# 20. Valid Parentheses

| | |
|---|---|
| Created | @July 16, 2020 12:26 AM |
| Difficulty | Easy |
| LC Url | |
| Importance | **** |
| Tag | Stack |
| Video | |

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 `'()[]{}'`.

## Solution

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

        for c in s:
            if c not in Map:
                stack.append(c)
                continue
            if not stack or stack[-1] != Map[c]:
                return False
            stack.pop()

        return not stack
```

```
class Solution:
    def isValid(self, s: str) -> bool:
        stack = []
        for i in range(len(s)):
            char_i = s[i]
            if char_i in ['(', '[', '{']:
                stack.append(char_i)
            elif char_i in [')', ']', '}']:
                if not stack:
                    return False

                cur = stack.pop()
                if (cur == '(' and char_i != ')') \
                        or (cur == '[' and char_i != ']') \
                        or (cur == '}' and char_i != '}'):
                    return False
        if not stack:
            return True
        return False
```

```
class Solution {
    public boolean isValid(String s) {
        Stack<Character> mark = new Stack<Character>();
        for (int i = 0; i < s.length(); i++) {
            char char_i = s.charAt(i);
            if (char_i == '(' || char_i == '[' || char_i == '{') {
                mark.push(char_i);
            } else if (char_i == ')' || char_i == ']' || char_i == '}'){
                if (mark.isEmpty()) return false;
                char cur = mark.pop();

                if (cur == '(' && char_i != ')') return false;
                if (cur == '[' && char_i != ']') return false;
                if (cur == '{' && char_i != '}') return false;
            }
        }
        if (mark.isEmpty()) return true;
        return false;
    }
}
```

# 155. Min Stack

| | |
|---|---|
| Created | @June 25, 2021 12:17 AM |
| Difficulty | Medium |
| LC Url | https://leetcode.com/problems/min-stack/ |
| Importance | |
| Tag | NEET Stack |
| Video | |

Design a stack that supports push, pop, top, and retrieving the minimum element in constant time.

Implement the `MinStack` class:

- `MinStack()` initializes the stack object.
- `void push(val)` pushes the element `val` onto the stack.
- `void pop()` removes the element on the top of the stack.
- `int top()` gets the top element of the stack.
- `int getMin()` retrieves the minimum element in the stack.

**Example 1:**

```
Input
["MinStack","push","push","push","getMin","pop","top","getMin"]
[[],[-2],[0],[-3],[],[],[],[]]

Output
[null,null,null,null,-3,null,0,-2]

Explanation
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin(); // return -3
minStack.pop();
minStack.top();    // return 0
minStack.getMin(); // return -2
```

**Constraints:**

- `2` `31` ` <= val <= 2` `31` ` - 1`
- Methods `pop`, `top` and `getMin` operations will always be called on **non-empty** stacks.
- At most `3 * 10` `4` calls will be made to `push`, `pop`, `top`, and `getMin`.

# Solution

```
class MinStack:

    def __init__(self):
        """
        initialize your data structure here.
        """
        self.stack = []
        self.minStack = []

    def push(self, val: int) -> None:
        self.stack.append(val)
        val = min(val, self.minStack[-1] if self.minStack else val)
        self.minStack.append(val)

    def pop(self) -> None:
        self.stack.pop()
        self.minStack.pop()

    def top(self) -> int:
        return self.stack[-1]

    def getMin(self) -> int:
        return self.minStack[-1]


# Your MinStack object will be instantiated and called as such:
# obj = MinStack()
# obj.push(val)
# obj.pop()
# param_3 = obj.top()
# param_4 = obj.getMin()
```

# 150. Evaluate Reverse Polish Notation

| | |
|---|---|
| Created | @March 2, 2022 9:51 PM |
| Difficulty | Medium |
| LC Url | https://leetcode.com/problems/evaluate-reverse-polish-notation/ |
| Importance | |
| Tag | NEET Stack |
| Video | https://www.youtube.com/watch?v=iu0082c4HDE |

Evaluate the value of an arithmetic expression in Reverse Polish Notation.

Valid operators are `+`, `-`, `*`, and `/`. Each operand may be an integer or another expression.

**Note** that division between two integers should truncate toward zero.

It is guaranteed that the given RPN expression is always valid. That means the expression would always evaluate to a result, and there will not be any division by zero operation.

**Example 1:**

```
Input: tokens = ["2","1","+","3","*"]
Output: 9
Explanation: ((2 + 1) * 3) = 9
```

**Example 2:**

```
Input: tokens = ["4","13","5","/","+"]
Output: 6
Explanation: (4 + (13 / 5)) = 6
```

**Example 3:**

