20. Valid Parentheses

Hint

Easy 21.8K 1.4K

Companies

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
```

Example 2:

```
Input: s = "()[]{}"
Output: true
```

Example 3:

```
Input: s = "(]"
Output: false
```

Constraints:

- `1 <= s.length <= 10^4`
- `s` consists of parentheses only `'()[]{}'`.

→ Take a stack

Take a stack

→ now start scanning each character c of given string

1) if c is ')' then top of stack must have '('
if not return false.

similarly

2) if c is ']' then top of stack must have '['
if not return false

3) if c is '}' then top of stack must have '{'
if not return false.

4) if c is '(' or '{' or '[' then push it.

5) if c is ')' and top is '('

or

c is ']' and top is '['

or

c is '}' and top is '{'

Then pop.

After entire string is processed,

```
if ( stack is empty)
        string is valid
else
        string is invalid.
```

T(n) : O(n)
S(n) : O(n)

Approach 2:

we can also implement using unordered map along with stack to avoid multiple if else.

→ Create a map m = { {'(', ')'}, }

{'[', ']'},
{'{', '}'}, }

```
for (auto c : s)
{
    if( m.find(c) != m.end() )
        st.push(c)
    else if( stack is empty || m[st.top()] == c)
        st.pop()
    else
        return false
}

return stack.empty()
```

T(n) : O(n)
S(n) : O(n) + O(n)