Problem 864: Shortest Path to Get All Keys

Problem Information

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

Problem Description

You are given an

m x n

grid

grid

where:

'.'

is an empty cell.

'#'

is a wall.

'@'

is the starting point.

Lowercase letters represent keys.

Uppercase letters represent locks.

You start at the starting point and one move consists of walking one space in one of the four cardinal directions. You cannot walk outside the grid, or walk into a wall.

If you walk over a key, you can pick it up and you cannot walk over a lock unless you have its corresponding key.

For some

1 <= k <= 6

, there is exactly one lowercase and one uppercase letter of the first

k

letters of the English alphabet in the grid. This means that there is exactly one key for each lock, and one lock for each key; and also that the letters used to represent the keys and locks were chosen in the same order as the English alphabet.

Return

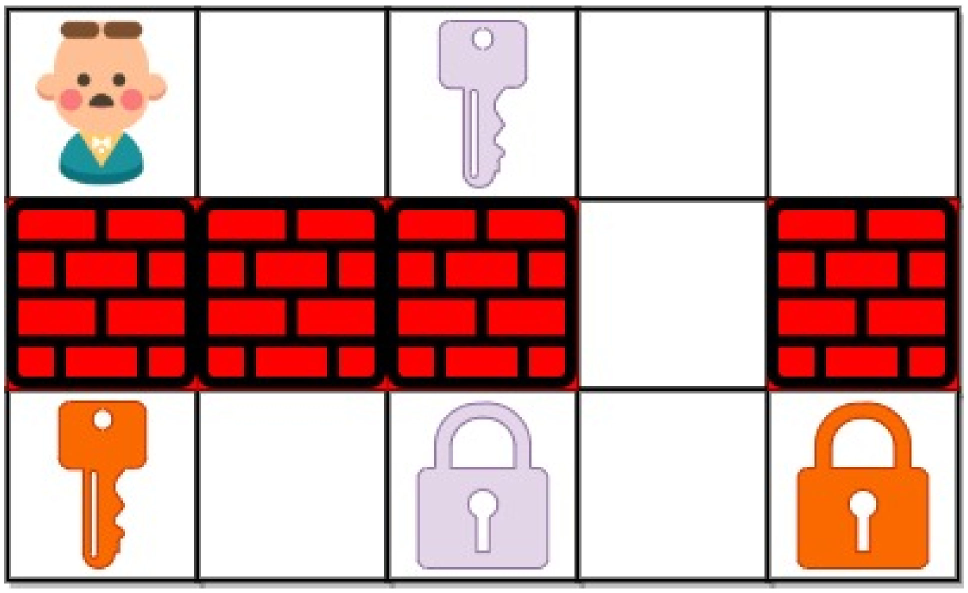
the lowest number of moves to acquire all keys

. If it is impossible, return

-1

.

Example 1:



Input:

grid = ["@.a..","###.#","b.A.B"]

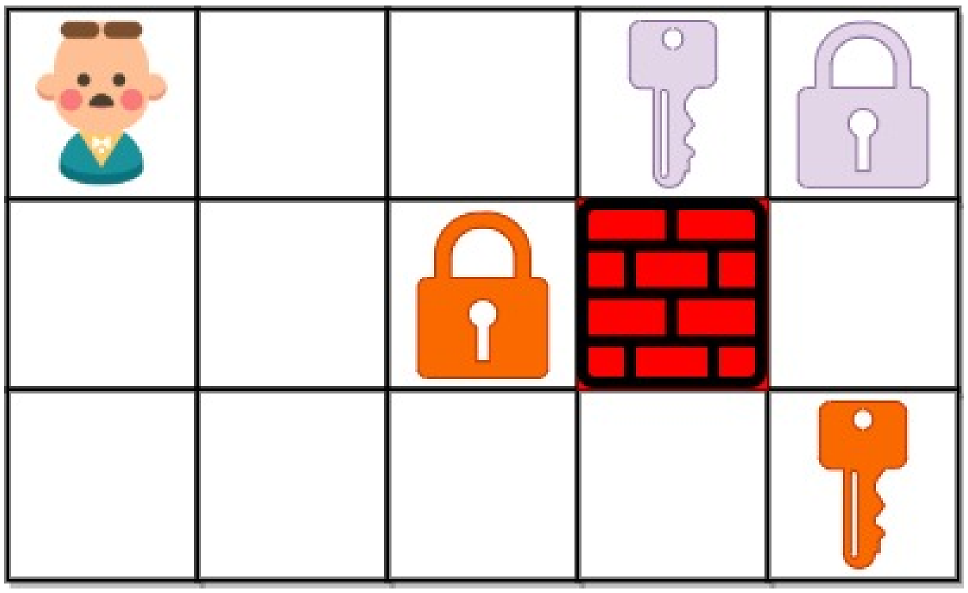
Output:

8

Explanation:

Note that the goal is to obtain all the keys not to open all the locks.

