# Problem 2596: Check Knight Tour Configuration

## Problem Information

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

## Problem Description

There is a knight on an

n x n

chessboard. In a valid configuration, the knight starts

at the top-left cell

of the board and visits every cell on the board

exactly once

.

You are given an

n x n

integer matrix

grid

consisting of distinct integers from the range

[0, n * n - 1]

where

grid[row][col]

indicates that the cell

(row, col)

is the

grid[row][col]

th

cell that the knight visited. The moves are

0-indexed

.

Return

true

if

grid

represents a valid configuration of the knight's movements or

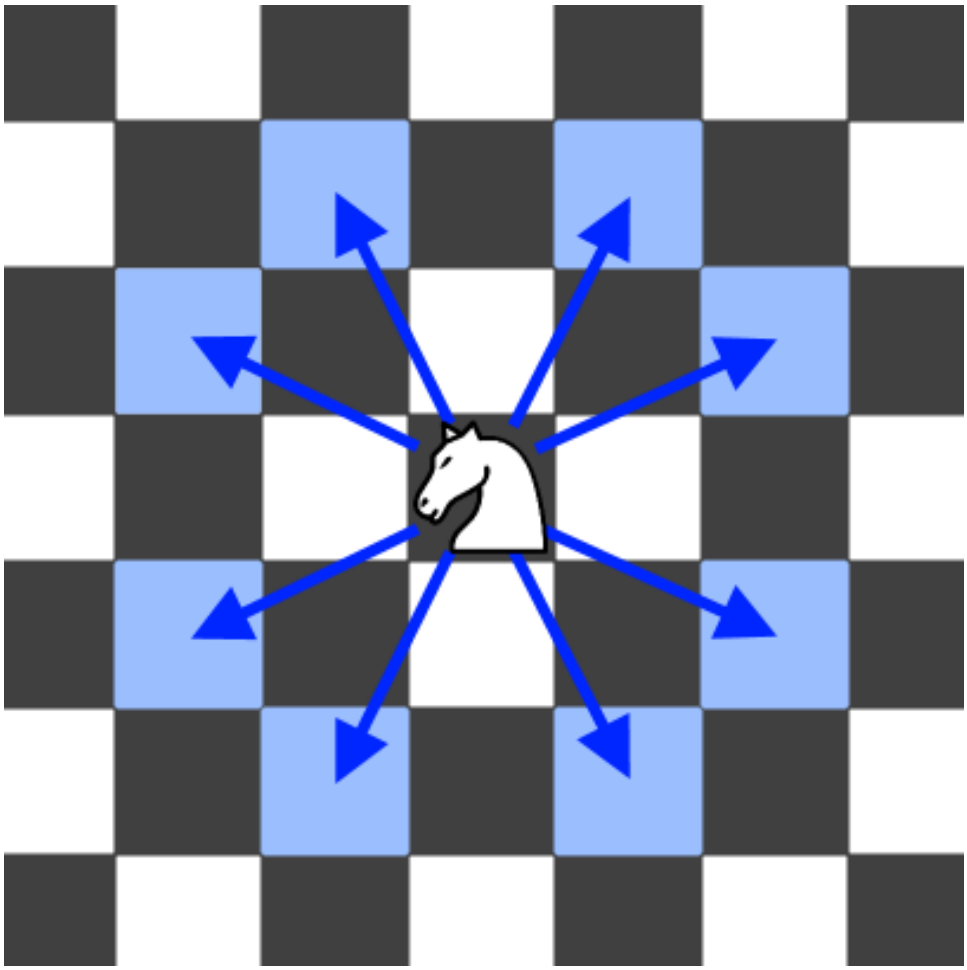
false

otherwise

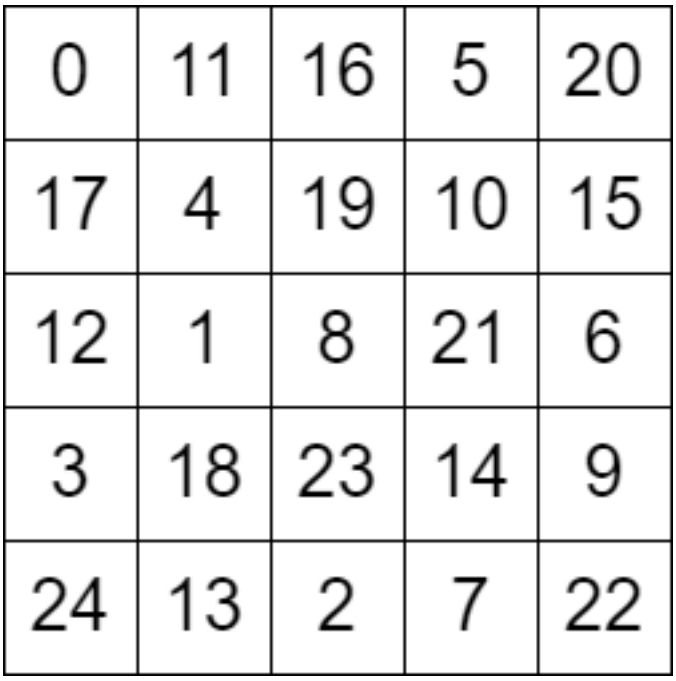
.

Note

that a valid knight move consists of moving two squares vertically and one square horizontally, or two squares horizontally and one square vertically. The figure below illustrates all the possible eight moves of a knight from some cell.



