592. Fraction Addition and Subtraction

Given a string `expression` representing an expression of fraction addition and subtraction, return the calculation result in string format.

The final result should be an irreducible fraction. If your final result is an integer, change it to the format of a fraction that has a denominator `1`. So in this case, `2` should be converted to `2/1`.

Example 1:

```
Input: expression = "-1/2+1/2"
Output: "0/1"
```

Example 2:

```
Input: expression = "-1/2+1/2+1/3"
Output: "1/3"
```

Example 3:

```
Input: expression = "1/3-1/2"
Output: "-1/6"
```

Constraints:

- The input string only contains `'0'` to `'9'`, `'/'`, `'+'` and `'-'`. So does the output.
- Each fraction (input and output) has the format `±numerator/denominator`. If the first input fraction or the output is positive, then `'+'` will be omitted.
- The input only contains valid **irreducible fractions**, where the **numerator** and **denominator** of each fraction will always be in the range `[1, 10]`. If the denominator is `1`, it means this fraction is actually an integer in a fraction format defined above.
- The number of given fractions will be in the range `[1, 10]`.
- The numerator and denominator of the **final result** are guaranteed to be valid and in the range of **32-bit** int.

592. Fraction Addition and Subtraction

```
/*
   Intuitive approach:  String manipulation+Math+implementation
   Time complexity: O(n+n+m+m+m)=O(2n+3m)=O(n)
   Space complexity: O(m+m+2m)=O(m)
   where:  n is the length of the given expression
           m the number of fractions in the given expression
           m<n
*/

typedef std::vector<int> vi;
typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution {
   public:
      /*
         Return an array of integers containing all denominators
         extracted from the given expression.
         Time Complexity: O(n)
         Space complexity: O(m)
         where:  n is the length of the given expression
                 m the number of fractions in the given expression
                 m<n
      */
      vi get_denominators(std::string& exp){
         vi res;
         int n=exp.size();
         int i=0;
         while(i<n){
            char c=exp[i];
            if(c=='/'){
               int j=i+1;
               std::string number_str="";
               while(j<n&&std::isdigit(exp[j])){
                  number_str+=exp[j];
                  j++;
               }
               if(number_str!="") res.push_back(stoi(number_str));
               i=j;
            }
            else i++;
         }
         return res;
      }
```

```
/*
    Return an array of integers containing all numerators
    extracted from the given expression.
    Time Complexity: O(n)
    Space complexity: O(m)
    where:  n is the length of the given expression
            m the number of fractions in the given expression
            m<n
  */
  vi get_numerators(std::string& exp){
    int n=exp.size();
    vi res;
    int i=0;
    while(i<n){
      char c=exp[i];
      if(c=='/'){
        int j=i-1;
        std::string number_str="";
        while(j>=0 && std::isdigit(exp[j])){
          number_str=exp[j]+number_str;
          j--;
        }
        if(j==0 && exp[j]=='-') number_str='-'+number_str;
        if(j-1>=0 && (exp[j-1]=='+' || exp[j-1]=='-')) number_str='-'+number_str;
        if(j-1>=0 && (exp[j]=='-' && std::isdigit(exp[j-1]))) number_str='-'+number_str;
        if(number_str!="") res.push_back(stoi(number_str));
      }
      i++;
    }
    return res;
  }
```

```
    /*
        Fractions addition

    xp/xq + yp/yq = (xp . yq + yp . xq) / (xq . yq)   to reduce it   (xp . yq + yp . xq) / gcd(xp . yq + yp . xq, xq . yq)
                                                                     ---------------------------------------------------
                                                                                  xq . yq
                                                                     ---------------------------------------------------
                                                                     gcd(xp . yq + yp . xq, xq . yq)
    */
    ii add(ii& x, ii& y){
        auto [xp, xq]=x;
        auto [yp, yq]=y;
        long long q=xq*yq;
        long long p=xp*yq+xq*yp;
        long long g=std::gcd(abs(p), abs(q));
        return {p/g, q/g};
    }

    std::string fractionAddition(std::string expression){
        vi denominators=get_denominators(expression);

        vi numerators=get_numerators(expression);

        int m=numerators.size();
        vii fractions(m);

        for(int i=0;i<m;++i){
            fractions[i]={numerators[i],denominators[i]};
        }

        ii ans={0,1};
        for(auto& f: fractions){
            ans=add(ans,f);
        }
        std::string s=to_string(ans.first)+"/"+to_string(ans.second);
        return s;
    }
};
```

592. Fraction Addition and Subtraction

```
/*
   C++ convert using stringstream
   Time complexity: O(n+n+m+m+m)=O(2n+3m)=O(n)
   Space complexity: O(m)
   where:  n is the length of the given expression
           m the number of fractions in the given expression
            m<n
*/

typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution {
   public:
      vii convert(string& expression) {
         vii fractions;
         std::stringstream ss(expression);
         char op;
         int num, denom;

         while (ss>>num>>op>>denom) {
            fractions.emplace_back(num, denom);
         }

         return fractions;
      }
```

```
      ii add(ii& x, ii& y){
         auto [xp, xq]=x;
         auto [yp, yq]=y;
         long long q=xq*yq;
         long long p=xp*yq+xq*yp;
         long long g=gcd(p, q);
         return {p/g, q/g};
      }
```

```
std::string fractionAddition(std::string expression){
        vii fractions=convert(expression);

        int m=fractions.size();

        ii ans={0,1};
        for(auto& f: fractions){
           ans=add(ans,f);
        }

        string s=to_string(ans.first)+"/"+to_string(ans.second);

        return s;
      }

};
```

592. Fraction Addition and Subtraction

```
/*
   C++ convert using regex
   Time complexity: O(2^x+n+m)
   Space complexity: O(m)
   where:  x is the size of the regex
           n is the length of the given expression
           m the number of fractions in the given expression
            m<n
*/

typedef std::pair<int,int> ii;
typedef std::vector<ii> vii;

class Solution {
   public:
      vii convert(string& expression) {
         vii fractions;
         std::regex regex("([+-]?\\d+)/(\\d+)");
         smatch matches;
         while (std::regex_search (expression,matches,regex)){
            int num=stoi(matches[1]),denom=stoi(matches[2]);
            fractions.push_back({num,denom});
            expression=matches.suffix().str();
         }
         return fractions;
      }

      ii add(ii& x, ii& y){
         auto [xp, xq]=x;
         auto [yp, yq]=y;
         long long q=xq*yq;
         long long p=xp*yq+xq*yp;
         long long g=gcd(p, q);
         return {p/g, q/g};
      }
```

```
std::string fractionAddition(std::string expression){
        vii fractions=convert(expression);
        int m=fractions.size();
        ii ans={0,1};
        for(auto& f: fractions){
            ans=add(ans,f);
        }
        string s=to_string(ans.first)+"/"+to_string(ans.second);
        return s;
    }

};
```