```
Input: tokens = ["10","6","9","3","+","-11","*","/","*","17","+","5","+"]
Output: 22
Explanation: ((10 * (6 / ((9 + 3) * -11))) + 17) + 5
= ((10 * (6 / (12 * -11))) + 17) + 5
= ((10 * (6 / -132)) + 17) + 5
= ((10 * 0) + 17) + 5
= (0 + 17) + 5
= 17 + 5
= 22
```

**Constraints:**

- `1 <= tokens.length <= 10` `4`
- `tokens[i]` is either an operator: `"+"`, `"-"`, `"*"`, or `"/"`, or an integer in the range `[-200, 200]`.

## Solution

```
class Solution:
    def evalRPN(self, tokens: List[str]) -> int:
        stack = []
        for c in tokens:
            if c == '+':
                stack.append(stack.pop() + stack.pop())
            elif c == '-':
                post_c, pre_c = stack.pop(), stack.pop()
                stack.append(pre_c - post_c)
            elif c == '*':
                stack.append(stack.pop() * stack.pop())
            elif c == '/':
                post_c, pre_c = stack.pop(), stack.pop()
                stack.append(int(pre_c / post_c))
            else:
                stack.append(int(c))
        return stack[0]
```

# 22. Generate Parentheses

| | |
|---|---|
| Created | @July 16, 2020 2:02 AM |
| Difficulty | Medium |
| LC Url | https://leetcode.com/problems/generate-parentheses/ |
| Importance | |
| Tag | NEET Stack backtrack |
| Video | |

Given `n` pairs of parentheses, write a function to *generate all combinations of well-formed parentheses*.

**Example 1:**

```
Input: n = 3
Output: ["((()))","(()())","(())()","()(())","()()()"]
```

**Example 2:**

```
Input: n = 1
Output: ["()"]
```

**Constraints:**

- `1 <= n <= 8`

## Solution

### 方法一：暴力法

#### 思路

我们可以生成所有 $2^{2n}$ 个 '(' 和 ')' 字符构成的序列，然后我们检查每一个是否有效即可。

#### 算法

为了生成所有序列，我们可以使用递归。长度为 $n$ 的序列就是在长度为 $n-1$ 的序列前加一个 '(' 或 ')'。

为了检查序列是否有效，我们遍历这个序列，并使用一个变量 $balance$ 表示左括号的数量减去右括号的数量。如果在遍历过程中 $balance$ 的值小于零，或者结束时 $balance$ 的值不为零，那么该序列就是无效的，否则它是有效的。

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        def generate(A):
            if len(A) == 2*n:
                if valid(A):
                    ans.append("".join(A))
            else:
                A.append('(')
                generate(A)
                A.pop()
                A.append(')')
                generate(A)
                A.pop()

        def valid(A):
            bal = 0
            for c in A:
                if c == '(': bal += 1
                else: bal -= 1
                if bal < 0: return False
            return bal == 0

        ans = []
        generate([])
        return ans

# 作者：LeetCode-Solution
# 链接：https://leetcode.cn/problems/generate-parentheses/solution/gua-hao-sheng-cheng-by-leetcode-solution/
# 来源：力扣（LeetCode）
# 著作权归作者所有。商业转载请联系作者获得授权，非商业转载请注明出处。
```

### 复杂度分析

- 时间复杂度：$O(2^{2n}n)$，对于 $2^{2n}$ 个序列中的每一个，我们用于建立和验证该序列的复杂度为 $O(n)$。
- 空间复杂度：$O(n)$，除了答案数组之外，我们所需要的空间取决于递归栈的深度，每一层递归函数需要 $O(1)$ 的空间，最多递归 $2n$ 层，因此空间复杂度为 $O(n)$。

### 回溯法

方法一还有改进的余地：我们可以只在序列仍然保持有效时才添加 '(' 或 ')'，而不是像 方法一 那样每次添加。我们可以通过跟踪到目前为止放置的左括号和右括号的数目来做到这一点，

如果左括号数量不大于 $n$，我们可以放一个左括号。如果右括号数量小于左括号的数量，我们可以放一个右括号。

```python
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        stack = []
        res = []

        def backtrack(openN, closedN):
            if openN == closedN == n:
                res.append(''.join(stack))
                return

            if openN < n:
                stack.append('(')
                backtrack(openN + 1, closedN)
                stack.pop()

            if closedN < openN:
                stack.append(')')
                backtrack(openN, closedN + 1)
                stack.pop()

        backtrack(0, 0)
        return res
```

我们的复杂度分析依赖于理解 $generateParenthesis(n)$ 中有多少个元素。这个分析超出了本文的范畴，但事实证明这是第 $n$ 个卡特兰数 $\frac{1}{n+1}\binom{2n}{n}$，这是由 $\frac{4^n}{n\sqrt{n}}$ 渐近界定的。

