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]`

Credits:
Special thanks to @mithmatt for adding this problem and creating all test cases.

Solution 1

```
public class Solution {
    public List<Integer> diffWaysToCompute(String input) {
        List<Integer> ret = new LinkedList<Integer>();
        for (int i=0; i<input.length(); i++) {
            if (input.charAt(i) == '-' ||
                input.charAt(i) == '*' ||
                input.charAt(i) == '+' ) {
                String part1 = input.substring(0, i);
                String part2 = input.substring(i+1);
                List<Integer> part1Ret = diffWaysToCompute(part1);
                List<Integer> part2Ret = diffWaysToCompute(part2);
                for (Integer p1 :   part1Ret) {
                    for (Integer p2 :   part2Ret) {
                        int c = 0;
                        switch (input.charAt(i)) {
                            case '+': c = p1+p2;
                                break;
                            case '-': c = p1-p2;
                                break;
                            case '*': c = p1*p2;
                                break;
                        }
                        ret.add(c);
                    }
                }
            }
        }
        if (ret.size() == 0) {
            ret.add(Integer.valueOf(input));
        }
        return ret;
    }
}
```

written by 2guotou original link here

Solution 2

Here is the basic recursive solution

```
class Solution {
public:
    vector<int> diffWaysToCompute(string input) {
        vector<int> result;
        int size = input.size();
        for (int i = 0; i < size; i++) {
            char cur = input[i];
            if (cur == '+' || cur == '-' || cur == '*') {
                // Split input string into two parts and solve them recursively
                vector<int> result1 = diffWaysToCompute(input.substr(0, i));
                vector<int> result2 = diffWaysToCompute(input.substr(i+1));
                for (auto n1 : result1) {
                    for (auto n2 : result2) {
                        if (cur == '+')
                            result.push_back(n1 + n2);
                        else if (cur == '-')
                            result.push_back(n1 - n2);
                        else
                            result.push_back(n1 * n2);
                    }
                }
            }
        }
        // if the input string contains only number
        if (result.empty())
            result.push_back(atoi(input.c_str()));
        return result;
    }
};
```

There are many repeating subquestions in this recursive method, therefore, we could use dynamic programming to avoid this situation by saving the results for subquestions. Here is the DP solution.

```
class Solution {
public:
    vector<int> diffWaysToCompute(string input) {
        unordered_map<string, vector<int>> dpMap;
        return computeWithDP(input, dpMap);
    }

    vector<int> computeWithDP(string input, unordered_map<string, vector<int>> &d
pMap) {
        vector<int> result;
        int size = input.size();
        for (int i = 0; i < size; i++) {
            char cur = input[i];
            if (cur == '+' || cur == '-' || cur == '*') {
                // Split input string into two parts and solve them recursively
                vector<int> result1, result2;
                string substr = input.substr(0, i);
                // check if dpMap has the result for substr
                if (dpMap.find(substr) != dpMap.end())
                    result1 = dpMap[substr];
                else
                    result1 = computeWithDP(substr, dpMap);

                substr = input.substr(i + 1);
                if (dpMap.find(substr) != dpMap.end())
                    result2 = dpMap[substr];
                else
                    result2 = computeWithDP(substr, dpMap);

                for (auto n1 : result1) {
                    for (auto n2 : result2) {
                        if (cur == '+')
                            result.push_back(n1 + n2);
                        else if (cur == '-')
                            result.push_back(n1 - n2);
                        else
                            result.push_back(n1 * n2);
                    }
                }
            }
        }
        // if the input string contains only number
        if (result.empty())
            result.push_back(atoi(input.c_str()));
        // save to dpMap
        dpMap[input] = result;
        return result;
    }
};
```

written by Gcdofree original link here

Solution 3

Just doing it...

Solution 1 ... 48 ms

```
def diffWaysToCompute(self, input):
   tokens = re.split('(\D)', input)
   nums = map(int, tokens[::2])
   ops = map({'+': operator.add, '-': operator.sub, '*': operator.mul}.get, token
s[1::2])
   def build(lo, hi):
       if lo == hi:
           return [nums[lo]]
       return [ops[i](a, b)
               for i in xrange(lo, hi)
               for a in build(lo, i)
               for b in build(i + 1, hi)]
   return build(0, len(nums) - 1)
```

Solution 2 ... 168 ms

One-liner inspired by Soba.

```
def diffWaysToCompute(self, input):
    return [eval(`a`+c+`b`)
            for i, c in enumerate(input) if c in '+-*'
            for a in self.diffWaysToCompute(input[:i])
            for b in self.diffWaysToCompute(input[i+1:])] or [int(input)]
```

Solution 3 ... 64 ms

Faster version of solution 2.

```
def diffWaysToCompute(self, input):
    return [a+b if c == '+' else a-b if c == '-' else a*b
            for i, c in enumerate(input) if c in '+-*'
            for a in self.diffWaysToCompute(input[:i])
            for b in self.diffWaysToCompute(input[i+1:])] or [int(input)]
```

Solution 4 ... 188 ms

A code golf version of solution 2.

```
diffWaysToCompute=d=lambda s,t:[eval(`a`+c+`b`)for i,c in enumerate(t)if
c<'0'for a in s.d(t[:i])for b in s.d(t[i+1:])]or[int(t)]
```

C++ ... 8 ms

C++ version of solution 3.

```
vector<int> diffWaysToCompute(string input) {
    vector<int> output;
    for (int i=0; i<input.size(); i++) {
        char c = input[i];
        if (ispunct(c))
            for (int a : diffWaysToCompute(input.substr(0, i)))
                for (int b : diffWaysToCompute(input.substr(i+1)))
                    output.push_back(c=='+' ? a+b : c=='-' ? a-b : a*b);
    }
    return output.size() ? output : vector<int>{stoi(input)};
}
```

written by StefanPochmann original link here