Problem 1730: Shortest Path to Get Food

Problem Information

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

Problem Description

You are starving and you want to eat food as quickly as possible. You want to find the shortest path to arrive at any food cell.

You are given an

m x n

character matrix,

grid

, of these different types of cells:

'*'

is your location. There is

exactly one

'*'

cell.

'#'

is a food cell. There may be

multiple

food cells.

'O'

is free space, and you can travel through these cells.

'X'

is an obstacle, and you cannot travel through these cells.

You can travel to any adjacent cell north, east, south, or west of your current location if there is not an obstacle.

Return

the

length

of the shortest path for you to reach

any

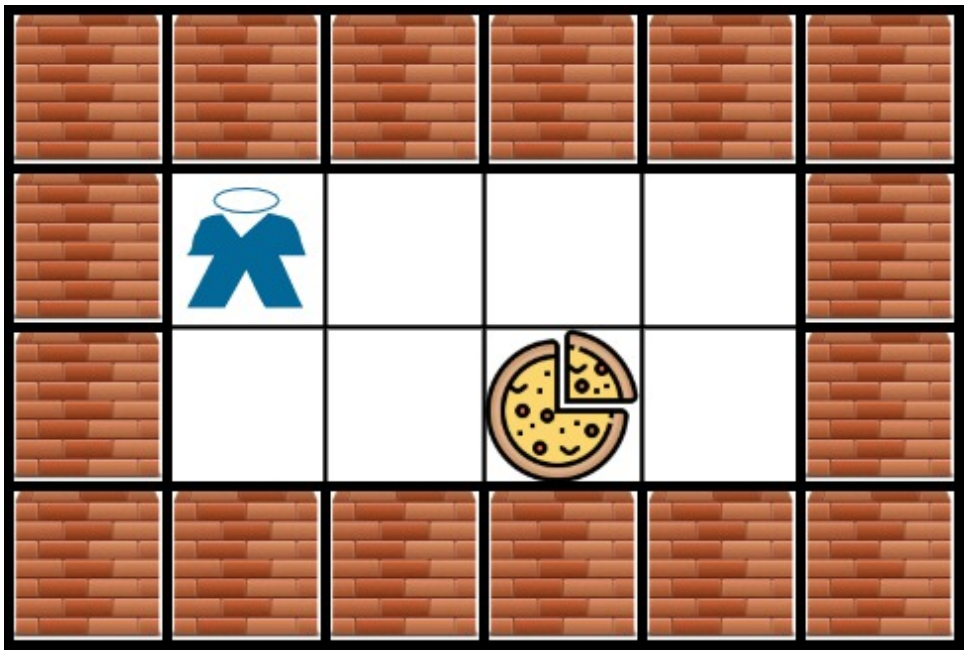
food cell

. If there is no path for you to reach food, return

-1

.

Example 1:



Input:

grid = [["X","X","X","X","X","X"],["X","*","O","O","O","X"],["X","O","O","#","O","X"],["X","X","X","X","X","X"]]

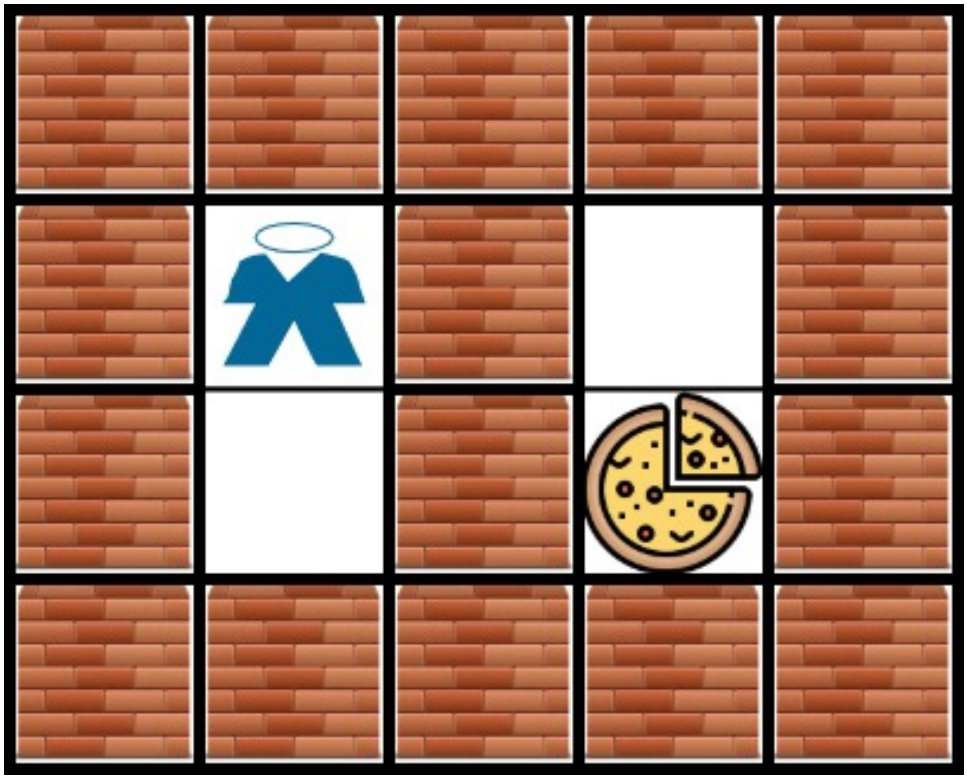
Output:

3

Explanation:

It takes 3 steps to reach the food.

