String Rotation

We'll discuss how to determine if two strings are rotations of one another. We'll go over a brute-force solution and an elegant solution.

String Rotation

Instructions

Create a function that takes in 2 strings as parameters.

Return `true` if the strings are rotations of each other. Otherwise, return `false`.

Input: String, String

Output: Boolean

Examples

The following sets of strings are rotations:

```
"rotation"  "tationro" "tionrota"

"javascript"  "scriptjava"  "iptjavascr"

"RotateMe"  "teMeRota"  "eRotateM"
```

Hints

- You may have to spend time thinking about how best to arrange these strings before processing in a loop.
- Think about ways to short-circuit the check.

```
function stringRotation(str1, str2) {
    // Your code here
}
```

Solution 1

```
function stringRotation(str1, str2) {
    if(str1.length !== str2.length) {
        return false;
    }

    for(let i = 0; i < str1.length; i++) {
        const rotation = str1.slice(i, str1.length) + str1.slice(0, i);

        if(rotation === str2) {
            return true;
        }
    }

    return false;
}
```

How it Works

This is the brute-force solution.

Before we start the loop, we make sure that the strings have the same number of characters. If they don't we automatically know that they're not rotations of each other.

In the for-loop, we're rotating `str1` repeatedly, one character at a time. To see this, insert a log statement and print out `rotation` in the loop.

At every iteration, we check if it's equal to `str2`. If it is, we stop and return `true`. Eventually, we get through the entire string. If we never find that they're equal, we return `false`.

Time

The for-loop already tells us it's going to at least be `O(n)`.

The string concatenation on line 7 is likely O(n) as well. According to the spec for String.prototype.slice (https://tc39.github.io/ecma262/#sec-string.prototype.slice), it seems as if internally, slicing a string occurs in a loop.

The equality check on line 9 is also an `O(n)` operation, as each character needs to be compared.

The `O(n)` for-loop combined with its two `O(n)` internal processes bring the time complexity to:

O(n^2).

Space

The space complexity is:

O(n).

The amount of memory `rotation` needs is proportional to the number of characters in the input strings.

Solution 2

This solution requires some intense critical thinking.

Hints

- Try thinking about the relationship between a string and its rotation. Try to short-circuit the check from Solution 1.

```
function stringRotation(str1, str2) {
    return str1.length === str2.length && (str1 + str1).includes(str2);
}
```

How it Works

If we take a string and create a new string by repeating it, it will contain all possible rotations of the string.

Time

n here will represent the length of the strings. n will be calculated for the case where the two strings are the same length.

Checking and comparing string length is constant-time, `O(n)`.

The amount of time it takes to add a string to itself is proportional to the length of the string, so `(str1 + str1)` is `O(n)`.

Checking string equality is also `O(n)` - all characters must be compared.

So, the final time complexity is

O(n).

Space

Space complexity is determined by the `(str1 + str1)` statement above. That temporary value will be proportional to the length of the strings, so:

O(n).

← Back

Highest Frequency

Next →

Array Subset

Mark as Completed

Report an Issue