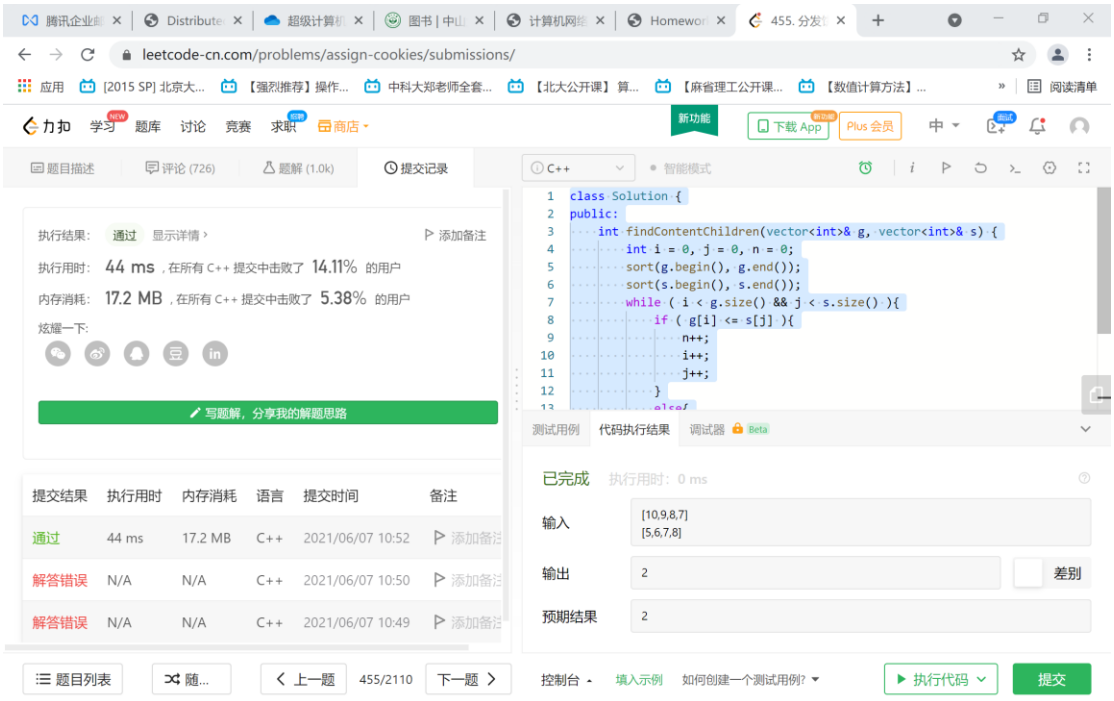
1. 算法思路：排序+贪心算法

   复杂度分析：O(NlogN) + O(N) = O(NlogN)

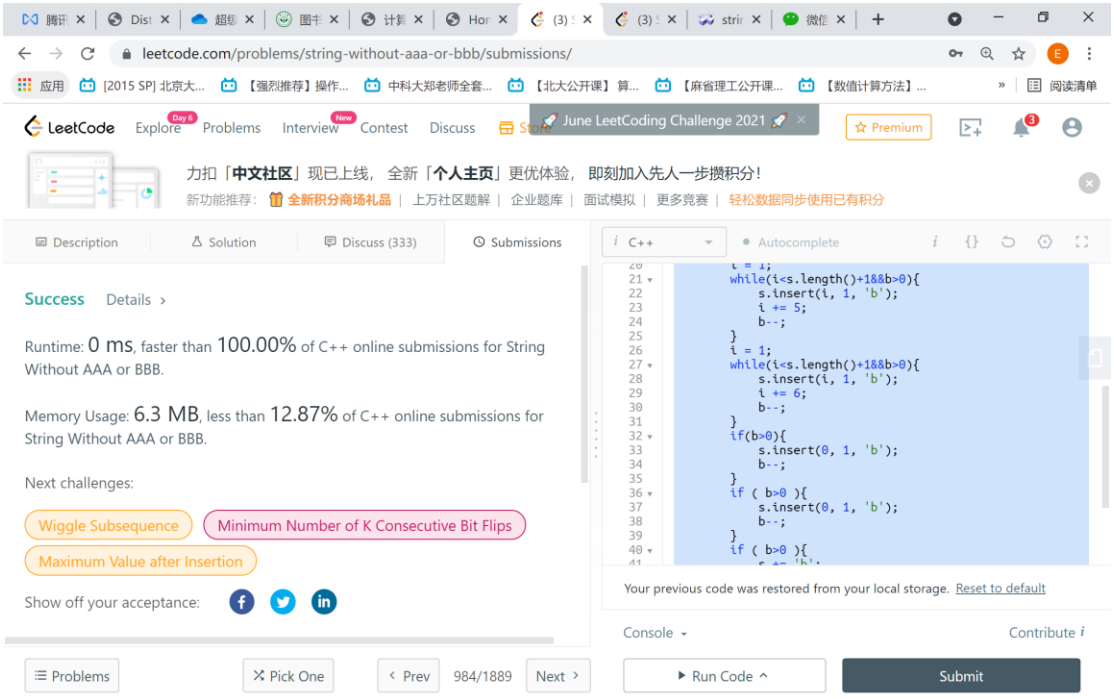
```
class Solution {
public:
    int findContentChildren(vector<int>& g, vector<int>& s) {
        int i = 0, j = 0, n = 0;
        sort(g.begin(), g.end());
        sort(s.begin(), s.end());
        while ( i < g.size() && j < s.size() ){
            if ( g[i] <= s[j] ){
                n++;
                i++;
                j++;
            }
            else{
                j++;
            }
        }
        return n;
    }
};
```



