Problem 1706: Where Will the Ball Fall

Problem Information

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

Problem Description

You have a 2-D

grid

of size

m x n

representing a box, and you have

n

balls. The box is open on the top and bottom sides.

Each cell in the box has a diagonal board spanning two corners of the cell that can redirect a ball to the right or to the left.

A board that redirects the ball to the right spans the top-left corner to the bottom-right corner and is represented in the grid as

1

.

A board that redirects the ball to the left spans the top-right corner to the bottom-left corner and is represented in the grid as

-1

.

We drop one ball at the top of each column of the box. Each ball can get stuck in the box or fall out of the bottom. A ball gets stuck if it hits a "V" shaped pattern between two boards or if a board redirects the ball into either wall of the box.

Return

an array

answer

of size

n

where

answer[i]

is the column that the ball falls out of at the bottom after dropping the ball from the

i

th

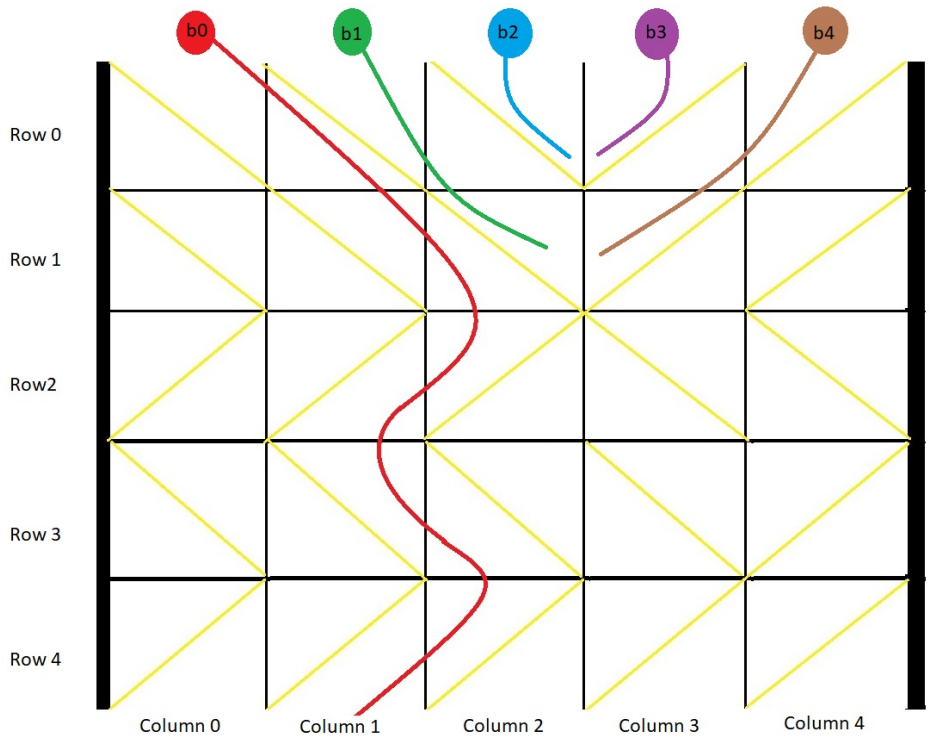
column at the top, or

-1

if the ball gets stuck in the box

.

Example 1:



Input:

grid = [[1,1,1,-1,-1],[1,1,1,-1,-1],[-1,-1,-1,1,1],[1,1,1,1,-1],[-1,-1,-1,-1,-1]]

Output:

[1,-1,-1,-1,-1]

Explanation:

This example is shown in the photo. Ball b0 is dropped at column 0 and falls out of the box at column 1. Ball b1 is dropped at column 1 and will get stuck in the box between column 2 and 3 and row 1. Ball b2 is dropped at column 2 and will get stuck on the box between column 2 and 3 and row 0. Ball b3 is dropped at column 3 and will get stuck on the box between column 2 and 3 and row 0. Ball b4 is dropped at column 4 and will get stuck on the box between column 2 and 3 and row 1.

Example 2:

Input:

grid = [[-1]]

Output:

[-1]

Explanation:

The ball gets stuck against the left wall.

Example 3:

Input:

grid = [[1,1,1,1,1,1],[-1,-1,-1,-1,-1,-1],[1,1,1,1,1,1],[-1,-1,-1,-1,-1,-1]]

Output:

[0,1,2,3,4,-1]

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 100

grid[i][j]

is

1

or

-1

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] findBall(int[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int[] FindBall(int[][] grid) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-ball grid)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Where Will the Ball Fall
* Difficulty: Medium
* 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> findBall(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Where Will the Ball Fall
* Difficulty: Medium
* 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 int[] findBall(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Where Will the Ball Fall
Difficulty: Medium
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 findBall(self, grid: List[List[int]]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Where Will the Ball Fall
* Difficulty: Medium
* 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 findBall = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Where Will the Ball Fall
* Difficulty: Medium
* 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 findBall(grid: number[][]): number[] {
```

```
};
```

C# Solution:

```
/*
* Problem: Where Will the Ball Fall
* Difficulty: Medium
* 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 int[] FindBall(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Where Will the Ball Fall
* Difficulty: Medium
* 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* findBall(int** grid, int gridSize, int* gridColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Where Will the Ball Fall
// Difficulty: Medium
// 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 findBall(grid [][]int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Where Will the Ball Fall
// Difficulty: Medium
// 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 find_ball(grid: Vec<Vec<i32>>) -> Vec<i32> {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-ball grid)
(-> (listof (listof exact-integer?)) (listof exact-integer?))
)
```