Problem 2514: Count Anagrams

Problem Information

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

Problem Description

You are given a string

s

containing one or more words. Every consecutive pair of words is separated by a single space

' '

.

A string

t

is an

anagram

of string

s

if the

i

th

word of

t

is a

permutation

of the

i

th

word of

s

.

For example,

"acb dfe"

is an anagram of

"abc def"

, but

"def cab"

and

"adc bef"

are not.

Return

the number of

distinct anagrams

of

s

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

s = "too hot"

Output:

18

Explanation:

Some of the anagrams of the given string are "too hot", "oot hot", "oto toh", "too toh", and "too oht".

Example 2:

Input:

s = "aa"

Output:

1

Explanation:

There is only one anagram possible for the given string.

Constraints:

1 <= s.length <= 10

5

s

consists of lowercase English letters and spaces

' '

.

There is single space between consecutive words.

Code Snippets

C++:

```
class Solution {
public:
int countAnagrams(string s) {

}
};
```

Java:

```
class Solution {
public int countAnagrams(String s) {

}
}
```

Python3:

```
class Solution:
def countAnagrams(self, s: str) -> int:
```

Python:

```
class Solution(object):
def countAnagrams(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
var countAnagrams = function(s) {

};
```

TypeScript:

```
function countAnagrams(s: string): number {

};
```

C#:

```
public class Solution {
public int CountAnagrams(string s) {
```

```
}
}
```

C:

```
int countAnagrams(char* s) {

}
```

Go:

```
func countAnagrams(s string) int {

}
```

Kotlin:

```
class Solution {
fun countAnagrams(s: String): Int {

}
}
```

Swift:

```
class Solution {
func countAnagrams(_ s: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_anagrams(s: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def count_anagrams(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countAnagrams($s) {

}
}
```

Dart:

```
class Solution {
int countAnagrams(String s) {

}
}
```

Scala:

```
object Solution {
def countAnagrams(s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_anagrams(s :: String.t) :: integer
def count_anagrams(s) do

end
end
```

Erlang:

```
-spec count_anagrams(S :: unicode:unicode_binary()) -> integer().
count_anagrams(S) ->
.
```

Racket:

```
(define/contract (count-anagrams s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Anagrams
* Difficulty: Hard
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int countAnagrams(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Anagrams
* Difficulty: Hard
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countAnagrams(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Anagrams
Difficulty: Hard
Tags: string, math, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def countAnagrams(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countAnagrams(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Anagrams
* Difficulty: Hard
```

```
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {string} s
* @return {number}
*/
var countAnagrams = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Anagrams
* Difficulty: Hard
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countAnagrams(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Anagrams
* Difficulty: Hard
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

public class Solution {
public int CountAnagrams(string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Anagrams
* Difficulty: Hard
* Tags: string, math, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int countAnagrams(char* s) {

}
```

Go Solution:

```
// Problem: Count Anagrams
// Difficulty: Hard
// Tags: string, math, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countAnagrams(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countAnagrams(s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countAnagrams(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Anagrams
// Difficulty: Hard
// Tags: string, math, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn count_anagrams(s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def count_anagrams(s)

end
```

PHP Solution:

```
class Solution {
```

```
/**
* @param String $s
* @return Integer
*/
function countAnagrams($s) {

}
}
```

Dart Solution:

```
class Solution {
int countAnagrams(String s) {

}
}
```

Scala Solution:

```
object Solution {
def countAnagrams(s: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_anagrams(s :: String.t) :: integer
def count_anagrams(s) do

end
end
```

Erlang Solution:

```
-spec count_anagrams(S :: unicode:unicode_binary()) -> integer().
count_anagrams(S) ->
.
```

Racket Solution:

```
(define/contract (count-anagrams s)
(-> string? exact-integer?)
)
```