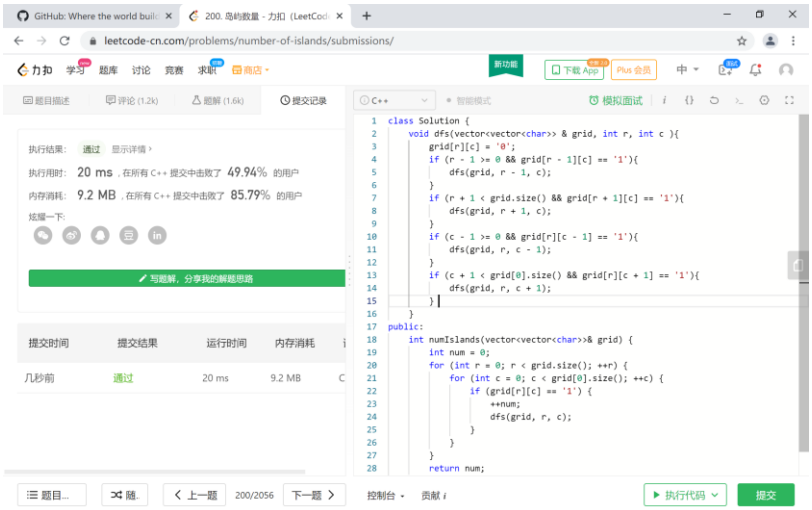
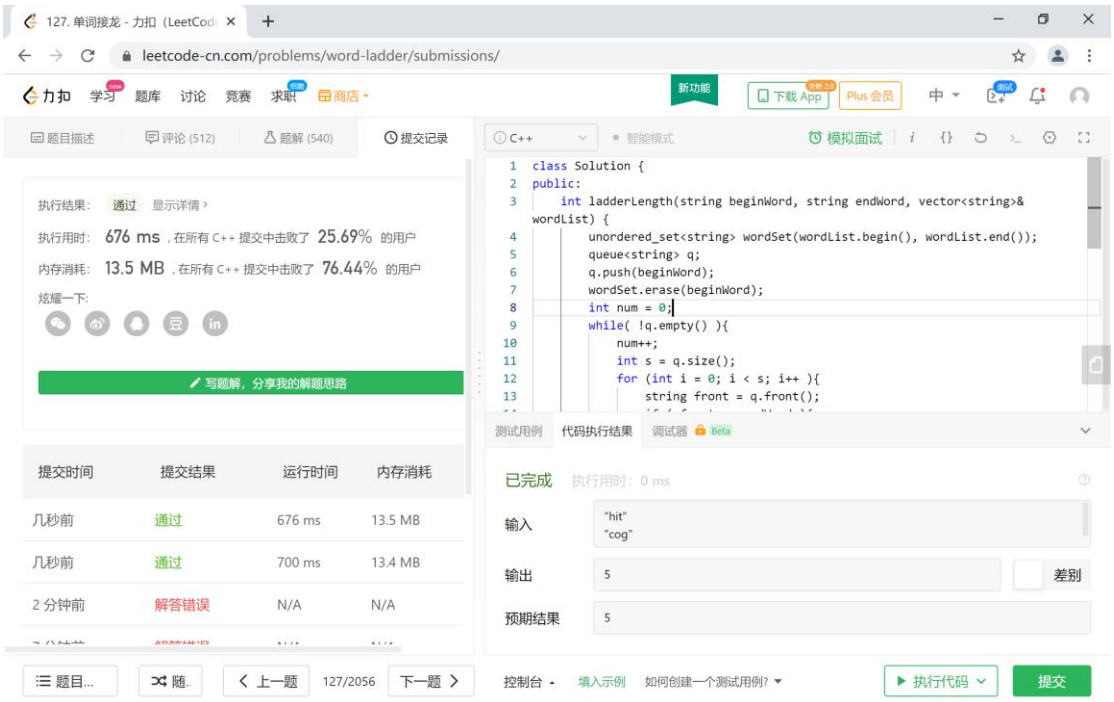
1. 算法思路：深度优先搜索，对于陆地上的点，搜索其相邻的陆地节点，并置 0
   复杂度分析：$O(MN)$，其中 MM 和 NN 分别为行数和列数

```
class Solution {
    void dfs(vector<vector<char>> & grid, int r, int c ){
        grid[r][c] = '0';
        if (r - 1 >= 0 && grid[r - 1][c] == '1'){
            dfs(grid, r - 1, c);
        }
        if (r + 1 < grid.size() && grid[r + 1][c] == '1'){
            dfs(grid, r + 1, c);
        }
        if (c - 1 >= 0 && grid[r][c - 1] == '1'){
            dfs(grid, r, c - 1);
        }
        if (c + 1 < grid[0].size() && grid[r][c + 1] == '1'){
            dfs(grid, r, c + 1);
        }
    }
public:
    int numIslands(vector<vector<char>>& grid) {
        int num = 0;
        for (int r = 0; r < grid.size(); ++r) {
            for (int c = 0; c < grid[0].size(); ++c) {
                if (grid[r][c] == '1') {
                    ++num;
                    dfs(grid, r, c);
                }
            }
        }
        return num;
    }
};
```



