Solution:

```
class Solution {
int knightDialer(int n) {

}
}
```

Scala Solution:

```
object Solution {
def knightDialer(n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec knight_dialer(n :: integer) :: integer
def knight_dialer(n) do

end
end
```

Erlang Solution:

```
-spec knight_dialer(N :: integer()) -> integer().
knight_dialer(N) ->
.
```

Racket Solution:

```
(define/contract (knight-dialer n)
(-> exact-integer? exact-integer?)
)
```