# Divide and Conquer

The third common pattern in Backtracking is divide and conquer. This is something like a top down resolution, first divide the big problem into small parts, then divide the smaller part further to smaller pieces, until the basic unit. We can use memorization for a range which is already searched.

The counter part of this solution is the Dynamic Programming which build from bottom to top.

## 241. Different Ways to Add Parentheses

Medium

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"

**Output:** [0, 2]

**Explanation:**

((2-1)-1) = 0

(2-(1-1)) = 2

**Example 2:**

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

**Output:** [-34, -14, -10, -10, 10]

**Explanation:**

(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

### Analysis:

When we see an operator, we can simply divide the expression as two parts, left and right, and the recursive call the function to get all the possible result for left and right.

/// <summary>

/// Leet code #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]

/// </summary>

vector<int> LeetCodeDFS::diffWaysToCompute(string input)

{

vector<int> result;

for (size\_t i = 0; i <= input.size(); i++)

{

if ((input[i] == '+') || (input[i] == '-') || (input[i] == '\*'))

{

vector<int> left = diffWaysToCompute(input.substr(0, i));

vector<int> right = diffWaysToCompute(input.substr(i + 1));

for (size\_t j = 0; j < left.size(); j++)

{

for (size\_t k = 0; k < right.size(); k++)

{

if (input[i] == '+') result.push\_back(left[j] + right[k]);

else if (input[i] == '-') result.push\_back(left[j] - right[k]);

else if (input[i] == '\*') result.push\_back(left[j] \* right[k]);

}

}

}

}

if (result.empty())

{

result.push\_back(atoi(input.c\_str()));

}

return result;

}