2. 算法思路：广度优先搜索，使用队列存储
复杂度分析：$O(N^2 \times C)$，其中 N 为字符串个数，C 为字符串长度

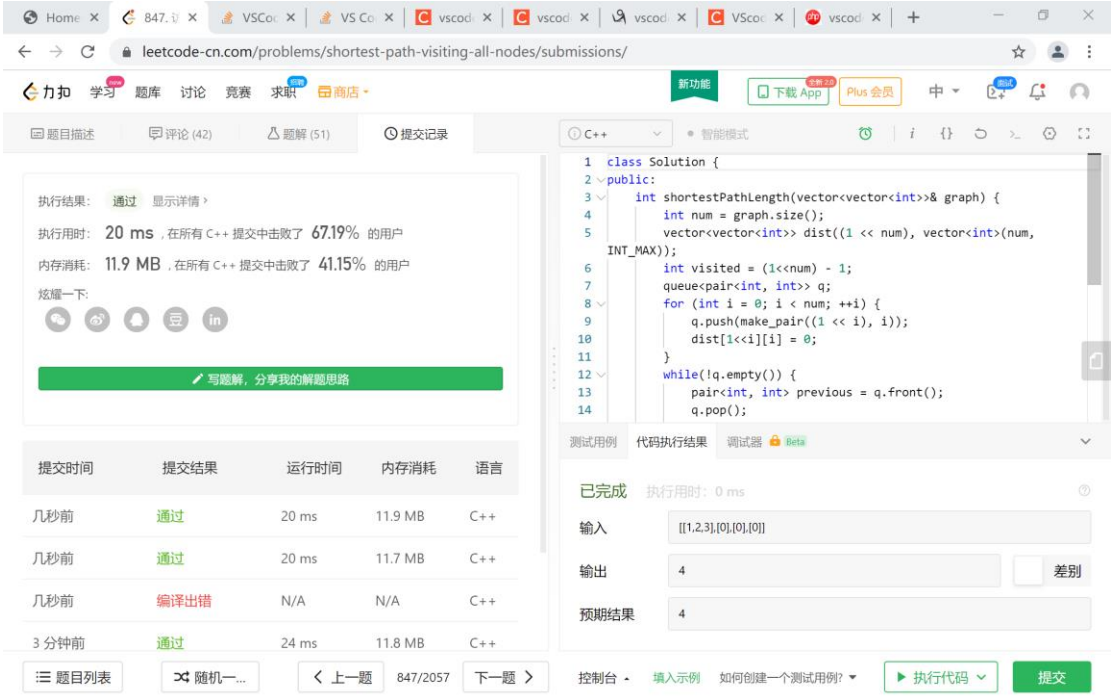
```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList
) {
        unordered_set<string> wordSet(wordList.begin(), wordList.end());
        queue<string> q;
        q.push(beginWord);
        wordSet.erase(beginWord);
        int num = 0;
        while( !q.empty() ){
            num++;
            int s = q.size();
            for (int i = 0; i < s; i++ ){
                string front = q.front();
                if ( front == endWord ){
                    return num;
                }
                q.pop();
                for ( auto it = wordSet.begin(); it != wordSet.end(); ){
                    string word = *it;
                    int d = 0;
                    for (int j = 0; j < word.length(); j++ ){
                        if ( word[j] != front[j] ){
                            d++;
                        }
                        if ( d > 1){
                            break;
                        }
                    }
                    it++;
                    if ( d <= 1 ){
                        q.push(word);
                        wordSet.erase(word);
                    }
                }
            }
        }
        return 0;
    }
};
```



