Expression Add Operators

Given a string that contains only digits `0-9` and a target value, return all possibilities to add **binary** operators (not unary) `+`, `-`, or `*` between the digits so they evaluate to the target value.

Examples:

```
"123", 6 -> ["1+2+3", "1*2*3"]
"232", 8 -> ["2*3+2", "2+3*2"]
"105", 5 -> ["1*0+5","10-5"]
"00", 0 -> ["0+0", "0-0", "0*0"]
"3456237490", 9191 -> []
```

Credits:
Special thanks to @davidtan1890 for adding this problem and creating all test cases.

Solution 1

This problem has a lot of edge cases to be considered:

1. overflow: we use a long type once it is larger than Integer.MAX_VALUE or minimum, we get over it.
2. 0 sequence: because we can't have numbers with multiple digits started with zero, we have to deal with it too.
3. a little trick is that we should save the value that is to be multiplied in the next recursion.

```
public class Solution {
    public List<String> addOperators(String num, int target) {
        List<String> rst = new ArrayList<String>();
        if(num == null || num.length() == 0) return rst;
        helper(rst, "", num, target, 0, 0, 0);
        return rst;
    }
    public void helper(List<String> rst, String path, String num, int target, int
pos, long eval, long multed){
        if(pos == num.length()){
            if(target == eval)
                rst.add(path);
            return;
        }
        for(int i = pos; i < num.length(); i++){
            if(i != pos && num.charAt(pos) == '0') break;
            long cur = Long.parseLong(num.substring(pos, i + 1));
            if(pos == 0){
                helper(rst, path + cur, num, target, i + 1, cur, cur);
            }
            else{
                helper(rst, path + "+" + cur, num, target, i + 1, eval + cur , cu
r);

                helper(rst, path + "-" + cur, num, target, i + 1, eval -cur, -cur
);

                helper(rst, path + "*" + cur, num, target, i + 1, eval - multed +
multed * cur, multed * cur );
            }
        }
    }
}
```

written by czonzhu original link here

Solution 2

```
class Solution {
private:
    // cur: {string} expression generated so far.
    // pos: {int}    current visiting position of num.
    // cv:  {long}   cumulative value so far.
    // pv:  {long}   previous operand value.
    // op:  {char}   previous operator used.
    void dfs(std::vector<string>& res, const string& num, const int target, strin
g cur, int pos, const long cv, const long pv, const char op) {
        if (pos == num.size() && cv == target) {
            res.push_back(cur);
        } else {
            for (int i=pos+1; i<=num.size(); i++) {
                string t = num.substr(pos, i-pos);
                long now = stol(t);
                if (to_string(now).size() != t.size()) continue;
                dfs(res, num, target, cur+'+'+t, i, cv+now, now, '+');
                dfs(res, num, target, cur+'-'+t, i, cv-now, now, '-');
                dfs(res, num, target, cur+'*'+t, i, (op == '-') ? cv+pv - pv*now
: ((op == '+') ? cv-pv + pv*now : pv*now), pv*now, op);
            }
        }
    }

public:
    vector<string> addOperators(string num, int target) {
        vector<string> res;
        if (num.empty()) return res;
        for (int i=1; i<=num.size(); i++) {
            string s = num.substr(0, i);
            long cur = stol(s);
            if (to_string(cur).size() != s.size()) continue;
            dfs(res, num, target, s, i, cur, cur, '#');         // no operator de
fined.
        }

        return res;
    }
};
```

written by cmyzx original link here

Solution 3

```
void addOperators(vector<string>& result, string nums, string t, long long last,
long long curVal, int target) {
    if (nums.length() == 0) {
        if (curVal == target)
            result.push_back(t);
        return;
    }

    for (int i = 1; i<=nums.length(); i++) {
        string num = nums.substr(0, i);
        if(num.length() > 1 && num[0] == '0')
            return;

        string nextNum = nums.substr(i);

        if (t.length() > 0) {
            addOperators(result, nextNum, t + "+" + num, stoll(num), curVal + sto
ll(num), target);
            addOperators(result, nextNum, t + "-" + num, -stoll(num), curVal - st
oll(num), target);
            addOperators(result, nextNum, t + "*" + num, last * stoll(num), (curV
al - last) + (last * stoll(num)), target);
        }
        else
            addOperators(result, nextNum, num, stoll(num), stoll(num), target);
    }
}

vector<string> addOperators(string num, int target) {
    vector<string> result;
    addOperators(result, num, "", 0, 0, target);
    return result;
}
```

written by jaewoo original link here