Problem 3417: Zigzag Grid Traversal With Skip

Problem Information

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

Problem Description

You are given an

m x n

2D array

grid

of

positive

integers.

Your task is to traverse

grid

in a

zigzag

pattern while skipping every

alternate

cell.

Zigzag pattern traversal is defined as following the below actions:

Start at the top-left cell

(0, 0)

.

Move

right

within a row until the end of the row is reached.

Drop down to the next row, then traverse

left

until the beginning of the row is reached.

Continue

alternating

between right and left traversal until every row has been traversed.

Note

that you

must skip

every

alternate

cell during the traversal.

Return an array of integers

result

containing,

in order

, the value of the cells visited during the zigzag traversal with skips.

Example 1:

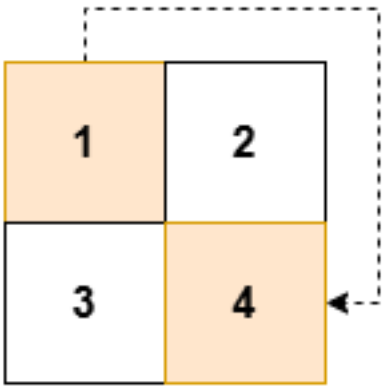
Input:

grid = [[1,2],[3,4]]

Output:

[1,4]

Explanation:



Example 2:

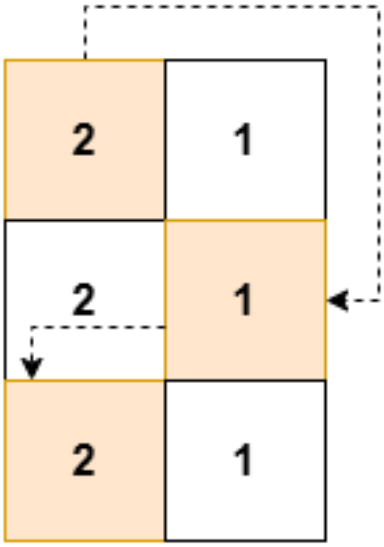
Input:

grid = [[2,1],[2,1],[2,1]]

Output:

[2,1,2]

Explanation:



Example 3:

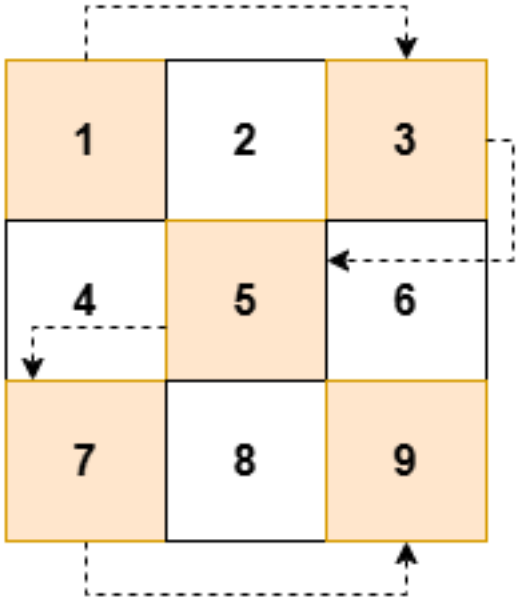
Input:

grid = [[1,2,3],[4,5,6],[7,8,9]]

Output:

[1,3,5,7,9]

Explanation:



Constraints:

2 <= n == grid.length <= 50

2 <= m == grid[i].length <= 50

1 <= grid[i][j] <= 2500

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> zigzagTraversal(int[][] grid) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function zigzagTraversal(grid: number[][]): number[] {

};
```

C#:

```
public class Solution {
public IList<int> ZigzagTraversal(int[][] grid) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* zigzagTraversal(int** grid, int gridSize, int* gridColSize, int*
returnSize) {

}
```

Go:

```
func zigzagTraversal(grid [][]int) []int {

}
```

Kotlin:

```
class Solution {
fun zigzagTraversal(grid: Array<IntArray>): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
List<int> zigzagTraversal(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def zigzagTraversal(grid: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (zigzag-traversal grid)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Zigzag Grid Traversal With Skip
* Difficulty: Easy
* Tags: array
*
* 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:
vector<int> zigzagTraversal(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Zigzag Grid Traversal With Skip
* Difficulty: Easy
* Tags: array
*
* 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 List<Integer> zigzagTraversal(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Zigzag Grid Traversal With Skip
Difficulty: Easy
Tags: array

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 zigzagTraversal(self, grid: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Zigzag Grid Traversal With Skip
* Difficulty: Easy
* Tags: array
*
```

```
* 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 {number[]}
*/
var zigzagTraversal = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Zigzag Grid Traversal With Skip
* Difficulty: Easy
* Tags: array
*
* 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 zigzagTraversal(grid: number[][]): number[] {

};
```

C# Solution:

```
/*
* Problem: Zigzag Grid Traversal With Skip
* Difficulty: Easy
* Tags: array
*
* 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 IList<int> ZigzagTraversal(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Zigzag Grid Traversal With Skip
* Difficulty: Easy
* Tags: array
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* zigzagTraversal(int** grid, int gridSize, int* gridColSize, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Zigzag Grid Traversal With Skip
// Difficulty: Easy
// Tags: array
//
// 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 zigzagTraversal(grid [][]int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun zigzagTraversal(grid: Array<IntArray>): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Zigzag Grid Traversal With Skip
// Difficulty: Easy
// Tags: array
//
// 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 zigzag_traversal(grid: Vec<Vec<i32>>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param Integer[][] $grid
* @return Integer[]
*/
function zigzagTraversal($grid) {

}
}
```

Dart Solution:

```
class Solution {
List<int> zigzagTraversal(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def zigzagTraversal(grid: Array[Array[Int]]): List[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (zigzag-traversal grid)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```