Problem 474: Ones and Zeroes

Problem Information

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

Problem Description

You are given an array of binary strings

strs

and two integers

m

and

n

.

Return

the size of the largest subset of

strs

such that there are

at most

m

0

's and

n

1

's in the subset

.

A set

x

is a

subset

of a set

y

if all elements of

x

are also elements of

y

.

Example 1:

Input:

strs = ["10","0001","111001","1","0"], m = 5, n = 3

Output:

4

Explanation:

The largest subset with at most 5 0's and 3 1's is {"10", "0001", "1", "0"}, so the answer is 4. Other valid but smaller subsets include {"0001", "1"} and {"10", "1", "0"}. {"111001"} is an invalid subset because it contains 4 1's, greater than the maximum of 3.

Example 2:

Input:

strs = ["10","0","1"], m = 1, n = 1

Output:

2

Explanation:

The largest subset is {"0", "1"}, so the answer is 2.

Constraints:

1 <= strs.length <= 600

1 <= strs[i].length <= 100

strs[i]

consists only of digits

'0'

and

'1'

.

1 <= m, n <= 100

Code Snippets

C++:

```
class Solution {
public:
int findMaxForm(vector<string>& strs, int m, int n) {

}
};
```

Java:

```
class Solution {
public int findMaxForm(String[] strs, int m, int n) {

}
}
```

Python3:

```
class Solution:
def findMaxForm(self, strs: List[str], m: int, n: int) -> int:
```

Python:

```
class Solution(object):
def findMaxForm(self, strs, m, n):
"""
:type strs: List[str]
:type m: int
:type n: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} strs
* @param {number} m
* @param {number} n
* @return {number}
*/
var findMaxForm = function(strs, m, n) {

};
```

TypeScript:

```
function findMaxForm(strs: string[], m: number, n: number): number {

};
```

C#:

```
public class Solution {
public int FindMaxForm(string[] strs, int m, int n) {

}
}
```

C:

```
int findMaxForm(char** strs, int strsSize, int m, int n) {

}
```

Go:

```
func findMaxForm(strs []string, m int, n int) int {

}
```

Kotlin:

```
class Solution {
fun findMaxForm(strs: Array<String>, m: Int, n: Int): Int {

}
}
```

Swift:

```
class Solution {
func findMaxForm(_ strs: [String], _ m: Int, _ n: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_max_form(strs: Vec<String>, m: i32, n: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} strs
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def find_max_form(strs, m, n)

end
```

PHP:

```
class Solution {

/**
* @param String[] $strs
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function findMaxForm($strs, $m, $n) {

}
}
```

Dart:

```
class Solution {
int findMaxForm(List<String> strs, int m, int n) {

}
}
```

Scala:

```
object Solution {
def findMaxForm(strs: Array[String], m: Int, n: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_max_form(strs :: [String.t], m :: integer, n :: integer) ::
integer
def find_max_form(strs, m, n) do

end
end
```

Erlang:

```
-spec find_max_form(Strs :: [unicode:unicode_binary()], M :: integer(), N ::
integer()) -> integer().
find_max_form(Strs, M, N) ->
.
```

Racket:

```
(define/contract (find-max-form strs m n)
(-> (listof string?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Ones and Zeroes
* Difficulty: Medium
* Tags: array, string, 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 findMaxForm(vector<string>& strs, int m, int n) {

}
};
```

Java Solution:

```
/**
* Problem: Ones and Zeroes
* Difficulty: Medium
* Tags: array, string, 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 findMaxForm(String[] strs, int m, int n) {

}
}
```

Python3 Solution:

```
"""
Problem: Ones and Zeroes
Difficulty: Medium
Tags: array, string, 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 findMaxForm(self, strs: List[str], m: int, n: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMaxForm(self, strs, m, n):
"""
:type strs: List[str]
:type m: int
:type n: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Ones and Zeroes
* Difficulty: Medium
* Tags: array, string, 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[]} strs
* @param {number} m
* @param {number} n
* @return {number}
*/
var findMaxForm = function(strs, m, n) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Ones and Zeroes
* Difficulty: Medium
* Tags: array, string, 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 findMaxForm(strs: string[], m: number, n: number): number {

};
```

C# Solution:

```
/*
* Problem: Ones and Zeroes
* Difficulty: Medium
* Tags: array, string, 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 FindMaxForm(string[] strs, int m, int n) {

}
}
```

C Solution:

```
/*
* Problem: Ones and Zeroes
* Difficulty: Medium
```

```
* Tags: array, string, 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 findMaxForm(char** strs, int strsSize, int m, int n) {

}
```

Go Solution:

```
// Problem: Ones and Zeroes
// Difficulty: Medium
// Tags: array, string, 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 findMaxForm(strs []string, m int, n int) int {

}
```

Kotlin Solution:

```
class Solution {
fun findMaxForm(strs: Array<String>, m: Int, n: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func findMaxForm(_ strs: [String], _ m: Int, _ n: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Ones and Zeroes
// Difficulty: Medium
// Tags: array, string, 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 find_max_form(strs: Vec<String>, m: i32, n: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} strs
# @param {Integer} m
# @param {Integer} n
# @return {Integer}
def find_max_form(strs, m, n)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $strs
* @param Integer $m
* @param Integer $n
* @return Integer
*/
function findMaxForm($strs, $m, $n) {

}
}
```

Dart Solution:

```
class Solution {
int findMaxForm(List<String> strs, int m, int n) {

}
}
```

Scala Solution:

```
object Solution {
def findMaxForm(strs: Array[String], m: Int, n: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_max_form(strs :: [String.t], m :: integer, n :: integer) ::
integer
def find_max_form(strs, m, n) do

end
end
```

Erlang Solution:

```
-spec find_max_form(Strs :: [unicode:unicode_binary()], M :: integer(), N ::
integer()) -> integer().
find_max_form(Strs, M, N) ->
.
```

Racket Solution:

```
(define/contract (find-max-form strs m n)
(-> (listof string?) exact-integer? exact-integer? exact-integer?)
)
```