241. Different Ways to Add Parentheses

Given a string `expression` of numbers and operators, return *all possible results from computing all the different possible ways to group numbers and operators*. You may return the answer in **any order**.

The test cases are generated such that the output values fit in a 32-bit integer and the number of different results does not exceed `10^4`.

Example 1:

```
Input: expression = "2-1-1"
Output: [0,2]
Explanation:
((2-1)-1) = 0
(2-(1-1)) = 2
```

Example 2:

```
Input: expression = "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
```

Constraints:

- `1 <= expression.length <= 20`
- `expression` consists of digits and the operator `'+'`, `'-'`, and `'*'`.
- All the integer values in the input expression are in the range `[0, 99]`.
- The integer values in the input expression do not have a leading `'-'` or `'+'` denoting the sign.

241. Different Ways to Add Parentheses

```
/*
   Recursivity
   Time complexity: O(n.2^n)
   Space complexity: O(2^n)
*/
typedef std::vector<int> vi;
class Solution {
   public:
      // dispatch table
      std::unordered_map<char , std::function<int(int,int)> > operations{
         {'+',[](int a, int b){ return a + b;} },
         {'-',[](int a, int b){ return a - b;} },
         {'*',[](int a, int b){ return a * b;} } };
   public:
      vi diffWaysToCompute(std::string expression){
         // Base case
         if(std::all_of(expression.begin(),expression.end(),::isdigit)){
            return {std::stoi(expression)};
         }
         int n=expression.size();
         vi ans;
         // Divide...
         for(int i=0;i<n;++i){
            char c=expression[i];
            if(operations.find(c)!=operations.end()){
               //... and conquer
               vi left=diffWaysToCompute(expression.substr(0,i));
               vi right=diffWaysToCompute(expression.substr(i+1,n-1-i));
               // Combine
               for(auto& x: left){
                  for(auto& y: right){
                     ans.push_back(operations[c](x,y));
                  }
               }
            }
         }
        return ans;
      }
};
```