Problem 2814: Minimum Time Takes to Reach Destination Without Drowning

Problem Information

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

Problem Description

You are given an

n * m

0-indexed

grid of string

land

. Right now, you are standing at the cell that contains

"S"

, and you want to get to the cell containing

"D"

. There are three other types of cells in this land:

"."

: These cells are empty.

"X"

: These cells are stone.

"*"

: These cells are flooded.

At each second, you can move to a cell that shares a side with your current cell (if it exists). Also, at each second, every

empty cell

that shares a side with a flooded cell becomes flooded as well.

There are two problems ahead of your journey:

You can't step on stone cells.

You can't step on flooded cells since you will drown (also, you can't step on a cell that will be flooded at the same time as you step on it).

Return

the

minimum

time it takes you to reach the destination in seconds, or

-1

if it is impossible.

Note

that the destination will never be flooded.

Example 1:

Input:

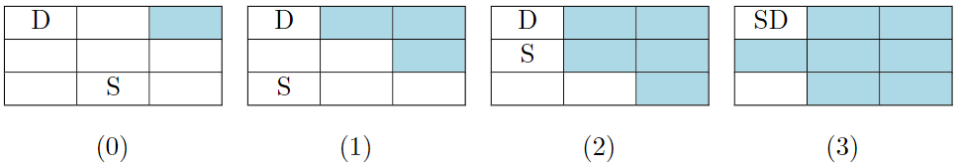
land = [["D",".","*"],[".",".","."],[".","S","."]]

Output:

3

Explanation:

The picture below shows the simulation of the land second by second. The blue cells are flooded, and the gray cells are stone. Picture (0) shows the initial state and picture (3) shows the final state when we reach destination. As you see, it takes us 3 second to reach destination and the answer would be 3. It can be shown that 3 is the minimum time needed to reach from S to D.



Example 2:

Input:

land = [["D","X","*"],[".",".","."],[".",".","S"]]

Output:

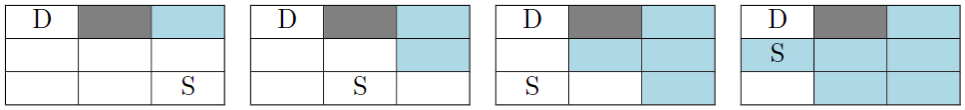
-1

Explanation:

The picture below shows the simulation of the land second by second. The blue cells are flooded, and the gray cells are stone. Picture (0) shows the initial state. As you see, no matter which paths we choose, we will drown at the 3

rd

second. Also the minimum path takes us 4 seconds to reach from S to D. So the answer would be -1.



Example 3:

Input:

land = [["D",".",".",".",".","*","."],[".","X",".","X",".",".","."],[".",".",".",".",".",".","S","."]]

Output:

6

Explanation:

It can be shown that we can reach destination in 6 seconds. Also it can be shown that 6 is the minimum seconds one need to reach from S to D.

Constraints:

2 <= n, m <= 100

land

consists only of

"S"

,

"D"

,

"."

,

"*"

and

"X"

.

Exactly

one of the cells is equal to

"S"

.

Exactly

one of the cells is equal to

"D"

.

Code Snippets

C++:

```
class Solution {
public:
int minimumSeconds(vector<vector<string>>& land) {

}
};
```

Java:

```
class Solution {
public int minimumSeconds(List<List<String>> land) {
```

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumSeconds(land: string[][]): number {

};
```

C#:

```
public class Solution {
public int MinimumSeconds(IList<IList<string>> land) {

}
}
```

C:

```
int minimumSeconds(char*** land, int landSize, int* landColSize) {

}
```

Go:

```
func minimumSeconds(land [][]string) int {

}
```

Kotlin:

```
class Solution {
fun minimumSeconds(land: List<List<String>>): Int {

}
}
```

Swift:

```
class Solution {
func minimumSeconds(_ land: [[String]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_seconds(land: Vec<Vec<String>>) -> i32 {

}
}
```

Ruby:

```
# @param {String[][]} land
# @return {Integer}
def minimum_seconds(land)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $land
* @return Integer
*/
function minimumSeconds($land) {

}
}
```

Dart:

```
class Solution {
int minimumSeconds(List<List<String>> land) {

}
}
```

Scala:

```
object Solution {
def minimumSeconds(land: List[List[String]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_seconds(land :: [[String.t]]) :: integer
def minimum_seconds(land) do

end
end
```

Erlang:

```
-spec minimum_seconds(Land :: [[unicode:unicode_binary()]]) -> integer().
minimum_seconds(Land) ->
.
```

Racket:

```
(define/contract (minimum-seconds land)
(-> (listof (listof string?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Time Takes to Reach Destination Without Drowning
* Difficulty: Hard
* Tags: array, string, search
*
* 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 minimumSeconds(vector<vector<string>>& land) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Time Takes to Reach Destination Without Drowning
* Difficulty: Hard
* Tags: array, string, search
*
* 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 minimumSeconds(List<List<String>> land) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Time Takes to Reach Destination Without Drowning
Difficulty: Hard
Tags: array, string, search

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Time Takes to Reach Destination Without Drowning
* Difficulty: Hard
* Tags: array, string, search
*
* 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 {string[][]} land
* @return {number}
*/
var minimumSeconds = function(land) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Time Takes to Reach Destination Without Drowning
* Difficulty: Hard
* Tags: array, string, search
*
* 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 minimumSeconds(land: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Time Takes to Reach Destination Without Drowning
* Difficulty: Hard
* Tags: array, string, search
*
* 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 MinimumSeconds(IList<IList<string>> land) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Time Takes to Reach Destination Without Drowning
* Difficulty: Hard
* Tags: array, string, search
*
* 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
*/

int minimumSeconds(char*** land, int landSize, int* landColSize) {

}
```

Go Solution:

```
// Problem: Minimum Time Takes to Reach Destination Without Drowning
// Difficulty: Hard
// Tags: array, string, search
//
// 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 minimumSeconds(land [][]string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumSeconds(land: List<List<String>>): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumSeconds(_ land: [[String]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Time Takes to Reach Destination Without Drowning
// Difficulty: Hard
// Tags: array, string, search
//
// 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 minimum_seconds(land: Vec<Vec<String>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[][]} land
# @return {Integer}
def minimum_seconds(land)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $land
* @return Integer
*/
function minimumSeconds($land) {

}
}
```

Dart Solution:

```
class Solution {
int minimumSeconds(List<List<String>> land) {

}
}
```

Scala Solution:

```
object Solution {
def minimumSeconds(land: List[List[String]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_seconds(land :: [[String.t]]) :: integer
def minimum_seconds(land) do

end
end
```

Erlang Solution:

```
-spec minimum_seconds(Land :: [[unicode:unicode_binary()]]) -> integer().
minimum_seconds(Land) ->
.
```

Racket Solution:

```
(define/contract (minimum-seconds land)
(-> (listof (listof string?)) exact-integer?)
)
```