# Problem 329: Longest Increasing Path in a Matrix

## Problem Information

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

## Problem Description

Given an

m x n

integers

matrix

, return

the length of the longest increasing path in

matrix

.

From each cell, you can either move in four directions: left, right, up, or down. You

may not

move

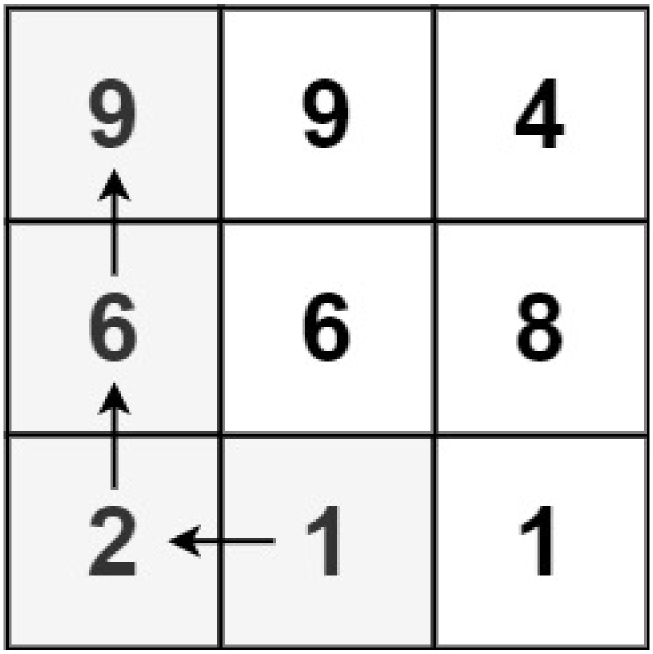
diagonally

or move

outside the boundary

(i.e., wrap-around is not allowed).

Example 1:



| | | |
|---|---|---|
| 9 | 9 | 4 |
| 6 | 6 | 8 |
| 2 | 1 | 1 |

Input:

matrix = [[9,9,4],[6,6,8],[2,1,1]]

Output:

4

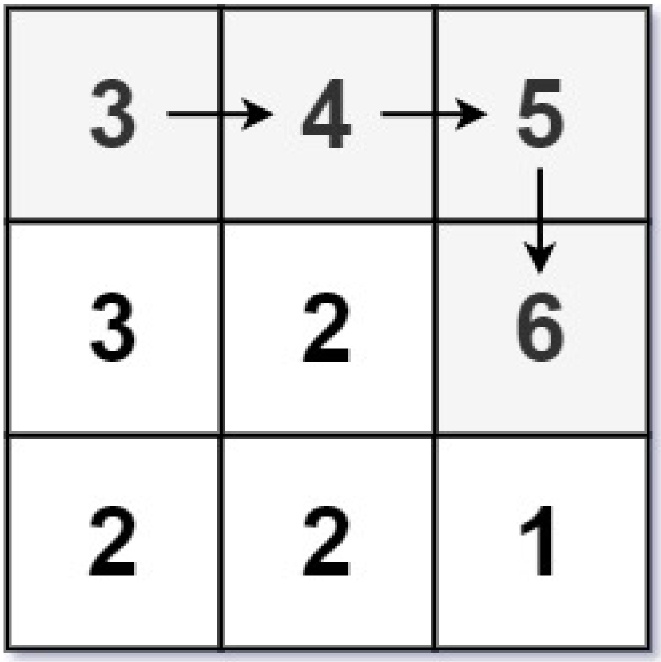
Explanation:

The longest increasing path is

[1, 2, 6, 9]

.

Example 2:



Input:

matrix = [[3,4,5],[3,2,6],[2,2,1]]

Output:

4

Explanation:

The longest increasing path is

[3, 4, 5, 6]

. Moving diagonally is not allowed.

