# LeetCode 241

https://leetcode.com/problems/different-ways-to-add-parentheses/description/

Yifeng Zeng

## Description

241. Different Ways to Add Parentheses

Given a string of numbers and operators, return all possible results from computing all the different possible ways to group numbers and operators. The valid operators are +, - and *.

```
Example 1
Input: "2-1-1".

((2-1)-1) = 0
(2-(1-1)) = 2
Output: [0, 2]

Example 2
Input: "2*3-4*5"

(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
Output: [-34, -14, -10, -10, 10]
```

## Idea Report

The basic idea of add parentheses is to prioritize the operators in different orders. And we need to list possible results, so this is a search problem. We can enumerate the lowest priority of the operator, and this operator will split the input into three parts. Part 1, the string on the left of this operator; Part 2, this operator; Part 3, the string on the right of this operator. The we can recursively do the same calculation for part 1 and 3, and calculate the left result and right result by using current operator, pair by pair.

Code:

```java
class Solution {
    public List<Integer> diffWaysToCompute(String input) {
        List<Integer> res = new ArrayList<>();

        if (isDigit(input)) {
            res.add(Integer.parseInt(input));
            return res;
        }

        for (int i = 0; i < input.length(); i++) {
            char ch = input.charAt(i);
            if (!Character.isDigit(ch)) {
                List<Integer> left = diffWaysToCompute(input.substring(0, i));
                List<Integer> right = diffWaysToCompute(input.substring(i + 1));
                for (int l : left) {
                    for (int r : right) {
                        int num = 0;
                        if (ch == '+') {
                            res.add(l + r);
                        } else if (ch == '-') {
                            res.add(l - r);
                        } else if (ch == '*') {
                            res.add(l * r);
                        }
                    }
                }
            }
        }

        return res;
    }

    private boolean isDigit(String str) {
        return str.matches("[0-9]+");
    }
}
```

## Summary

- Divide and conquer.
- Search problem if asked to list all possible solution.
- DFS