Problem 1444: Number of Ways of Cutting a Pizza

Problem Information

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

Problem Description

Given a rectangular pizza represented as a

rows x cols

matrix containing the following characters:

'A'

(an apple) and

'.'

(empty cell) and given the integer

k

. You have to cut the pizza into

k

pieces using

k-1

cuts.

For each cut you choose the direction: vertical or horizontal, then you choose a cut position at the cell boundary and cut the pizza into two pieces. If you cut the pizza vertically, give the left part of the pizza to a person. If you cut the pizza horizontally, give the upper part of the pizza to a person. Give the last piece of pizza to the last person.

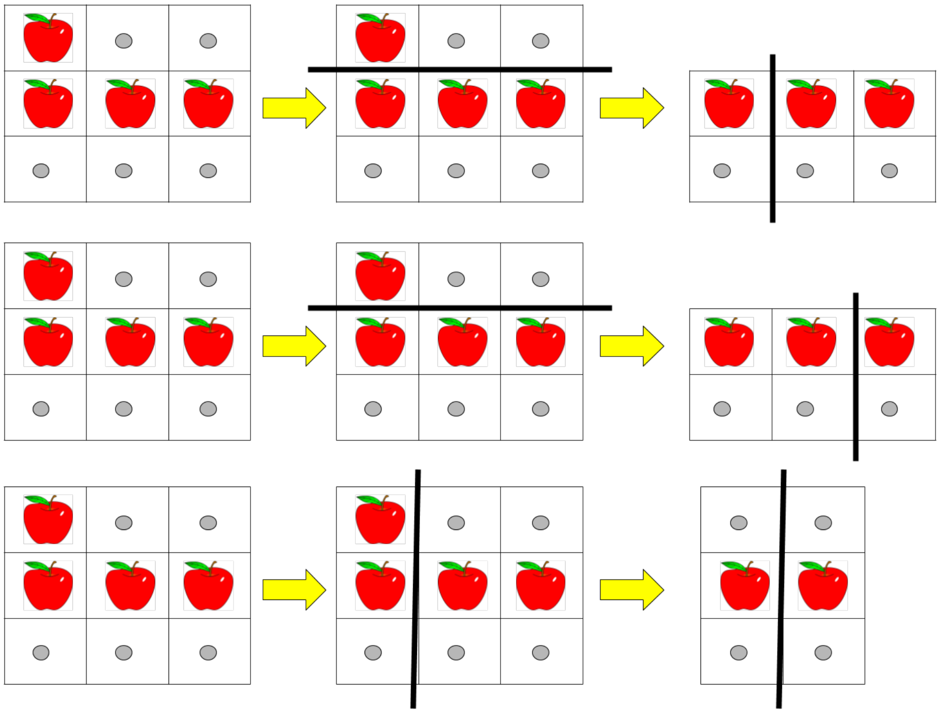
Return the number of ways of cutting the pizza such that each piece contains

at least

one apple.

Since the answer can be a huge number, return this modulo 10^9 + 7.

Example 1:



Input:

pizza = ["A..","AAA","..."], k = 3

Output:

3

Explanation:

The figure above shows the three ways to cut the pizza. Note that pieces must contain at least one apple.

Example 2:

Input:

pizza = ["A..","AA.","..."], k = 3

Output:

1

Example 3:

Input:

pizza = ["A..","A..","..."], k = 1

Output:

1

Constraints:

1 <= rows, cols <= 50

rows == pizza.length

cols == pizza[i].length

1 <= k <= 10

pizza

consists of characters

'A'

and

'.'

only.

Code Snippets

C++:

```
class Solution {
public:
int ways(vector<string>& pizza, int k) {

}
};
```

Java:

```
class Solution {
public int ways(String[] pizza, int k) {

}
}
```

Python3:

```
class Solution:
def ways(self, pizza: List[str], k: int) -> int:
```

Python:

```
class Solution(object):
def ways(self, pizza, k):
"""
:type pizza: List[str]
:type k: int
:rtype: int
```

```
"""
```

JavaScript:

```
/**
* @param {string[]} pizza
* @param {number} k
* @return {number}
*/
var ways = function(pizza, k) {

};
```

TypeScript:

```
function ways(pizza: string[], k: number): number {

};
```

C#:

```
public class Solution {
public int Ways(string[] pizza, int k) {

}
}
```

C:

```
int ways(char** pizza, int pizzaSize, int k) {

}
```

Go:

```
func ways(pizza []string, k int) int {

}
```

Kotlin:

```
class Solution {
fun ways(pizza: Array<String>, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func ways(_ pizza: [String], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn ways(pizza: Vec<String>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} pizza
# @param {Integer} k
# @return {Integer}
def ways(pizza, k)

end
```

PHP:

```
class Solution {

/**
* @param String[] $pizza
* @param Integer $k
* @return Integer
*/
function ways($pizza, $k) {

}
}
```

Dart:

```
class Solution {
int ways(List<String> pizza, int k) {

}
}
```

Scala:

```
object Solution {
def ways(pizza: Array[String], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec ways(pizza :: [String.t], k :: integer) :: integer
def ways(pizza, k) do

end
end
```

Erlang:

```
-spec ways(Pizza :: [unicode:unicode_binary()], K :: integer()) -> integer().
ways(Pizza, K) ->
.
```

Racket:

```
(define/contract (ways pizza k)
(-> (listof string?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways of Cutting a Pizza
* Difficulty: Hard
* Tags: array, 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 ways(vector<string>& pizza, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways of Cutting a Pizza
* Difficulty: Hard
* Tags: array, 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 ways(String[] pizza, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways of Cutting a Pizza
Difficulty: Hard
Tags: array, 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 ways(self, pizza: List[str], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def ways(self, pizza, k):
"""
:type pizza: List[str]
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways of Cutting a Pizza
* Difficulty: Hard
* Tags: array, 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[]} pizza
* @param {number} k
* @return {number}
*/
var ways = function(pizza, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways of Cutting a Pizza
* Difficulty: Hard
* Tags: array, 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 ways(pizza: string[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways of Cutting a Pizza
* Difficulty: Hard
* Tags: array, 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 Ways(string[] pizza, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways of Cutting a Pizza
* Difficulty: Hard
* Tags: array, 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
*/

int ways(char** pizza, int pizzaSize, int k) {

}
```

Go Solution:

```
// Problem: Number of Ways of Cutting a Pizza
// Difficulty: Hard
// Tags: array, 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 ways(pizza []string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun ways(pizza: Array<String>, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func ways(_ pizza: [String], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways of Cutting a Pizza
// Difficulty: Hard
// Tags: array, 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 ways(pizza: Vec<String>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} pizza
# @param {Integer} k
# @return {Integer}
def ways(pizza, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $pizza
* @param Integer $k
* @return Integer
*/
function ways($pizza, $k) {

}
}
```

Dart Solution:

```
class Solution {
int ways(List<String> pizza, int k) {
}
}
```

Scala Solution:

```
object Solution {
def ways(pizza: Array[String], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ways(pizza :: [String.t], k :: integer) :: integer
def ways(pizza, k) do

end
end
```

Erlang Solution:

```
-spec ways(Pizza :: [unicode:unicode_binary()], K :: integer()) -> integer().
ways(Pizza, K) ->
.
```

Racket Solution:

```
(define/contract (ways pizza k)
(-> (listof string?) exact-integer? exact-integer?)
)
```