Example 3:

Input:

matrix = [[1]]

Output:

1

Constraints:

m == matrix.length

n == matrix[i].length

1 <= m, n <= 200

0 <= matrix[i][j] <= 2

31

- 1

## Code Snippets

**C++:**

```
class Solution {
public:
int longestIncreasingPath(vector<vector<int>>& matrix) {

}
};
```

**Java:**

```
class Solution {
public int longestIncreasingPath(int[][] matrix) {

}
}
```

**Python3:**

```
class Solution:
def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
```

## Python:

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

## JavaScript:

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

};
```

## TypeScript:

```
function longestIncreasingPath(matrix: number[][]): number {

};
```

## C#:

```
public class Solution {
public int LongestIncreasingPath(int[][] matrix) {

}
}
```

## C:

```
int longestIncreasingPath(int** matrix, int matrixSize, int* matrixColSize) {

}
```

## Go:

```
func longestIncreasingPath(matrix [][]int) int {

}
```

### Kotlin:

```
class Solution {
fun longestIncreasingPath(matrix: Array<IntArray>): Int {

}
}
```

### Swift:

```
class Solution {
func longestIncreasingPath(_ matrix: [[Int]]) -> Int {

}
}
```

### Rust:

```
impl Solution {
pub fn longest_increasing_path(matrix: Vec<Vec<i32>>) -> i32 {

}
}
```

### Ruby:

```
# @param {Integer[][]} matrix
# @return {Integer}
def longest_increasing_path(matrix)

end
```

### PHP:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer
*/
function longestIncreasingPath($matrix) {

}
}
```

**Dart:**

```
class Solution {
int longestIncreasingPath(List<List<int>> matrix) {

}
}
```

**Scala:**

```
object Solution {
def longestIncreasingPath(matrix: Array[Array[Int]]): Int = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec longest_increasing_path(matrix :: [[integer]]) :: integer
def longest_increasing_path(matrix) do

end
end
```

**Erlang:**

```
-spec longest_increasing_path(Matrix :: [[integer()]]) -> integer().
longest_increasing_path(Matrix) ->
.
```

**Racket:**

```
(define/contract (longest-increasing-path matrix)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Longest Increasing Path in a Matrix
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int longestIncreasingPath(vector<vector<int>>& matrix) {

}
};
```

### Java Solution:

```
/**
* Problem: Longest Increasing Path in a Matrix
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int longestIncreasingPath(int[][] matrix) {

}
}
```

### Python3 Solution:

```
"""
Problem: Longest Increasing Path in a Matrix
Difficulty: Hard
Tags: array, graph, dp, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Longest Increasing Path in a Matrix
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[][]} matrix
* @return {number}
*/
var longestIncreasingPath = function(matrix) {
};
```

### TypeScript Solution:

```
/**
* Problem: Longest Increasing Path in a Matrix
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function longestIncreasingPath(matrix: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Longest Increasing Path in a Matrix
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int LongestIncreasingPath(int[][] matrix) {

}
}
```

### C Solution:

```
/*
* Problem: Longest Increasing Path in a Matrix
* Difficulty: Hard
* Tags: array, graph, dp, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int longestIncreasingPath(int** matrix, int matrixSize, int* matrixColSize) {

}
```

### Go Solution:

```
// Problem: Longest Increasing Path in a Matrix
// Difficulty: Hard
// Tags: array, graph, dp, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func longestIncreasingPath(matrix [][]int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun longestIncreasingPath(matrix: Array<IntArray>): Int {

}
}
```

### Swift Solution:

```
class Solution {
func longestIncreasingPath(_ matrix: [[Int]]) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Longest Increasing Path in a Matrix
// Difficulty: Hard
// Tags: array, graph, dp, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn longest_increasing_path(matrix: Vec<Vec<i32>>) -> i32 {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} matrix
# @return {Integer}
def longest_increasing_path(matrix)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @return Integer
*/
function longestIncreasingPath($matrix) {

}
}
```

### Dart Solution:

```
class Solution {
int longestIncreasingPath(List<List<int>> matrix) {
}
}
```

### Scala Solution:

```
object Solution {
def longestIncreasingPath(matrix: Array[Array[Int]]): Int = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec longest_increasing_path(matrix :: [[integer]]) :: integer
def longest_increasing_path(matrix) do

end
end
```

### Erlang Solution:

```
-spec longest_increasing_path(Matrix :: [[integer()]]) -> integer().
longest_increasing_path(Matrix) ->
.
```

### Racket Solution:

```
(define/contract (longest-increasing-path matrix)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```