Problem 115: Distinct Subsequences

Problem Information

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

Problem Description

Given two strings s and t, return

the number of distinct

subsequences

of

s

which equals

t.

The test cases are generated so that the answer fits on a 32-bit signed integer.

Example 1:

Input:

s = "rabbbit", t = "rabbit"

Output:

3

Explanation:

As shown below, there are 3 ways you can generate "rabbit" from s.

rabb

b

it

ra

b

bbit

rab

b

bit

Example 2:

Input:

s = "babgbag", t = "bag"

Output:

5

Explanation:

As shown below, there are 5 ways you can generate "bag" from s.

ba

b

g

bag

ba

bgba

g

b

abgb

ag

ba

b

gb

ag

babg

bag

Constraints:

1 <= s.length, t.length <= 1000

s

and

t

consist of English letters.

Code Snippets

C++:

```
class Solution {
public:
int numDistinct(string s, string t) {

}
};
```

Java:

```
class Solution {
public int numDistinct(String s, String t) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
* @param {string} t
```

```
 * @return {number}
 */
var numDistinct = function(s, t) {

};
```

TypeScript:

```
function numDistinct(s: string, t: string): number {

};
```

C#:

```
public class Solution {
public int NumDistinct(string s, string t) {

}
}
```

C:

```
int numDistinct(char* s, char* t) {

}
```

Go:

```
func numDistinct(s string, t string) int {

}
```

Kotlin:

```
class Solution {
fun numDistinct(s: String, t: String): Int {

}
}
```

Swift:

```
class Solution {
func numDistinct(_ s: String, _ t: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_distinct(s: String, t: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {Integer}
def num_distinct(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Integer
*/
function numDistinct($s, $t) {

}
}
```

Dart:

```
class Solution {
int numDistinct(String s, String t) {

}
```

```
}
```

Scala:

```
object Solution {
def numDistinct(s: String, t: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec num_distinct(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> integer().
num_distinct(S, T) ->
.
```

Racket:

```
(define/contract (num-distinct s t)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Distinct Subsequences
* Difficulty: Hard
* Tags: string, dp
```

```
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numDistinct(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Distinct Subsequences
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numDistinct(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Distinct Subsequences
Difficulty: Hard
Tags: string, dp

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
```

```
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Distinct Subsequences
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} s
* @param {string} t
* @return {number}
*/
var numDistinct = function(s, t) {

};
```

TypeScript Solution:

```
/**
* Problem: Distinct Subsequences
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numDistinct(s: string, t: string): number {

};
```

C# Solution:

```
/*
* Problem: Distinct Subsequences
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* 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 NumDistinct(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Distinct Subsequences
* Difficulty: Hard
* Tags: string, dp
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int numDistinct(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Distinct Subsequences
// Difficulty: Hard
// Tags: string, dp
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numDistinct(s string, t string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numDistinct(s: String, t: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numDistinct(_ s: String, _ t: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Distinct Subsequences
// Difficulty: Hard
// Tags: string, dp
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn num_distinct(s: String, t: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {Integer}
def num_distinct(s, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Integer
*/
function numDistinct($s, $t) {

}
}
```

Dart Solution:

```
class Solution {
int numDistinct(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def numDistinct(s: String, t: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec num_distinct(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> integer().
num_distinct(S, T) ->
.
```

Racket Solution:

```
(define/contract (num-distinct s t)
(-> string? string? exact-integer?)
)
```