Valid Parentheses - Solution Approach and Notes

Valid Parentheses - Solution Approach and Notes

Introduction:

Valid Parentheses introduces a fundamental use of stacks to verify balanced and correctly ordered brackets, building essential skills in problem-solving and algorithmic thinking. This is my first publication of this solution, and I look forward to publishing a more efficient version in the future.

Approach:

This solution utilizes a stack-based approach to validate the correct opening and closing of parentheses in the input string `s`.

1. Dictionary Setup with Closing Brackets as Keys:

- The dictionary `pairs = {")":"(", "]":"[", "}":"{"}` is defined with **closing brackets as keys** and their corresponding opening brackets as values.
- By setting up the dictionary this way, we can check each character in the string `s` to see if it's a closing bracket and, if so, quickly find its matching opening bracket.
- Using closing brackets as keys helps in checking pairs, especially when we encounter a closing bracket and need to see if it matches the last open bracket in the stack.

2. Stack Usage for Open Brackets:

- We use the list `open_bracket` as a stack to keep track of open brackets as we iterate through the string.
- The stack works on a **Last-In-First-Out (LIFO)** basis, meaning the last added bracket must be matched by the next closing bracket we encounter to be valid.
- This stack helps us check pairs in the correct order, as we "push" open brackets onto it and "pop" them off when a matching closing bracket is found.

3. Checking for Open Brackets First:

- For each character `i` in the string `s`, we first check if it's an open bracket using `if i in pairs.values()`.

- Starting with open brackets makes sure we capture all unmatched opening brackets in the stack.
- Only when we encounter a closing bracket do we check if it matches the last open bracket on the stack, ensuring proper nesting and order.

4. Returning False for Unmatched Closing Brackets:
   - If we encounter a closing bracket before any open bracket has been added to the stack (meaning the stack is empty), or if it doesn't match the last open bracket, we return `False` immediately.
   - This immediate return ensures that the string `s` fails validation as soon as an invalid closing bracket appears, improving efficiency.

5. Final Check for Unmatched Open Brackets:
   - After iterating through the entire string `s`, we check the stack `open_bracket`.
   - If the stack is empty (`len(open_bracket) == 0`), it means all open brackets had corresponding closing brackets in the correct order, so we return `True`.
   - If there are any remaining open brackets in the stack, it implies an unmatched open bracket, so we return `False`.

Summary:

- This solution is efficient because it stops immediately upon detecting an invalid closing bracket.
- Using a dictionary with closing brackets as keys, combined with a stack for open brackets, ensures a well-organized and efficient pairing process.