balanced or not hackerrank solution

Understanding the Balanced or Not HackerRank Problem

Balanced or not HackerRank solution is a popular problem that tests one's understanding of data structures and their applications, particularly in the context of strings and stacks. The problem involves determining if the brackets in a given string are balanced, which is a common task in programming, especially in compilers and interpreters. In this article, we will explore the definition of balanced brackets, how to approach solving this problem, and a detailed solution using a stack data structure.

What Does it Mean for Brackets to be Balanced?

A string containing various types of brackets is considered balanced if:

1. Every opening bracket has a corresponding closing bracket.
2. Brackets are closed in the correct order.

For example, the following strings are balanced:
- `{[()]}`
- `{{[[(())]]}}`

Conversely, the following strings are not balanced:
- `{[(])}`
- `{{[[(())]]}}}`
- `{{]}}`

Types of Brackets

The problem typically involves three types of brackets:
- Parentheses: `()`
- Square brackets: `[]`
- Curly braces: `{}`

Each type of bracket must be opened and closed in the correct order, following Last In, First Out (LIFO) principles, which are characteristic of stack data structures.

Approach to Solve the Problem

To determine if a string is balanced, we can utilize a stack data structure. The stack allows us to efficiently manage and check the rules of bracket balancing. Here’s a step-by-step approach:

1. Initialize a Stack: Create an empty stack that will be used to keep track of opening brackets.

2. Iterate Through the String: For each character in the input string:
- If it is an opening bracket (`(`, `{`, `[`), push it onto the stack.
- If it is a closing bracket (`)`, `}`, `]`):
- Check if the stack is empty. If it is, then the string is not balanced.
- If the stack is not empty, pop the top element from the stack and check whether it matches the closing bracket. If it does not match, the string is not balanced.

3. Final Check: After processing all characters, if the stack is empty, the string is balanced; if it is not empty, the string is not balanced.

Sample Implementation

Here is a Python implementation of the above algorithm:

```python
def is_balanced(s):
stack = []
bracket_map = {')': '(', '}': '{', ']': '['}

for char in s:
if char in bracket_map.values(): if it's an opening bracket
stack.append(char)
elif char in bracket_map.keys(): if it's a closing bracket
if not stack or stack[-1] != bracket_map[char]: mismatch or empty stack
return "NO"
stack.pop() pop the last opening bracket

return "YES" if not stack else "NO"

Example usage
input_string = "{[()]}"
print(is_balanced(input_string)) Output: YES
```

Explanation of the Code

- Stack Initialization: We start by initializing an empty stack.
- Bracket Mapping: We create a dictionary to map closing brackets to their corresponding opening brackets.
- Iterate Through Characters: For each character in the string:
- If it’s an opening bracket, we push it onto the stack.
- If it’s a closing bracket, we check if the stack is empty or if the top of the stack does not match the

corresponding opening bracket. If either condition is true, the function returns "NO".
- Final Return: After the loop, if the stack is empty, it means all brackets were matched and closed properly, hence we return "YES". If there are any unmatched opening brackets left in the stack, we return "NO".

Complexity Analysis

When analyzing the time and space complexity of the balanced or not HackerRank solution, we find:

- Time Complexity: O(n), where n is the length of the input string. We traverse the string once, and each operation (push/pull from the stack) is O(1).
- Space Complexity: O(n) in the worst case, where all characters are opening brackets, leading to n elements in the stack.

Common Mistakes to Avoid

When tackling the balanced or not problem, there are several common pitfalls to be aware of:

1. Ignoring Non-bracket Characters: If the string contains characters other than brackets, ensure that they are either ignored or handled appropriately.
2. Mismatched Brackets: Failing to check for mismatched pairs can lead to incorrect results. Always validate that the correct opening bracket is matched with the closing one.
3. Stack Underflow: Attempting to pop from an empty stack can lead to errors. Always check if the stack is empty before popping.

Conclusion

The balanced or not HackerRank solution is a fundamental exercise that provides insight into the use of stacks for managing nested structures. By following a systematic approach, we can effectively determine whether a string of brackets is balanced. This problem not only enhances our programming skills but also prepares us for more complex challenges in data structure and algorithm design. Understanding the principles behind stack operations and how they apply to real-world coding challenges is invaluable for any aspiring developer.

Frequently Asked Questions

What is the 'balanced or not' problem on HackerRank?

The 'balanced or not' problem on HackerRank requires you to determine if a given string of parentheses is balanced. A string is considered balanced if every opening bracket has a corresponding closing bracket and they are properly nested.

What data structure is commonly used to solve the 'balanced or not' problem?

A stack is commonly used to solve the 'balanced or not' problem. You can push opening brackets onto the stack and pop them when a closing bracket is encountered, checking for matches in the process.

How do you handle mismatched parentheses in the 'balanced or not' solution?

In the solution, if you encounter a closing bracket and the stack is empty or the top of the stack does not match the corresponding opening bracket, the string is unbalanced. You can return false at that point.

What are the time and space complexities of the 'balanced or not' algorithm?

The time complexity of the 'balanced or not' algorithm is O(n), where n is the length of the string, since you need to traverse the entire string once. The space complexity is also O(n) in the worst case due to the stack.

Can the 'balanced or not' problem handle different types of parentheses?

Yes, the 'balanced or not' problem can be extended to handle different types of parentheses (e.g., {}, [], ()) as long as you modify the solution to check for matching pairs accordingly.

What are common edge cases to consider for the 'balanced or not' solution?

Common edge cases include an empty string (which is considered balanced), strings with only opening or only closing brackets, and strings with nested brackets of different types.