- 时间复杂度：$O(\frac{4^n}{\sqrt{n}})$，在回溯过程中，每个答案需要 $O(n)$ 的时间复制到答案数组中。
- 空间复杂度：$O(n)$，除了答案数组之外，我们所需要的空间取决于递归栈的深度，每一层递归函数需要 $O(1)$ 的空间，最多递归 $2n$ 层，因此空间复杂度为 $O(n)$。

力扣

https://leetcode.cn/problems/generate-parentheses/solution/gua-hao-sheng-cheng-by-leetcode-solution/

# 739. Daily Temperatures

|  |  |
|---|---|
| Created | @October 14, 2022 2:56 PM |
| Difficulty | Medium |
| LC Url | https://leetcode.com/problems/daily-temperatures/ |
| Importance |  |
| Tag | NEET Stack |
| Video |  |

Given an array of integers `temperatures` represents the daily temperatures, return *an array* `answer` *such that* `answer[i]` *is the number of days you have to wait after the* `i` `th` *day to get a warmer temperature*. If there is no future day for which this is possible, keep `answer[i] == 0` instead.

**Example 1:**

```
Input: temperatures = [73,74,75,71,69,72,76,73]
Output: [1,1,4,2,1,1,0,0]
```

**Example 2:**

```
Input: temperatures = [30,40,50,60]
Output: [1,1,1,0]
```

**Example 3:**

```
Input: temperatures = [30,60,90]
Output: [1,1,0]
```

**Constraints:**

- `1 <= temperatures.length <= 10` `5`

- `30 <= temperatures[i] <= 100`

## Solution

```
class Solution:
    def dailyTemperatures(self, temperatures: List[int]) -> List[int]:
        res = [0] * len(temperatures)
        stack = []  # pair: [temp, index]

        for i, t in enumerate(temperatures):
            while stack and t > stack[-1][0]:
                stackT, stackInd = stack.pop()
                res[stackInd] = i - stackInd
            stack.append((t, i))
        return res
```

# 853. Car Fleet

| | |
|---|---|
| Created | @October 14, 2022 7:51 PM |
| Difficulty | Medium |
| LC Url | https://leetcode.com/problems/car-fleet/ |
| Importance | |
| Tag | NEET Stack |
| Video | |

There are `n` cars going to the same destination along a one-lane road. The destination is `target` miles away.

You are given two integer array `position` and `speed`, both of length `n`, where `position[i]` is the position of the `i` `th` car and `speed[i]` is the speed of the `i` `th` car (in miles per hour).

A car can never pass another car ahead of it, but it can catch up to it and drive bumper to bumper **at the same speed**. The faster car will **slow down** to match the slower car's speed. The distance between these two cars is ignored (i.e., they are assumed to have the same position).

A **car fleet** is some non-empty set of cars driving at the same position and same speed. Note that a single car is also a car fleet.

If a car catches up to a car fleet right at the destination point, it will still be considered as one car fleet.

Return *the **number of car fleets** that will arrive at the destination*.

**Example 1:**

```
Input: target = 12, position = [10,8,0,5,3], speed = [2,4,1,1,3]
Output: 3
Explanation:
The cars starting at 10 (speed 2) and 8 (speed 4) become a fleet, meeting each other at 1
2.
The car starting at 0 does not catch up to any other car, so it is a fleet by itself.
The cars starting at 5 (speed 1) and 3 (speed 3) become a fleet, meeting each other at 6.
```

```
 The fleet moves at speed 1 until it reaches target.
Note that no other cars meet these fleets before the destination, so the answer is 3.
```

### Example 2:

```
Input: target = 10, position = [3], speed = [3]
Output: 1
Explanation: There is only one car, hence there is only one fleet.
```

### Example 3:

```
Input: target = 100, position = [0,2,4], speed = [4,2,1]
Output: 1
Explanation:
The cars starting at 0 (speed 4) and 2 (speed 2) become a fleet, meeting each other at 4.
 The fleet moves at speed 2.
Then, the fleet (speed 2) and the car starting at 4 (speed 1) become one fleet, meeting ea
ch other at 6. The fleet moves at speed 1 until it reaches target.
```

### Constraints:

- `n == position.length == speed.length`
- `1 <= n <= 10` `5`
- `0 < target <= 10` `6`
- `0 <= position[i] < target`
- All the values of `position` are **unique**.
- `0 < speed[i] <= 10` `6`

# Solution

```
class Solution:
    def carFleet(self, target: int, position: List[int], speed: List[int]) -> int:
        pair = [(p, s) for p, s in zip(position, speed)]
        pair.sort(reverse=True)
        stack = []
        for p, s in pair:  # Reverse Sorted Order
            stack.append((target - p) / s)
            if len(stack) >= 2 and stack[-1] <= stack[-2]:
                stack.pop()
        return len(stack)
```