2. 算法思路：多轮循环，在 a 之间插入 b
   复杂度分析：O(N)

```
class Solution {
    public:
        string strWithout3a3b(int a, int b) {
            string s;
            s.insert(0, a, 'a');
            int i = 2;
            while(i<s.length()+1&&b>0){
                s.insert(i, 1, 'b');
                i+=3;
                b--;
            }
            //cout << b << endl;
            //cout << s << endl;
            i = 2;
            while(i<s.length()+1&&b>0){
                s.insert(i, 1, 'b');
                i += 4;
                b--;
            }
            i = 1;
            while(i<s.length()+1&&b>0){
                s.insert(i, 1, 'b');
                i += 5;
                b--;
            }
            i = 1;
            while(i<s.length()+1&&b>0){
                s.insert(i, 1, 'b');
                i += 6;
                b--;
            }
            if(b>0){
                s.insert(0, 1, 'b');
                b--;
            }
            if ( b>0 ){
                s.insert(0, 1, 'b');
                b--;
            }
            if ( b>0 ){
                s += 'b';
                b--;
```

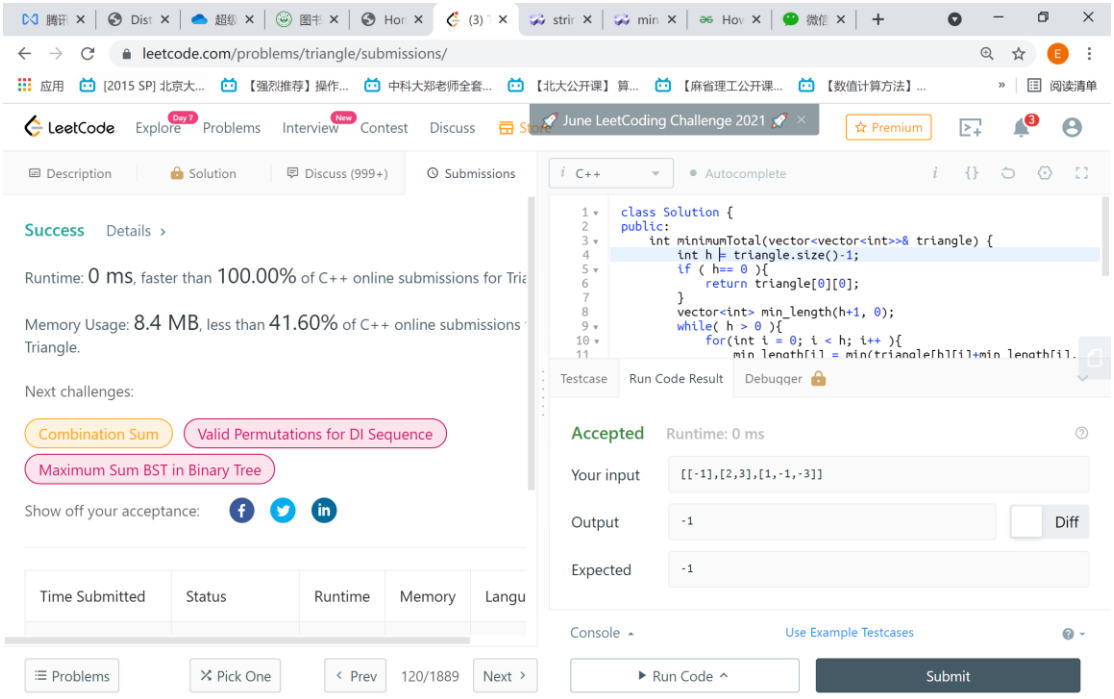
```
            }
            if ( b > 0 ){
                s+='b';
            }
            return s;
        }
};
```



3. 算法思路：从底层往上搜索，每一层逐步缩小数组
   复杂度分析：时间复杂度 O(N^2) (N 为层数)
   空间复杂度 O(N) （只需要一个额外的 O(N)数组）