Example 1:




Input:

grid = [[0,11,16,5,20],[17,4,19,10,15],[12,1,8,21,6],[3,18,23,14,9],[24,13,2,7,22]]

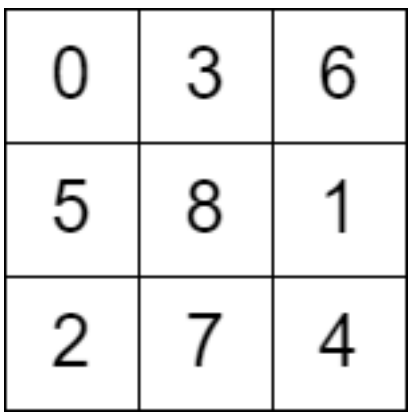
Output:

true

Explanation:

The above diagram represents the grid. It can be shown that it is a valid configuration.

Example 2:



Input:

grid = [[0,3,6],[5,8,1],[2,7,4]]

Output:

false

Explanation:

The above diagram represents the grid. The 8

th

move of the knight is not valid considering its position after the 7

th

move.

Constraints:

n == grid.length == grid[i].length

3 <= n <= 7

0 <= grid[row][col] < n * n

All integers in

grid

are

unique

.

## Code Snippets

### C++:

```
class Solution {
public:
bool checkValidGrid(vector<vector<int>>& grid) {

}
};
```

### Java:

```
class Solution {
public boolean checkValidGrid(int[][] grid) {

}
}
```

### Python3:

```
class Solution:
def checkValidGrid(self, grid: List[List[int]]) -> bool:
```

### Python:

```
class Solution(object):
def checkValidGrid(self, grid):
"""
:type grid: List[List[int]]
:rtype: bool
"""
```

### JavaScript:

```
/**
* @param {number[][]} grid
* @return {boolean}
*/
var checkValidGrid = function(grid) {

};
```

**TypeScript:**

```
function checkValidGrid(grid: number[][]): boolean {

};
```

**C#:**

```
public class Solution {
public bool CheckValidGrid(int[][] grid) {

}
}
```

**C:**

```
bool checkValidGrid(int** grid, int gridSize, int* gridColSize) {

}
```

**Go:**

```
func checkValidGrid(grid [][]int) bool {

}
```

**Kotlin:**

```
class Solution {
fun checkValidGrid(grid: Array<IntArray>): Boolean {

}
}
```

**Swift:**

```
class Solution {
func checkValidGrid(_ grid: [[Int]]) -> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn check_valid_grid(grid: Vec<Vec<i32>>) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[][]} grid
# @return {Boolean}
def check_valid_grid(grid)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $grid
* @return Boolean
*/
function checkValidGrid($grid) {

}
}
```

**Dart:**

```
class Solution {
bool checkValidGrid(List<List<int>> grid) {

}
}
```

**Scala:**

```
object Solution {
def checkValidGrid(grid: Array[Array[Int]]): Boolean = {

}
}
```

### Elixir:

```
defmodule Solution do
@spec check_valid_grid(grid :: [[integer]]) :: boolean
def check_valid_grid(grid) do

end
end
```

### Erlang:

```
-spec check_valid_grid(Grid :: [[integer()]]) -> boolean().
check_valid_grid(Grid) ->
.
```

### Racket:

```
(define/contract (check-valid-grid grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Check Knight Tour Configuration
* 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:
bool checkValidGrid(vector<vector<int>>& grid) {

}
};
```

### Java Solution:

```
/**
* Problem: Check Knight Tour Configuration
* 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 boolean checkValidGrid(int[][] grid) {

}
}
```

### Python3 Solution:

```
"""
Problem: Check Knight Tour Configuration
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 checkValidGrid(self, grid: List[List[int]]) -> bool:
# TODO: Implement optimized solution
pass
```

### Python Solution:

```
class Solution(object):
def checkValidGrid(self, grid):
"""
:type grid: List[List[int]]
:rtype: bool
```

```
"""
```

### JavaScript Solution:

```
/**
* Problem: Check Knight Tour Configuration
* 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[][]} grid
* @return {boolean}
*/
var checkValidGrid = function(grid) {

};
```

### TypeScript Solution:

```
/**
* Problem: Check Knight Tour Configuration
* 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 checkValidGrid(grid: number[][]): boolean {

};
```

### C# Solution:

```
/*
* Problem: Check Knight Tour Configuration
* 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 bool CheckValidGrid(int[][] grid) {

}
}
```

### C Solution:

```
/*
* Problem: Check Knight Tour Configuration
* 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
*/

bool checkValidGrid(int** grid, int gridSize, int* gridColSize) {

}
```

### Go Solution:

```
// Problem: Check Knight Tour Configuration
// 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 checkValidGrid(grid [][]int) bool {

}
```

### Kotlin Solution:

```
class Solution {
fun checkValidGrid(grid: Array<IntArray>): Boolean {

}
}
```

### Swift Solution:

```
class Solution {
func checkValidGrid(_ grid: [[Int]]) -> Bool {

}
}
```

### Rust Solution:

```
// Problem: Check Knight Tour Configuration
// 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 check_valid_grid(grid: Vec<Vec<i32>>) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Boolean}
def check_valid_grid(grid)
```

```
end
```

**PHP Solution:**

```
class Solution {

/**
* @param Integer[][] $grid
* @return Boolean
*/
function checkValidGrid($grid) {

}
}
```

**Dart Solution:**

```
class Solution {
bool checkValidGrid(List<List<int>> grid) {

}
}
```

**Scala Solution:**

```
object Solution {
def checkValidGrid(grid: Array[Array[Int]]): Boolean = {

}
}
```

**Elixir Solution:**

```
defmodule Solution do
@spec check_valid_grid(grid :: [[integer]]) :: boolean
def check_valid_grid(grid) do

end
end
```

### Erlang Solution:

```
-spec check_valid_grid(Grid :: [[integer()]]) -> boolean().
check_valid_grid(Grid) ->
.
```

### Racket Solution:

```
(define/contract (check-valid-grid grid)
(-> (listof (listof exact-integer?)) boolean?)
)
```