Example 2:



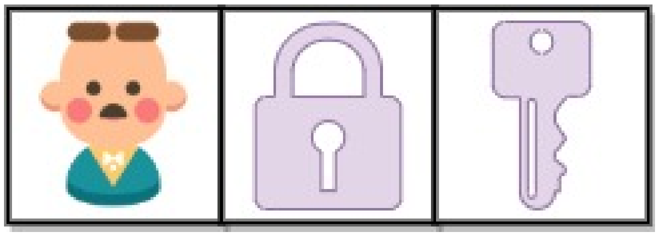
Input:

grid = ["@..aA","..B#.","....b"]

Output:

6

Example 3:



Input:

grid = ["@Aa"]

Output:

-1

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 30

grid[i][j]

is either an English letter,

'.'

,

'#'

, or

'@'

.

There is exactly one

'@'

in the grid.

The number of keys in the grid is in the range

[1, 6]

.

Each key in the grid is

unique

.

Each key in the grid has a matching lock.

Code Snippets

C++:

```
class Solution {
public:
int shortestPathAllKeys(vector<string>& grid) {

}
};
```

Java:

```
class Solution {
public int shortestPathAllKeys(String[] grid) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string[]} grid
* @return {number}
*/
```

```
var shortestPathAllKeys = function(grid) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int ShortestPathAllKeys(string[] grid) {

}
}
```

C:

```
int shortestPathAllKeys(char** grid, int gridSize) {

}
```

Go:

```
func shortestPathAllKeys(grid []string) int {

}
```

Kotlin:

```
class Solution {
fun shortestPathAllKeys(grid: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func shortestPathAllKeys(_ grid: [String]) -> Int {
}
}
```

Rust:

```
impl Solution {
pub fn shortest_path_all_keys(grid: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} grid
# @return {Integer}
def shortest_path_all_keys(grid)

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

```
object Solution {
def shortestPathAllKeys(grid: Array[String]): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec shortest_path_all_keys(Grid :: [unicode:unicode_binary()]) ->
integer().
shortest_path_all_keys(Grid) ->
.
```

Racket:

```
(define/contract (shortest-path-all-keys grid)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Shortest Path to Get All Keys
* Difficulty: Hard
* Tags: array, 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 shortestPathAllKeys(vector<string>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Shortest Path to Get All Keys
* Difficulty: Hard
* Tags: array, 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 shortestPathAllKeys(String[] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Shortest Path to Get All Keys
Difficulty: Hard
Tags: array, 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 shortestPathAllKeys(self, grid: List[str]) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Shortest Path to Get All Keys
* Difficulty: Hard
* Tags: array, 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[]} grid
* @return {number}
*/
var shortestPathAllKeys = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Shortest Path to Get All Keys
* Difficulty: Hard
* Tags: array, 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 shortestPathAllKeys(grid: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Shortest Path to Get All Keys
* Difficulty: Hard
* Tags: array, 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 ShortestPathAllKeys(string[] grid) {

}
}
```

C Solution:

```
/*
* Problem: Shortest Path to Get All Keys
* Difficulty: Hard
* Tags: array, 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 shortestPathAllKeys(char** grid, int gridSize) {

}
```

Go Solution:

```
// Problem: Shortest Path to Get All Keys
// Difficulty: Hard
// Tags: array, 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 shortestPathAllKeys(grid []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun shortestPathAllKeys(grid: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func shortestPathAllKeys(_ grid: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Shortest Path to Get All Keys
// Difficulty: Hard
// Tags: array, 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 shortest_path_all_keys(grid: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} grid
# @return {Integer}
def shortest_path_all_keys(grid)

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def shortestPathAllKeys(grid: Array[String]): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec shortest_path_all_keys(Grid :: [unicode:unicode_binary()]) ->
integer().
shortest_path_all_keys(Grid) ->
.
```

Racket Solution:

```
(define/contract (shortest-path-all-keys grid)
(-> (listof string?) exact-integer?)
)
```