<https://leetcode.com/problems/expression-add-operators/>

Given a string num that contains only digits and an integer target, return ***all possibilities*** *to insert the binary operators* '+'*,* '-'*, and/or* '\*' *between the digits of* num *so that the resultant expression evaluates to the* target *value*.

Note that operands in the returned expressions **should not** contain leading zeros.

**Example 1:**

Input: num = "123", target = 6

Output: ["1\*2\*3","1+2+3"]

Explanation: Both "1\*2\*3" and "1+2+3" evaluate to 6.

**Example 2:**

Input: num = "232", target = 8

Output: ["2\*3+2","2+3\*2"]

Explanation: Both "2\*3+2" and "2+3\*2" evaluate to 8.

**Example 3:**

Input: num = "3456237490", target = 9191

Output: []

Explanation: There are no expressions that can be created from "3456237490" to evaluate to 9191.

**Constraints:**

* 1 <= num.length <= 10
* num consists of only digits.
* -231 <= target <= 231 - 1

**Attempt 1: 2022-12-20**

**Solution 1:  Backtracking (120 min)**

class Solution {

public List<String> addOperators(String num, int target) {

List<String> result = new ArrayList<String>();

helper(num, 0, target, 0, 0, result, new StringBuilder());

return result;

}

private void helper(String num, int index, int target, long prev, long multi, List<String> result, StringBuilder sb) {

if(index == num.length() && prev == target) {

result.add(sb.toString());

return;

}

for(int i = index; i < num.length(); i++) {

// Corner case: if current position is 0, we can only use it as a single

// digit number, should be 0, if it is not a single digit number with

// leading 0, it should be considered as an invalid number, in simple, any

// number start with '0' is invalid except single digit '0', hence break out

// e.g

// Valid answers are:

// "105", 5 -> ["1\*0+5","10-5"]

// "00", 0 -> ["0+0", "0-0", "0\*0"]

// but with leading zeros we will have:

// "105", 5 -> ["1\*0+5","10-5", "1\*05"] -> "05" invalid

// "00", 0 -> ["0+0", "0-0", "0\*0", "00"] -> "00" invalid

if(num.charAt(index) == '0' && i != index) {

break;

}

long curr = Long.parseLong(num.substring(index, i + 1));

int len = sb.length();

// position 0 should be considered individually, since it does not

// have any operand character before current num

if(index == 0) {

helper(num, i + 1, target, curr, curr, result, sb.append(curr));

sb.setLength(len);

} else {

// Case '+'

helper(num, i + 1, target, prev + curr, curr, result, sb.append("+").append(curr));

sb.setLength(len);

// Case '-'

helper(num, i + 1, target, prev - curr, -curr, result, sb.append("-").append(curr));

sb.setLength(len);

// Case '\*'

// e.g. if you have a sequence of 12345 and you have proceeded to 1 + 2 + 3,

// now your eval is 6 right? If you want to add a \* between 3 and 4, you would

// take 3 as the digit to be multiplied, so you want to take it out from the

// existing eval. You have 1 + 2 + 3 \* 4 and the eval now is

// (1 + 2 + 3) - 3 + (3 \* 4). Hope this could help : )

helper(num, i + 1, target, prev - multi + multi \* curr, multi \* curr, result, sb.append("\*").append(curr));

sb.setLength(len);

}

}

}

}

Time Complexity : O(N \* 4^(N - 1))

we can choose + , - , \* and empty space (in this case, eg. 12 was treated as one number), in total four kinds of choices. Also, remember there is a for loop inside every call stack, So, for the call stack of length N,

Total time complexity should be N \* 4^(N - 1)

Space Complexity : O(N)