Balanced Brackets

Learn how to determine if brackets are balanced or not. This is an actual problem that software developers must occasionally solve and many dev tools have their own implementation of it. We'll learn how a code editor can pick up our mistakes and highlight exactly which brackets don't match.

Balanced Brackets

Instructions

Given a string, return `true` if it contains all balanced parentheses `()`, curly-brackets `{}`, and square-brackets `[]`.

Input: String

Output: Boolean

Examples

```
isBalanced("(x + y) - (4)"); // -> true
isBalanced("(((10 ) ()) ((?)(:)))"); // -> true
isBalanced("[{()}]"); // -> true
isBalanced("(50)("); // -> false
isBalanced("[{]}"); // -> false
```

```
1  function isBalanced(str) {
2      // Your code here
3  }
```

Solution

```
    const openStack = [];
    const open = '([{';
    const close = ')]}';
    const matches = {
        ')': '(',
        ']': '[',
        '}': '{'
    };

    for (let i = 0; i < str.length; i++) {
        const char = str[i];

        // If it's an open bracket, push it into our array
        if(open.includes(char)) {
            openStack.push(char);

        // If it's a close bracket
        } else if(close.includes(char)) {
            // pop an item from the open brackets array.
            const lastOpenBracket = openStack.pop();

            // If the open and close bracket don't match, return false
            if(matches[char] !== lastOpenBracket) {
                return false;
            }
        }
    }

    // Ensures there are no characters left on the stack
    return !openStack.length;
}
```

How it Works

In our for-loop, we process every character. If the current character is an open bracket, we push it onto our `openStack` array.

If it's not a bracket, we do nothing.

If it's a close bracket, we pop the top off of our `openStack` array. We make sure the two items match. If they don't, our string isn't balanced and we can return `false`.

Once we're done processing the string, we have to check to make sure there are no open brackets left on our stack. If there are, our string is unbalanced.

Time

This is a solution with a time complexity of:

O(n).

Every character is processed in a loop. Inside the loop, we perform only constant-time actions.

Space

The space complexity is:

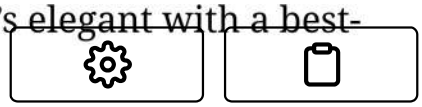
O(n).

Characters are stored in an array, generally proportional to the size of the input.

Conclusion

There are many ways to skin this cat. Another solution would have been to have two pointers, one at the beginning and one at the end, moving towards the middle. They would be checking to make sure that brackets matched until they met. This would also be an `O(n)` solution.

The solution I've chosen makes the most sense to me. It's elegant with a best-case time complexity.



← Back

Sorted Search

Next →

Creating a Queue with O(1) Operations

Mark as Completed

Report an Issue