Problem 773: Sliding Puzzle

Problem Information

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

Problem Description

On an

2 x 3

board, there are five tiles labeled from

1

to

5

, and an empty square represented by

0

. A

move

consists of choosing

0

and a 4-directionally adjacent number and swapping it.

The state of the board is solved if and only if the board is

[[1,2,3],[4,5,0]]

.

Given the puzzle board

board

, return

the least number of moves required so that the state of the board is solved

. If it is impossible for the state of the board to be solved, return

-1

.

Example 1:

Input:

board = [[1,2,3],[4,0,5]]

Output:

1

Explanation:

Swap the 0 and the 5 in one move.

Example 2:

Input:

board = [[1,2,3],[5,4,0]]

Output:

-1

Explanation:

No number of moves will make the board solved.

Example 3:

Input:

board = [[4,1,2],[5,0,3]]

Output:

5

Explanation:

5 is the smallest number of moves that solves the board. An example path: After move 0: [[4,1,2],[5,0,3]] After move 1: [[4,1,2],[0,5,3]] After move 2: [[0,1,2],[4,5,3]] After move 3: [[1,0,2],[4,5,3]] After move 4: [[1,2,0],[4,5,3]] After move 5: [[1,2,3],[4,5,0]]

Constraints:

board.length == 2

board[i].length == 3

0 <= board[i][j] <= 5

Each value

board[i][j]

is

unique

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[][]} board
* @return {number}
*/
```

```
var slidingPuzzle = function(board) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sliding-puzzle board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sliding Puzzle
* Difficulty: Hard
* Tags: array, dp, 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 slidingPuzzle(vector<vector<int>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Sliding Puzzle
* Difficulty: Hard
* Tags: array, dp, 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 slidingPuzzle(int[][] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Sliding Puzzle
Difficulty: Hard
Tags: array, dp, 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 slidingPuzzle(self, board: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sliding Puzzle
* Difficulty: Hard
* Tags: array, dp, 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[][]} board
* @return {number}
*/
var slidingPuzzle = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Sliding Puzzle
* Difficulty: Hard
* Tags: array, dp, 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 slidingPuzzle(board: number[][]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Sliding Puzzle
* Difficulty: Hard
* Tags: array, dp, 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 SlidingPuzzle(int[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Sliding Puzzle
* Difficulty: Hard
* Tags: array, dp, 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 slidingPuzzle(int** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Sliding Puzzle
// Difficulty: Hard
```

```
// Tags: array, dp, 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 slidingPuzzle(board [][]int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sliding Puzzle
// Difficulty: Hard
// Tags: array, dp, 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 sliding_puzzle(board: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sliding-puzzle board)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```