0241. 为运算表达式设计优先级

ITCharge 大约 2 分钟

- 标签：递归、记忆化搜索、数学、字符串、动态规划
- 难度：中等

题目链接

- 0241. 为运算表达式设计优先级 - 力扣

题目大意

描述：给定一个由数字和运算符组成的字符串 `expression` 。

要求：按不同优先级组合数字和运算符，计算并返回所有可能组合的结果。你可以按任意顺序返回答案。

说明：

- 生成的测试用例满足其对应输出值符合 32 位整数范围，不同结果的数量不超过 10^4。
- $1 \leq expression.length \leq 20$。
- `expression` 由数字和算符 `'+'` 、 `'-'` 和 `'*'` 组成。
- 输入表达式中的所有整数值在范围 $[0, 99]$。

示例：

- 示例 1：

```py
输入：expression = "2-1-1"
输出：[0,2]
解释：
((2-1)-1) = 0
(2-(1-1)) = 2
```

- 示例 2：

```py
输入：expression = "2*3-4*5"
输出：[-34,-14,-10,-10,10]
解释：
(2*(3-(4*5))) = -34
((2*3)-(4*5)) = -14
((2*(3-4))*5) = -10
(2*((3-4)*5)) = -10
(((2*3)-4)*5) = 10
```

解题思路

思路 1：分治算法

给定的字符串 `expression` 只包含有数字和字符，可以写成类似 `x op y` 的形式，其中 x、y 为表达式或数字，op 为字符。

则我们可以根据字符的位置，将其递归分解为 x、y 两个部分，接着分别计算 x 部分的结果与 y 部分的结果。然后再将其合并。

思路 1：代码

```py
class Solution:
    def diffWaysToCompute(self, expression: str) -> List[int]:
        res = []
        if len(expression) <= 2:
            res.append(int(expression))
            return res

        for i in range(len(expression)):
            ch = expression[i]
            if ch == '+' or ch == '-' or ch == '*':
                left_cnts = self.diffWaysToCompute(expression[ :i])
                right_cnts = self.diffWaysToCompute(expression[i + 1:])

                for left in left_cnts:
                    for right in right_cnts:
                        if ch == '+':
                            res.append(left + right)
```

```
                elif ch == '-':
                    res.append(left - right)
                else:
                    res.append(left * right)

        return res
```

思路 1：复杂度分析

- **时间复杂度**：$O(C_n)$，其中 n 为结果数组的大小，C_n 是第 n 个卡特兰数。
- **空间复杂度**：$O(C_n)$。