

Problem 2983: Palindrome Rearrangement Queries

Problem Information

Difficulty: **Hard**

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

You are given a

0-indexed

string

s

having an

even

length

n

.

You are also given a

0-indexed

2D integer array,

queries

, where

queries[i] = [a

i

, b

i

, c

i

, d

i

]

.

For each query

i

, you are allowed to perform the following operations:

Rearrange the characters within the

substring

s[a

i

:b

i

]

, where

$0 \leq a$

i

$\leq b$

i

$< n / 2$

.

Rearrange the characters within the

substring

s[c

i

:d

i

]

, where

$n / 2 \leq c$

i

$\leq d$

i

< n

.

For each query, your task is to determine whether it is possible to make

s

a

palindrome

by performing the operations.

Each query is answered

independently

of the others.

Return

a

0-indexed

array

answer

, where

answer[i] == true

if it is possible to make

s

a palindrome by performing operations specified by the

i

th

query, and

false

otherwise.

A

substring

is a contiguous sequence of characters within a string.

$s[x:y]$

represents the substring consisting of characters from the index

x

to index

y

in

s

,

both inclusive

.

Example 1:

Input:

`s = "abcabc", queries = [[1,1,3,5],[0,2,5,5]]`

Output:

`[true,true]`

Explanation:

In this example, there are two queries: In the first query: - a

0

= 1, b

0

= 1, c

0

= 3, d

0

= 5. - So, you are allowed to rearrange `s[1:1]` => a

b

cabc and `s[3:5]` => abc

abc

. - To make s a palindrome, `s[3:5]` can be rearranged to become => abc

cba

. - Now, s is a palindrome. So, answer[0] = true. In the second query: - a

1

= 0, b

1

= 2, c

1

= 5, d

1

= 5. - So, you are allowed to rearrange s[0:2] =>

abc

abc and s[5:5] => abcab

c

. - To make s a palindrome, s[0:2] can be rearranged to become =>

cba

abc. - Now, s is a palindrome. So, answer[1] = true.

Example 2:

Input:

s = "abbcdecbbba", queries = [[0,2,7,9]]

Output:

[false]

Explanation:

In this example, there is only one query. a

0

= 0, b

0

= 2, c

0

= 7, d

0

= 9. So, you are allowed to rearrange s[0:2] =>

abb

cdecbbba and s[7:9] => abbcdec

bba

. It is not possible to make s a palindrome by rearranging these substrings because s[3:6] is not a palindrome. So, answer[0] = false.

Example 3:

Input:

s = "acbcab", queries = [[1,2,4,5]]

Output:

[true]

Explanation:

In this example, there is only one query. a

0

= 1, b

0

= 2, c

0

= 4, d

0

= 5. So, you are allowed to rearrange s[1:2] => a

cb

cab and s[4:5] => acbc

ab

. To make s a palindrome s[1:2] can be rearranged to become a

bc

cab. Then, s[4:5] can be rearranged to become abcc

ba

. Now, s is a palindrome. So, answer[0] = true.

Constraints:

$2 \leq n \leq s.length \leq 10$

5

$1 \leq queries.length \leq 10$

5

`queries[i].length == 4`

`a`

`i`

`== queries[i][0], b`

`i`

`== queries[i][1]`

`c`

`i`

`== queries[i][2], d`

`i`

`== queries[i][3]`

$0 \leq a$

`i`

$\leq b$

`i`

$< n/2$

$n/2 \leq c$

i

$\leq d$

i

$< n$

n

is even.

s

consists of only lowercase English letters.

Code Snippets

C++:

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

    }

};
```

Java:

```
class Solution {
    public boolean[] canMakePalindromeQueries(String s, int[][] queries) {

    }

}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
    public bool[] CanMakePalindromeQueries(string s, int[][] queries) {

    }
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
    fun canMakePalindromeQueries(s: String, queries: Array<IntArray>):
BooleanArray {

    }
}
```

Swift:

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

    }
}
```

Rust:

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

    }
}
```

Ruby:

```

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

end

```

PHP:

```

class Solution {

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

    }

}

```

Dart:

```

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

  }

}

```

Scala:

```

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

    }

}

```

Elixir:

```

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

```

```

def can_make_palindrome_queries(s, queries) do

end

end

```

Erlang:

```

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

```

Racket:

```

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

```

Solutions

C++ Solution:

```

/*
 * Problem: Palindrome Rearrangement Queries
 * Difficulty: Hard
 * 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> canMakePalindromeQueries(string s, vector<vector<int>>& queries)
    {

    }

};

```

Java Solution:

```
/**
 * Problem: Palindrome Rearrangement Queries
 * Difficulty: Hard
 * 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 boolean[] canMakePalindromeQueries(String s, int[][] queries) {

}

}
```

Python3 Solution:

```
"""
Problem: Palindrome Rearrangement Queries
Difficulty: Hard
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 canMakePalindromeQueries(self, s: str, queries: List[List[int]]) ->
List[bool]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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



```
:rtype: List[bool]
"""
```

JavaScript Solution:

```
/**
 * Problem: Palindrome Rearrangement Queries
 * Difficulty: Hard
 * 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 canMakePalindromeQueries = function(s, queries) {

};
```

TypeScript Solution:

```
/**
 * Problem: Palindrome Rearrangement Queries
 * Difficulty: Hard
 * 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 canMakePalindromeQueries(s: string, queries: number[][]): boolean[]
{

};
```

C# Solution:

```
/*
 * Problem: Palindrome Rearrangement Queries
 * Difficulty: Hard
 * 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 bool[] CanMakePalindromeQueries(string s, int[][] queries) {

    }
}
```

C Solution:

```
/*
 * Problem: Palindrome Rearrangement Queries
 * Difficulty: Hard
 * 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* canMakePalindromeQueries(char* s, int** queries, int queriesSize, int*
queriesColSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Palindrome Rearrangement Queries
// Difficulty: Hard
// 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 canMakePalindromeQueries(s string, queries [][]int) []bool {

}
```

Kotlin Solution:

```
class Solution {
    fun canMakePalindromeQueries(s: String, queries: Array<IntArray>):
        BooleanArray {

    }
}
```

Swift Solution:

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

    }
}
```

Rust Solution:

```
// Problem: Palindrome Rearrangement Queries
// Difficulty: Hard
// 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_palindrome_queries(s: String, queries: Vec<Vec<i32>>) ->
        Vec<bool> {

    }
}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

    }

}
```

Dart Solution:

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

  }

}
```

Scala Solution:

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

  }

}
```

Elixir Solution:

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

  end
end
```

Erlang Solution:

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

Racket Solution:

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