Example 2:



Input:

grid = [["X","X","X","X","X"],["X","*","X","O","X"],["X","O","X","#","X"],["X","X","X","X","X"]]

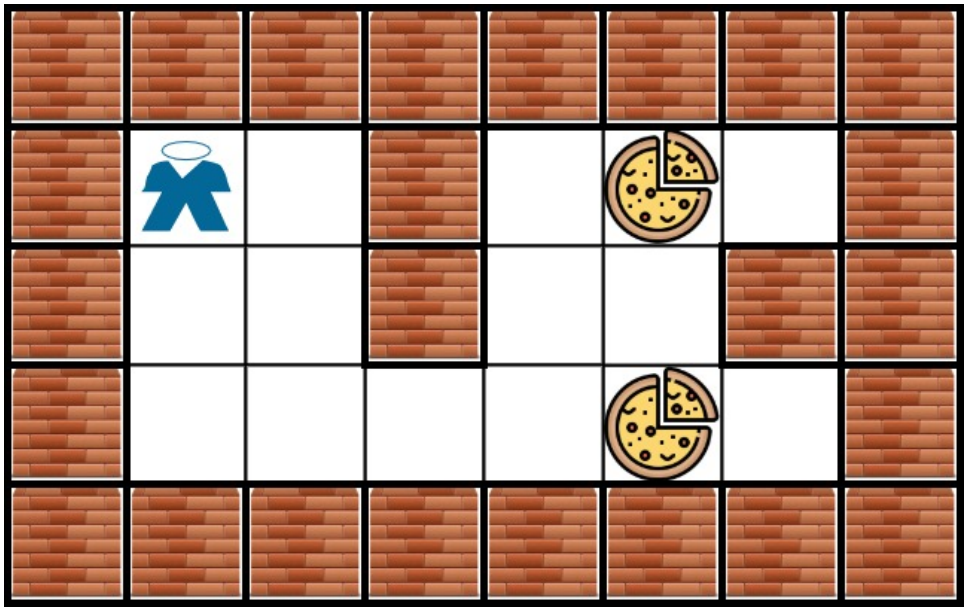
Output:

-1

Explanation:

It is not possible to reach the food.

Example 3:



Input:

grid = [["X","X","X","X","X","X","X","X"],["X","*","O","X","O","#","O","X"],["X","O","O","X","O","O","X","X"],["X","O","O","O","O","#","O","X"],["X","X","X","X","X","X","X","X"]]

Output:

6

Explanation:

There can be multiple food cells. It only takes 6 steps to reach the bottom food.

Example 4:

Input:

grid = [["X","X","X","X","X","X","X","X"],["X","*","O","X","O","#","O","X"],["X","O","O","X","O","O","X","X"],["X","O","O","O","O","#","O","X"],["O","O","O","O","O","O","O","O"]]

Output:

5

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 200

grid[row][col]

is

'*'

,

'X'

,

'O'

, or

'#'

.

The

grid

contains

exactly one

'*'

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int getFood(char[][] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function getFood(grid: string[][]): number {

};
```

C#:

```
public class Solution {
public int GetFood(char[][] grid) {

}
}
```

C:

```
int getFood(char** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func getFood(grid [][]byte) int {

}
```

Kotlin:

```
class Solution {
fun getFood(grid: Array<CharArray>): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int getFood(List<List<String>> grid) {

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (get-food grid)
(-> (listof (listof char?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path to Get Food
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

```
}
};
```

Java Solution:

```
/**
* Problem: Shortest Path to Get Food
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int getFood(char[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Path to Get Food
Difficulty: Medium
Tags: array, tree, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def getFood(self, grid: List[List[str]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Path to Get Food
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path to Get Food
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function getFood(grid: string[][]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Path to Get Food
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int GetFood(char[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Path to Get Food
* Difficulty: Medium
* Tags: array, tree, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int getFood(char** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```
// Problem: Shortest Path to Get Food
// Difficulty: Medium
// Tags: array, tree, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(h) for recursion stack where h is height

func getFood(grid [][]byte) int {

}
```

Kotlin Solution:

```
class Solution {
fun getFood(grid: Array<CharArray>): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Shortest Path to Get Food
// Difficulty: Medium
// Tags: array, tree, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn get_food(grid: Vec<Vec<char>>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int getFood(List<List<String>> grid) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (get-food grid)
(-> (listof (listof char?)) exact-integer?)
)
```