Problem 484: Find Permutation

Problem Information

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

Problem Description

A permutation

perm

of

n

integers of all the integers in the range

[1, n]

can be represented as a string

s

of length

n - 1

where:

s[i] == 'I'

if

perm[i] < perm[i + 1]

, and

s[i] == 'D'

if

perm[i] > perm[i + 1]

.

Given a string

s

, reconstruct the lexicographically smallest permutation

perm

and return it.

Example 1:

Input:

s = "I"

Output:

[1,2]

Explanation:

[1,2] is the only legal permutation that can represented by s, where the number 1 and 2 construct an increasing relationship.

Example 2:

Input:

s = "DI"

Output:

[2,1,3]

Explanation:

Both [2,1,3] and [3,1,2] can be represented as "DI", but since we want to find the smallest lexicographical permutation, you should return [2,1,3]

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'I'

or

'D'

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> findPermutation(string s) {
}
};
```

Java:

```
class Solution {
public int[] findPermutation(String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function findPermutation(s: string): number[] {

};
```

C#:

```
public class Solution {
public int[] FindPermutation(string s) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPermutation(char* s, int* returnSize) {

}
```

Go:

```
func findPermutation(s string) []int {

}
```

Kotlin:

```
class Solution {
fun findPermutation(s: String): IntArray {

}
}
```

Swift:

```
class Solution {
func findPermutation(_ s: String) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn find_permutation(s: String) -> Vec<i32> {
}
}
```

Ruby:

```
# @param {String} s
# @return {Integer[]}
def find_permutation(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer[]
*/
function findPermutation($s) {

}
}
```

Dart:

```
class Solution {
List<int> findPermutation(String s) {

}
}
```

Scala:

```
object Solution {
def findPermutation(s: String): Array[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec find_permutation(S :: unicode:unicode_binary()) -> [integer()].
find_permutation(S) ->
.
```

Racket:

```
(define/contract (find-permutation s)
(-> string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Permutation
* Difficulty: Medium
* Tags: array, string, graph, greedy, stack
*
* 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:
vector<int> findPermutation(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Find Permutation
* Difficulty: Medium
* Tags: array, string, graph, greedy, stack
*
* 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[] findPermutation(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Permutation
Difficulty: Medium
Tags: array, string, graph, greedy, stack

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 findPermutation(self, s: str) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Permutation
* Difficulty: Medium
* Tags: array, string, graph, greedy, stack
*
* 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} s
* @return {number[]}
*/
var findPermutation = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Permutation
* Difficulty: Medium
* Tags: array, string, graph, greedy, stack
*
* 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 findPermutation(s: string): number[] {

};
```

C# Solution:

```
/*
* Problem: Find Permutation
* Difficulty: Medium
* Tags: array, string, graph, greedy, stack
*
```

```
* 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[] FindPermutation(string s) {

}
}
```

C Solution:

```
/*
* Problem: Find Permutation
* Difficulty: Medium
* Tags: array, string, graph, greedy, stack
*
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* findPermutation(char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: Find Permutation
// Difficulty: Medium
// Tags: array, string, graph, greedy, stack
//
// 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 findPermutation(s string) []int {
}
```

Kotlin Solution:

```
class Solution {
fun findPermutation(s: String): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func findPermutation(_ s: String) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Find Permutation
// Difficulty: Medium
// Tags: array, string, graph, greedy, stack
//
// 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 find_permutation(s: String) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer[]}
def find_permutation(s)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer[]
*/
function findPermutation($s) {

}
}
```

Dart Solution:

```
class Solution {
List<int> findPermutation(String s) {

}
}
```

Scala Solution:

```
object Solution {
def findPermutation(s: String): Array[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec find_permutation(S :: unicode:unicode_binary()) -> [integer()].
find_permutation(S) ->
.
```

Racket Solution:

```
(define/contract (find-permutation s)
(-> string? (listof exact-integer?))
)
```