

Problem Challenge 2

Problem Challenge 3

Challenge 2

Solution Review: Problem

Solution Review: Problem Challenge 1

₩

? Ask a Question

```
We'll cover the following
Evaluate Expression (hard)
Solution
Code
Time complexity
Space complexity
Memoized version
```

Evaluate Expression (hard)

Given an expression containing digits and operations (+, -, *), find all possible ways in which the expression can be evaluated by grouping the numbers and operators using parentheses.

Example 1:

```
Input: "1+2*3"
Output: 7, 9
Explanation:
  1+(2*3) => 7
  (1+2)*3 => 9
```

Example 2:

```
Input: "2*3-4-5"
Output: 8, -12, 7, -7, -3
Explanation:
   2*(3-(4-5)) => 8
   2*(3-4-5) => -12
   2*3-(4-5) => 7
   2*(3-4)-5 => -7
   (2*3)-4-5 => -3
```

Solution

This problem follows the Subsets pattern and can be mapped to Balanced Parentheses. We can follow a similar BFS approach.

Let's take Example-1 mentioned above to generate different ways to evaluate the expression.

- 1. We can iterate through the expression character-by-character.
- 2. we can break the expression into two halves whenever we get an operator (+, -, *).
- 3. The two parts can be calculated by recursively calling the function.
- 4. Once we have the evaluation results from the left and right halves, we can combine them to produce all results.

Code

Here is what our algorithm will look like:

```
Python3
                        ⊗ C++
                                     Js JS
👙 Java
    import java.util.*;
                                                                                                   C →
    class EvaluateExpression {
       public static List<Integer> diffWaysToEvaluateExpression(String input) {
         List<Integer> result = new ArrayList<>();
         // base case: if the input string is a number, parse and add it to output.
         if (!input.contains("+") && !input.contains("-") && !input.contains("*")) {
           result.add(Integer.parseInt(input));
         } else {
           for (int i = 0; i < input.length(); i++) {</pre>
 10
 11
             char chr = input.charAt(i);
             if (!Character.isDigit(chr)) {
 13
               // break the equation here into two parts and make recursively calls
               List<Integer> leftParts = diffWaysToEvaluateExpression(input.substring(0, i));
 14
               List<Integer> rightParts = diffWaysToEvaluateExpression(input.substring(i + 1));
 15
               for (int part1 : leftParts) {
 16
                 for (int part2 : rightParts) {
                   if (chr == '+')
 18
                     result.add(part1 + part2);
 19
                   else if (chr == '-')
 20
                     result.add(part1 - part2);
 21
                   else if (chr == '*')
 22
 23
                     result.add(part1 * part2);
 24
 25
 26
27
 28
 Run
                                                                                     Save
                                                                                               Reset
```

Time complexity

The time complexity of this algorithm will be exponential and will be similar to Balanced Parentheses. Estimated time complexity will be $O(N*2^N)$ but the actual time complexity ($O(4^n/\sqrt{n})$) is bounded by the Catalan number and is beyond the scope of a coding interview. See more details here.

Space complexity

The space complexity of this algorithm will also be exponential, estimated at $O(2^N)$ though the actual will be ($O(4^n/\sqrt{n})$.

Memoized version# The problem has overlapping subproblems, as our recursive calls can be evaluating the same sub-

expression multiple times. To resolve this, we can use memoization and store the intermediate results in a **HashMap**. In each function call, we can check our map to see if we have already evaluated this sub-expression before. Here is the memoized version of our algorithm; please see highlighted changes:

```
Python3
                        ⊗ C++
                                    JS JS
👙 Java
    import java.util.*;
                                                                                                   class EvaluateExpression {
      // memoization map
      Map<String, List<Integer>> map = new HashMap<String, List<Integer>>();
      public List<Integer> diffWaysToEvaluateExpression(String input) {
        if (map.containsKey(input))
          return map.get(input);
        List<Integer> result = new ArrayList<>();
10
        // base case: if the input string is a number, parse and return it.
11
        if (!input.contains("+") && !input.contains("-") && !input.contains("*")) {
          result.add(Integer.parseInt(input));
13
14
        } else {
15
          for (int i = 0; i < input.length(); i++) {</pre>
16
            char chr = input.charAt(i);
            if (!Character.isDigit(chr)) {
17
              List<Integer> leftParts = diffWaysToEvaluateExpression(input.substring(0, i));
18
              List<Integer> rightParts = diffWaysToEvaluateExpression(input.substring(i + 1));
19
              for (int part1 : leftParts) {
20
                for (int part2 : rightParts) {
21
                  if (chr == '+')
22
                     result.add(part1 + part2);
23
                  else if (chr == '-')
24
25
                     result.add(part1 - part2);
                  else if (chr == '*')
26
                     result.add(part1 * part2);
27
28
                                                                                     Save
 Run
                                                                                              Reset
```

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. See how ①

