Problem 942: DI String Match

Problem Information

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

Problem Description

A permutation

perm

of

n + 1

integers of all the integers in the range

[0, n]

can be represented as a string

s

of length

n

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 permutation

perm

and return it. If there are multiple valid permutations perm, return

any of them

.

Example 1:

Input:

s = "IDID"

Output:

[0,4,1,3,2]

Example 2:

Input:

s = "III"

Output:

[0,1,2,3]

Example 3:

Input:

s = "DDI"

Output:

[3,2,0,1]

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'I'

or

'D'

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> diStringMatch(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (di-string-match s)
(-> string? (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: DI String Match
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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> diStringMatch(string s) {

}
};
```

Java Solution:

```
/**
* Problem: DI String Match
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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[] diStringMatch(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: DI String Match
Difficulty: Easy
Tags: array, string, greedy

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: DI String Match
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 diStringMatch = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: DI String Match
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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 diStringMatch(s: string): number[] {

};
```

C# Solution:

```
/*
* Problem: DI String Match
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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[] DiStringMatch(string s) {

}
}
```

C Solution:

```
/*
* Problem: DI String Match
* Difficulty: Easy
* Tags: array, string, greedy
*
* 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* diStringMatch(char* s, int* returnSize) {

}
```

Go Solution:

```
// Problem: DI String Match
// Difficulty: Easy
// Tags: array, string, greedy
//
// 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 diStringMatch(s string) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: DI String Match
// Difficulty: Easy
// Tags: array, string, greedy
//
// 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 di_string_match(s: String) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (di-string-match s)
(-> string? (listof exact-integer?))
)
```