

# Problem 773: Sliding Puzzle

## Problem Information

**Difficulty:** Hard

**Acceptance Rate:** 73.43%

**Paid Only:** No

**Tags:** Array, Dynamic Programming, Backtracking, Breadth-First Search, Memoization, Matrix

## 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:
```