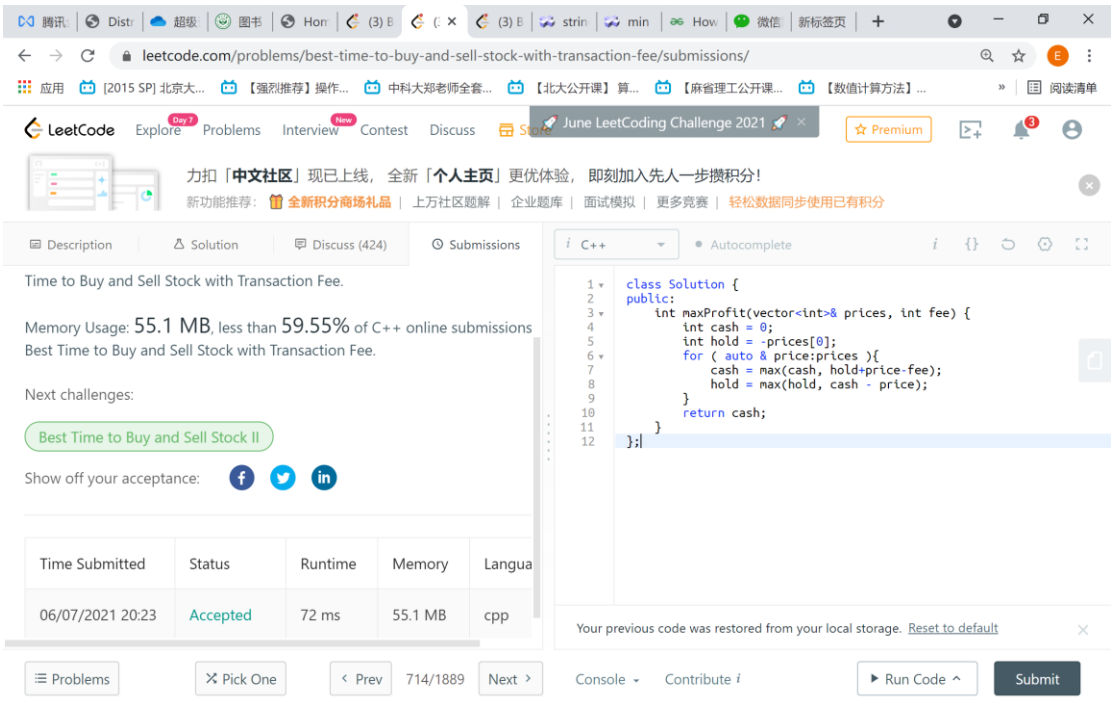
```
class Solution {
    public:
        int minimumTotal(vector<vector<int>>& triangle) {
            int h = triangle.size()-1;
            if ( h== 0 ){
                return triangle[0][0];
            }
            vector<int> min_length(h+1, 0);
            while( h > 0 ){
                for(int i = 0; i < h; i++ ){
                    min_length[i] = min(triangle[h][i]+min_length[i],
                                        triangle[h][i+1]+min_length[i+1]);

                }
                min_length.pop_back();
```

```
                h--;
            }
            return min_length[0]+triangle[0][0];
        }
};
```



4. 算法思路：动态规划，对于每一天，计算出当天能有的最大现金数和若持有股票的最大收益，最后输出最后一天能有的最大现金数
   复杂度：遍历每一天，O(N)

```
class Solution {
    public:
        int maxProfit(vector<int>& prices, int fee) {
            int cash = 0;
            int hold = -prices[0];
            for ( auto & price:prices ){
                cash = max(cash, hold+price-fee);
                hold = max(hold, cash - price);
            }
            return cash;
        }
};
```



5. 算法思路：动态规划，遍历字符串的每个字符并更新编码方案数
复杂度分析：遍历字符串，O(N)

```
class Solution {
    public:
        int numDecodings(string s) {
            if ( s[0] == '0' ){
                return 0;
            }
            int n1 = 0, n2 = 1, hold;
            for ( int i = 1; i < s.length(); i++ ){
                if ( s[i] == '0' ){
                    if ( s[i-1] !='1'&& s[i-1]!='2' ){
                        return 0;
                    }
                    hold = n1;
                    n1 = n2;
                    n2 = hold == 0? 1:hold;
                }
                else{
                    string s1;
                    int n = stoi(s1 + s[i-1] + s[i]);
                    if ( n >=11 && n <= 26 ){
                        hold = n2;
                        n2 = n1 == 0? 1+n2:n1+n2;
                        n1 = hold;
                    }
```

```
                    else{
                        n1 = n2;
                    }
                }
            }
            return n2;
        }
};
```