Problem 909: Snakes and Ladders

Problem Information

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

Problem Description

You are given an

n x n

integer matrix

board

where the cells are labeled from

1

to

n

2

in a

Boustrophedon style

starting from the bottom left of the board (i.e.

board[n - 1][0]

) and alternating direction each row.

You start on square

1

of the board. In each move, starting from square

curr

, do the following:

Choose a destination square

next

with a label in the range

[curr + 1, min(curr + 6, n

2

)]

.

This choice simulates the result of a standard

6-sided die roll

: i.e., there are always at most 6 destinations, regardless of the size of the board.

If

next

has a snake or ladder, you

must

move to the destination of that snake or ladder. Otherwise, you move to

next

.

The game ends when you reach the square

n

2

.

A board square on row

r

and column

c

has a snake or ladder if

board[r][c] != -1

. The destination of that snake or ladder is

board[r][c]

. Squares

1

and

n

2

are not the starting points of any snake or ladder.

Note that you only take a snake or ladder at most once per dice roll. If the destination to a snake or ladder is the start of another snake or ladder, you do

not

follow the subsequent snake or ladder.

For example, suppose the board is

[[-1,4],[-1,3]]

, and on the first move, your destination square is

2

. You follow the ladder to square

3

, but do

not

follow the subsequent ladder to

4

.

Return

the least number of dice rolls required to reach the square

n

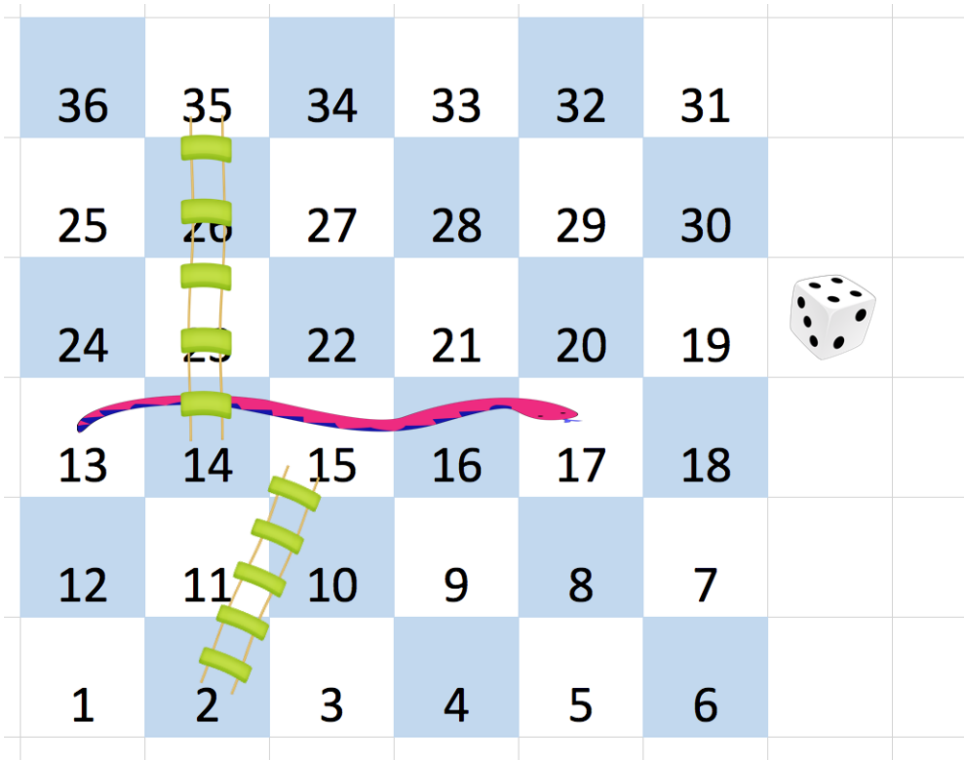
2

. If it is not possible to reach the square, return

-1

.