3. 算法思路：广度优先搜索，使用 bitmask
   复杂度分析：$O(N \times 2^n)$，其中 N 为节点数量。

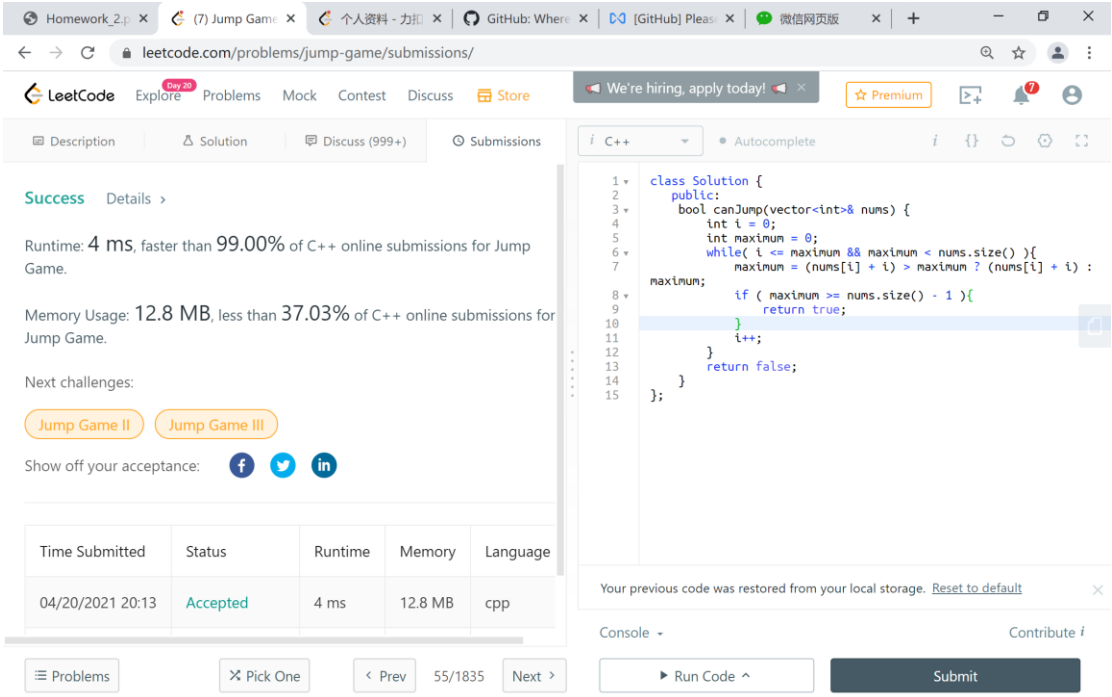
```
class Solution {
public:
    int shortestPathLength(vector<vector<int>>& graph) {
        int num = graph.size();
        vector<vector<int>> dist((1 << num), vector<int>(num, INT_MAX));
        int visited = (1<<num) - 1;
        queue<pair<int, int>> q;
        for (int i = 0; i < num; ++i) {
            q.push(make_pair((1 << i), i));
            dist[1<<i][i] = 0;
        }
        while(!q.empty()) {
            pair<int, int> previous = q.front();
            q.pop();
            int d = dist[previous.first][previous.second];
            if (previous.first == visited){
                return d;
            }
            for (auto & vertex : graph[previous.second]) {
                int v_new = previous.first |(1 << vertex);
                if (dist[v_new][vertex] > d + 1) {
                    dist[v_new][vertex] = d + 1;
                    q.push(make_pair(v_new, vertex));
```

```
                }
            }
        }
        return INT_MAX;
    }
};
```



4. 算法思路：贪心算法，每一步记录最大能到达的位置
   复杂度分析：O(N)

```
class Solution {
    public:
     bool canJump(vector<int>& nums) {
        int i = 0;
        int maximum = 0;
        while (i <= maximum && maximum < nums.size()) {
            maximum = (nums[i] + i) > maximum ? (nums[i] + i) : maximum;
            if (maximum >= nums.size() - 1) {
                return true;
            }
            i++;
        }
        return false;
    }
};
```



5. 算法思路：贪心算法，在每一步记录剩余油量的值
   复杂度：O(N)

```
class Solution {
   public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int start = 0;
        int current = 0;
        int tank = 0;
        bool flag = false;
        while (start - current != 1 && current - start != gas.size() - 1) {
            tank = tank + gas[current] - cost[current];  //gas in tank when arr
iving (i+1)th station
            current = (current + 1) % gas.size();
            if (current == 0) {
                flag = true;
            }
            if (tank < 0) {
                start = current;
                tank = 0;
                if (start == 0 || (flag && start >= current)) {
                    return -1;
                }
            }
        }
        if (tank + gas[current] - cost[current] < 0) {
```

```
            return -1;
        }
        return start;
    }
};
```