Problem 1177: Can Make Palindrome from Substring

Problem Information

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

Problem Description

You are given a string

s

and array

queries

where

queries[i] = [left

i

, right

i

, k

i

]

. We may rearrange the substring

s[left

i

...right

i

]

for each query and then choose up to

k

i

of them to replace with any lowercase English letter.

If the substring is possible to be a palindrome string after the operations above, the result of the query is

true

. Otherwise, the result is

false

.

Return a boolean array

answer

where

answer[i]

is the result of the

i

th

query

queries[i]

.

Note that each letter is counted individually for replacement, so if, for example

s[left

i

...right

i

] = "aaa"

, and

k

i

= 2

, we can only replace two of the letters. Also, note that no query modifies the initial string

s

.

Example :

Input:

s = "abcda", queries = [[3,3,0],[1,2,0],[0,3,1],[0,3,2],[0,4,1]]

Output:

[true,false,false,true,true]

Explanation:

queries[0]: substring = "d", is palidrome. queries[1]: substring = "bc", is not palidrome. queries[2]: substring = "abcd", is not palidrome after replacing only 1 character. queries[3]: substring = "abcd", could be changed to "abba" which is palidrome. Also this can be changed to "baab" first rearrange it "bacd" then replace "cd" with "ab". queries[4]: substring = "abcda", could be changed to "abcba" which is palidrome.

Example 2:

Input:

s = "lyb", queries = [[0,1,0],[2,2,1]]

Output:

[false,true]

Constraints:

1 <= s.length, queries.length <= 10

5

0 <= left

i

<= right

i

< s.length

0 <= k

i

<= s.length

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<bool> canMakePaliQueries(string s, vector<vector<int>>& queries) {

}
};
```

Java:

```
class Solution {
public List<Boolean> canMakePaliQueries(String s, int[][] queries) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def canMakePaliQueries(self, s, queries):
```

```
"""
:type s: str
:type queries: List[List[int]]
:rtype: List[bool]
"""
```

JavaScript:

```
/**
* @param {string} s
* @param {number[][]} queries
* @return {boolean[]}
*/
var canMakePaliQueries = function(s, queries) {

};
```

TypeScript:

```
function canMakePaliQueries(s: string, queries: number[][]): boolean[] {

};
```

C#:

```
public class Solution {
public IList<bool> CanMakePaliQueries(string s, int[][] queries) {

}
}
```

C:

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

}
```

Go:

```
func canMakePaliQueries(s string, queries [][]int) []bool {

}
```

Kotlin:

```
class Solution {
fun canMakePaliQueries(s: String, queries: Array<IntArray>): List<Boolean> {

}
}
```

Swift:

```
class Solution {
func canMakePaliQueries(_ s: String, _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust:

```
impl Solution {
pub fn can_make_pali_queries(s: String, queries: Vec<Vec<i32>>) -> Vec<bool>
{

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer[][]} queries
# @return {Boolean[]}
def can_make_pali_queries(s, queries)

end
```

PHP:

```
class Solution {

/**
```

```
* @param String $s
* @param Integer[][] $queries
* @return Boolean[]
*/
function canMakePaliQueries($s, $queries) {

}
}
```

Dart:

```
class Solution {
List<bool> canMakePaliQueries(String s, List<List<int>> queries) {

}
}
```

Scala:

```
object Solution {
def canMakePaliQueries(s: String, queries: Array[Array[Int]]): List[Boolean]
= {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_make_pali_queries(s :: String.t, queries :: [[integer]]) ::
[boolean]
def can_make_pali_queries(s, queries) do

end
end
```

Erlang:

```
-spec can_make_pali_queries(S :: unicode:unicode_binary(), Queries ::
[[integer()]]) -> [boolean()].
can_make_pali_queries(S, Queries) ->
.
```

Racket:

```
(define/contract (can-make-pali-queries s queries)
(-> string? (listof (listof exact-integer?)) (listof boolean?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Can Make Palindrome from Substring
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
vector<bool> canMakePaliQueries(string s, vector<vector<int>>& queries) {

}
};
```

Java Solution:

```
/**
* Problem: Can Make Palindrome from Substring
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public List<Boolean> canMakePaliQueries(String s, int[][] queries) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Can Make Palindrome from Substring
Difficulty: Medium
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def canMakePaliQueries(self, s: str, queries: List[List[int]]) -> List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Can Make Palindrome from Substring
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

/**
* @param {string} s
* @param {number[][]} queries
* @return {boolean[]}
*/
var canMakePaliQueries = function(s, queries) {

};
```

TypeScript Solution:

```
/**
* Problem: Can Make Palindrome from Substring
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function canMakePaliQueries(s: string, queries: number[][]): boolean[] {

};
```

C# Solution:

```
/*
* Problem: Can Make Palindrome from Substring
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public IList<bool> CanMakePaliQueries(string s, int[][] queries) {
```

```
}
}
```

C Solution:

```
/*
* Problem: Can Make Palindrome from Substring
* Difficulty: Medium
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
bool* canMakePaliQueries(char* s, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Can Make Palindrome from Substring
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func canMakePaliQueries(s string, queries [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
fun canMakePaliQueries(s: String, queries: Array<IntArray>): List<Boolean> {

}
}
```

Swift Solution:

```
class Solution {
func canMakePaliQueries(_ s: String, _ queries: [[Int]]) -> [Bool] {

}
}
```

Rust Solution:

```
// Problem: Can Make Palindrome from Substring
// Difficulty: Medium
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn can_make_pali_queries(s: String, queries: Vec<Vec<i32>>) -> Vec<bool>
{

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer[][]} queries
# @return {Boolean[]}
def can_make_pali_queries(s, queries)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer[][] $queries
* @return Boolean[]
*/
function canMakePaliQueries($s, $queries) {

}
}
```

Dart Solution:

```
class Solution {
List<bool> canMakePaliQueries(String s, List<List<int>> queries) {

}
}
```

Scala Solution:

```
object Solution {
def canMakePaliQueries(s: String, queries: Array[Array[Int]]): List[Boolean]
= {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_make_pali_queries(s :: String.t, queries :: [[integer]]) ::
[boolean]
def can_make_pali_queries(s, queries) do

end
end
```

Erlang Solution:

```
-spec can_make_pali_queries(S :: unicode:unicode_binary(), Queries ::
[[integer()]]) -> [boolean()].
can_make_pali_queries(S, Queries) ->
.
```

Racket Solution:

```
(define/contract (can-make-pali-queries s queries)
(-> string? (listof (listof exact-integer?)) (listof boolean?))
)
```