Example 1:



Input:

board = [[-1,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,-1],[-1,-1,-1,-1,-1,-1],[-1,35,-1,-1,13,-1],[-1,-1,-1,-1,-1,-1],[-1,15,-1,-1,-1,-1]]

Output:

4

Explanation:

In the beginning, you start at square 1 (at row 5, column 0). You decide to move to square 2 and must take the ladder to square 15. You then decide to move to square 17 and must take the snake to square 13. You then decide to move to square 14 and must take the ladder to square 35. You then decide to move to square 36, ending the game. This is the lowest possible number of moves to reach the last square, so return 4.

Example 2:

Input:

board = [[-1,-1],[-1,3]]

Output:

1

Constraints:

n == board.length == board[i].length

2 <= n <= 20

board[i][j]

is either

-1

or in the range

[1, n

2

]

.

The squares labeled

1

and

n

2

are not the starting points of any snake or ladder.

Code Snippets

C++:

```
class Solution {
public:
int snakesAndLadders(vector<vector<int>>& board) {

}
};
```

Java:

```
class Solution {
public int snakesAndLadders(int[][] board) {

}
}
```

Python3:

```
class Solution:
def snakesAndLadders(self, board: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def snakesAndLadders(self, board):
"""
:type board: List[List[int]]
```

```
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} board
* @return {number}
*/
var snakesAndLadders = function(board) {

};
```

TypeScript:

```
function snakesAndLadders(board: number[][]): number {

};
```

C#:

```
public class Solution {
public int SnakesAndLadders(int[][] board) {

}
}
```

C:

```
int snakesAndLadders(int** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func snakesAndLadders(board [][]int) int {

}
```

Kotlin:

```
class Solution {
fun snakesAndLadders(board: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func snakesAndLadders(_ board: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn snakes_and_ladders(board: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} board
# @return {Integer}
def snakes_and_ladders(board)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $board
* @return Integer
*/
function snakesAndLadders($board) {

}
}
```

Dart:

```
class Solution {
int snakesAndLadders(List<List<int>> board) {

}
}
```

Scala:

```
object Solution {
def snakesAndLadders(board: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec snakes_and_ladders(board :: [[integer]]) :: integer
def snakes_and_ladders(board) do

end
end
```

Erlang:

```
-spec snakes_and_ladders(Board :: [[integer()]]) -> integer().
snakes_and_ladders(Board) ->
.
```

Racket:

```
(define/contract (snakes-and-ladders board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Snakes and Ladders
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int snakesAndLadders(vector<vector<int>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Snakes and Ladders
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int snakesAndLadders(int[][] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Snakes and Ladders
Difficulty: Medium
Tags: array, search
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def snakesAndLadders(self, board: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def snakesAndLadders(self, board):
"""
:type board: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Snakes and Ladders
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} board
* @return {number}
*/
var snakesAndLadders = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Snakes and Ladders
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function snakesAndLadders(board: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Snakes and Ladders
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int SnakesAndLadders(int[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Snakes and Ladders
* Difficulty: Medium
* Tags: array, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

int snakesAndLadders(int** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Snakes and Ladders
// Difficulty: Medium
// Tags: array, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func snakesAndLadders(board [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun snakesAndLadders(board: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func snakesAndLadders(_ board: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Snakes and Ladders
// Difficulty: Medium
// Tags: array, search
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn snakes_and_ladders(board: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} board
# @return {Integer}
def snakes_and_ladders(board)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $board
* @return Integer
*/
function snakesAndLadders($board) {

}
}
```

Dart Solution:

```
class Solution {
int snakesAndLadders(List<List<int>> board) {

}
}
```

Scala Solution:

```
object Solution {
def snakesAndLadders(board: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec snakes_and_ladders(board :: [[integer]]) :: integer
def snakes_and_ladders(board) do

end
end
```

Erlang Solution:

```
-spec snakes_and_ladders(Board :: [[integer()]]) -> integer().
snakes_and_ladders(Board) ->
.
```

Racket Solution:

```
(define/contract (snakes-and-ladders board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```