Problem 1301: Number of Paths with Max Score

Problem Information

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

Problem Description

You are given a square

board

of characters. You can move on the board starting at the bottom right square marked with the character

'S'

.

You need to reach the top left square marked with the character

'E'

. The rest of the squares are labeled either with a numeric character

1, 2, ..., 9

or with an obstacle

'X'

. In one move you can go up, left or up-left (diagonally) only if there is no obstacle there.

Return a list of two integers: the first integer is the maximum sum of numeric characters you can collect, and the second is the number of such paths that you can take to get that maximum sum,

taken modulo

10^9 + 7

.

In case there is no path, return

[0, 0]

.

Example 1:

Input:

board = ["E23","2X2","12S"]

Output:

[7,1]

Example 2:

Input:

board = ["E12","1X1","21S"]

Output:

[4,2]

Example 3:

Input:

board = ["E11","XXX","11S"]

Output:

[0,0]

Constraints:

2 <= board.length == board[i].length <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int[] pathsWithMaxScore(List<String> board) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def pathsWithMaxScore(self, board):
"""
:type board: List[str]
```

```
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {string[]} board
* @return {number[]}
*/
var pathsWithMaxScore = function(board) {

};
```

TypeScript:

```
function pathsWithMaxScore(board: string[]): number[] {

};
```

C#:

```
public class Solution {
public int[] PathsWithMaxScore(IList<string> board) {

}
}
```

C:

```


/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* pathsWithMaxScore(char ** board, int boardSize, int* returnSize){

}
```

Go:

```
func pathsWithMaxScore(board []string) []int {
```

```
}
```

Kotlin:

```
class Solution {
fun pathsWithMaxScore(board: List<String>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String[] $board
* @return Integer[]
*/
```

```
function pathsWithMaxScore($board) {

}
}
```

Scala:

```
object Solution {
def pathsWithMaxScore(board: List[String]): Array[Int] = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Paths with Max Score
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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:
vector<int> pathsWithMaxScore(vector<string>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Paths with Max Score
* Difficulty: Hard
* Tags: array, tree, dp
```

```
*
* 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[] pathsWithMaxScore(List<String> board) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Paths with Max Score
Difficulty: Hard
Tags: array, tree, dp

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 pathsWithMaxScore(self, board: List[str]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Paths with Max Score
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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 {string[]} board
* @return {number[]}
*/
var pathsWithMaxScore = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Paths with Max Score
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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 pathsWithMaxScore(board: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Number of Paths with Max Score
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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[] PathsWithMaxScore(IList<string> board) {

}
}
```

C Solution:

```
/*
* Problem: Number of Paths with Max Score
* Difficulty: Hard
* Tags: array, tree, dp
*
* 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
*/




/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* pathsWithMaxScore(char ** board, int boardSize, int* returnSize){

}
```

Go Solution:

```
// Problem: Number of Paths with Max Score
// Difficulty: Hard
// Tags: array, tree, dp
//
// 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 pathsWithMaxScore(board []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun pathsWithMaxScore(board: List<String>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Paths with Max Score
// Difficulty: Hard
// Tags: array, tree, dp
//
// 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 paths_with_max_score(board: Vec<String>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

```
object Solution {
def pathsWithMaxScore(board: List[String]): Array[Int] = {

}
}
```