20. Valid Parentheses

Easy 👍 7795 👎 315 ♡ Add to List Share

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.

Example 1:

```
Input: s = "()"
Output: true
```

Example 2:

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

Example 3:

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

Example 4:

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

Example 5:

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

Constraints:

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

Related Topics

String Stack

Similar Questions

Generate Parentheses	Medium
Longest Valid Parentheses	Hard
Remove Invalid Parentheses	Hard
Check If Word Is Valid After Substitutions	Medium

Hide Hint 1

An interesting property about a valid parenthesis expression is that a sub-expression of a valid expression should also be a valid expression. (Not every sub-expression) e.g.

```
{ { } [ ] [ [ [ ] ] ] } is VALID expression
          [ [ [ ] ] ]    is VALID sub-expression
  { } [ ]                is VALID sub-expression
```

Can we exploit this recursive structure somehow?

Hide Hint 2

What if whenever we encounter a matching pair of parenthesis in the expression, we simply remove it from the expression? This would keep on shortening the expression. e.g.

```
{ { ( { } ) } }
      |_|

{ { (      ) } }
    |______|

{ {          } }
  |__________|

{                }
|________________|

VALID EXPRESSION!
```

Hide Hint 3

The **stack** data structure can come in handy here in representing this recursive structure of the problem. We can't really process this from the inside out because we don't have an idea about the overall structure. But, the stack can help us process this recursively i.e. from outside to inwards.