SQP5 Valid Parentheses

Sunday, November 20, 2022 12:28 PM

Given a string s containing just the characters '(', ')', '{', '}', '[' and ']', determine if the input string is valid.

An input string is valid if:

1. Open brackets must be closed by the same type of brackets.
2. Open brackets must be closed in the correct order.
3. Every close bracket has a corresponding open bracket of the same type.

Example 1:
Input: s = "()"
Output: true

This is the one of the most popular problem based on stack and Queue
This is also known as Balanced parenthases

→ Balanced parenthases means if we have any opening bracket then must have its closing bracket

Example s = ( ) [{ } ( )]

→ Now we can say the Given String is the Balanced

Example s =] ( ) { }

→ Now "]" this have not Opening bracket

Important Note's—

s = [ ) ( ]

→ In this case (C) have also (]) but still this is Not a balanced bracket

(C) opening Bracket, (]) closing bracket

→ Every Opening bracket should be its closing bracket in front of Opening Not backward

→ s = [( ])

→ again we can not say this is the Balanced parenthesees

→ Because "(" its closing bracket is after another "] " closing bracket that's why this is also Not a balanced Parenthesees.

Thought Process

→ Now in this problem we used Stack data structure (LIFO)

s = [ ]

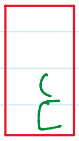
String = [( ) [{ ( ) }]]

→ Second Iteration

→ What we will do is whenever comes any Opening brackets take and put into the stack

if i == '['
if i == '('
if i == '{'
} → put into the stack

Stack =



i =)

→ Now whenever comes any closing Bracket then

→ check if Stack is Not Empty ✓

↳ that means might be that closing bracket exist

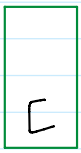
→ take the top most element from the stack

element = stack.pop()

↳ and check the opposite bracket can exist or not for current i =)

if i == ')' and element == '('
that means that bracket is balanced moved to the next bracket

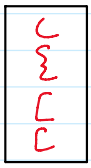
Stack =



→ moved to the Next Bracket
i = [

→ again all Continuous Opening bracket put into the stack

Stack =



[( ) [{ ( ) }]]

i =)

→ Again same process follow that we follow in previously step

(<)

→ Again same process follow that we follow in previously step

→ check if stack is Not Empty () ✓

↳ element = stack.pop()
element = '('

and check if ==) and element == (✓
that means that bracket is also balanced
So moved to the next bracket

Stack = { [[

→ Now Next all the bracket are the closing bracket (}]])

↳ So Again follow same Approach check stack is Not Empty pop one by one and check its Opasite Bracket present or Not

Edge cases

→ if the first element is Any closing Bracket so directly between that is Not Balanced parantheses

→ if any time pop element and i are both are Not Opasite to each other or come ② different bracket So also directly between that is also Not Balanced parantheses

→ if ② is Not opening bracket and stack is Empty So also that is Not balanced parantheses

→ that are the same edge case that are remember during solving this problem

→ And last if our stack is Empty then given string are balanced

after complete the hole iteration check
if len(stack) == 0:
that means given bracket are balanced
Otherwise Not balanced

Time Complexity = O(N)
↳ because we traverse the length of the given string

Space Complicity = O(N)
↳ because we used a stack data structure

Implementation Python

```python
# lc : https://leetcode.com/problems/valid-parentheses/
# gfg: https://practice.geeksforgeeks.org/problems/parenthesis-checker2744/1?utm_source=gfg&utm_medium=article&utm_campaign=bottom_sticky_on_article

class Solution:
    def isValid(self, s):

        Stack = []
        for i in s:
            if i in ['(', '{', '[']:
                Stack.append(i)

            elif i == ')' and len(Stack) != 0 and Stack[-1] == '(':
                Stack.pop()

            elif i == '}' and len(Stack) != 0 and Stack[-1] == '{':
                Stack.pop()

            elif i == ']' and len(Stack) != 0 and Stack[-1] == '[':
                Stack.pop()

            else:
                return False

        if len(Stack) == 0:
            return True
```

Implementation Java

```java
import java.util.*;
import java.util.Stack;

public class L5_Valid_Parentheses {

    public static void main(String[] args) {
        System.out.println("Valid Parentheses");
    }
}

class Solution1 {
    public boolean isValid(String s) {
        Stack<Character> arr = new Stack<>();
        for(char c: s.toCharArray()){
            if (c == '(' || c == '{' || c == '['){
                arr.push(c);
            }
            else if (c == ')' && !arr.isEmpty() && arr.peek() == '('){
                arr.pop();
            }
            else if (c == '}' && !arr.isEmpty() && arr.peek() == '{'){
                arr.pop();
            }
            else if (c == ']' && !arr.isEmpty() && arr.peek() == '['){

                arr.pop();
            }
            else {
                return false;
            }
        }
        return arr.isEmpty();
    }
}
```

This concept is applicable for any type Balanced parathesis problem
→ Just carry apply it