🚀 Leetcode Top Interview Questions

20. Valid Parentheses

Easy

Tags:
string | stack
Companies:
airbnb | amazon | bloomberg | facebook | google | microsoft | twitter | zenefits

---

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
```

Constraints:

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

Algorithm:

- First We will take an empty stack.
- If '(' or '{' or '[' is encountered, then we will push that into the stack.
- If ')' is encountered and last appended element in p is '(', then I will pop the last appended '(' Suppose, Stack = ['[', '{', '('] and I got ')', then ['[', '{', '(', ')']. The last 2 element will cancel out because of valid parenthases
- In the same way, If '}' is encountered and last appended element in p is '{', then I will pop the last appended '{'
- If ']' is encountered and last appended element in p is '[', then I will pop the last appended '['.
- In other condition we will return False Because, Suppose Stack = ['[', '{', '('] and we got '}'.Now, Stack = ['[', '{', '(', '}']. So, this is invalid.
- At the end there is no element in the stack, then that is a valid staring. So, return True
- Else, Return False

Time complexity: O(n)

Solutions: Using stack, dictionary / list / set

Approach #1:

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        for k in s:
            if k == "(" or k == "[" or k == "{":
                stack.append(k)
            elif stack:
                last_elem = stack.pop()
                if k == ")" and last_elem == "(":
                    continue
                elif k == "]" and last_elem == "[":
                    continue
                elif k == "}" and last_elem == "{":
                    continue
                else:
                    return False
            else:
                return False

        return True if not stack else False
```

Approach #2:

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
```

```
        for bracket in s:
            if bracket in ["(", "{", "["]:
                stack.append(bracket)
            else:
                if not stack:
                    return False
                curBracket = stack.pop()
                if curBracket == "(":
                    if bracket != ")":
                        return False
                elif curBracket == "{":
                    if bracket != "}":
                        return False
                elif curBracket == "[":
                    if bracket != "]":
                        return False
        if stack:
            return False
        else:
            return True
```

Approach #3:

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []

        for char in s:
            if char == '(' or char == '[' or char == '{':
                stack.append(char)
            else:
                if not stack:
                    return False
                elif stack:
                    p = stack.pop()
                    a = (p == '(' and char == ')')
                    b = (p == '[' and char == ']')
                    c = (p == '{' and char == '}')
                    if not a and not b and not c:
                        return False

        return not stack
```

Approach #4:

```
class Solution:
    def isValid(self, s: str) -> bool:
```

```
        if len(s)%2 != 0:
            return False

        Dict = {'(': ')', '{': '}', '[': ']'}
        stack = []
        dictKeys = Dict.keys()

        for i in s:
            if i in dictKeys:
                stack.append(i)
            else:
                if stack == []:
                    return False
                a = stack.pop()
                if i != Dict[a]:
                    return False

        return stack == []
```

Approach #5:

```
class Solution:
    def isValid(self, s: str) -> bool:
        dic, stack = {')': '(', ']': '[', '}': '{'}, []
        for i in s:
            if stack and dic.get(i) == stack[-1]:
                stack.pop()
            else:
                if i in dic:
                    return False
                stack.append(i)
        return not stack
```

Approach #6:

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        for i in s:
            if i == '(' or i == '{' or i == '[':
                stack.append(i)
            elif stack and stack[-1] == '(' and i == ')':
                stack.pop()
            elif stack and stack[-1] == '{' and i == '}':
                stack.pop()
            elif stack and stack[-1] == '[' and i == ']':
                stack.pop()
            else:
                return False
```

```
        if len(stack) == 0:
            return True
        return False
```

Approach #7:

```
class Solution:
    def isValid(self, s):
        stack = ['#']
        for c in s:
            if c == '(':
                stack.append(')')
            elif c == '{':
                stack.append('}')
            elif c == '[':
                stack.append(']')
            elif c != stack.pop():
                return False
        return stack == ['#']
```

Approach #8:

```
class Solution(object):
    def isValid(self, s):

        if len(s)%2!=0:
            return False
        opening = set('([{')
        matches = set([('(', ')'), ('[', ']'), ('{', '}')])
        stack = []

        for paren in s:
            if paren in opening:
                stack.append(paren)
            else:
                if len(stack) == 0:
                    return False
                last_open = stack.pop()
                if (last_open, paren) not in  matches:
                    return False
        return len(stack)==0
```

Approach #9:

```
def isValid(self, s: str) -> bool:
    while "()" in s or "[]" in s or "{}" in s:
        s = s.replace("()", "")
        s = s.replace("[]", "")
        s = s.replace("{}", "")
    return s == ""
```

Approach #10:

```
class Solution:
    def isValid(self, s: str) -> bool:
        d = {"}":"{", ")":"(", "]":"["}
        i = 0
        while i < len(s):
            if s[i] not in d:
                i += 1
            else:
                if i == 0:
                    return False
                else:
                    if s[i-1] == d[s[i]]:
                        s = s[:i-1]+s[i+1:]
                        i -=1
                    else:
                        return False
        if len(s) > 0:
            return False
        return True
```