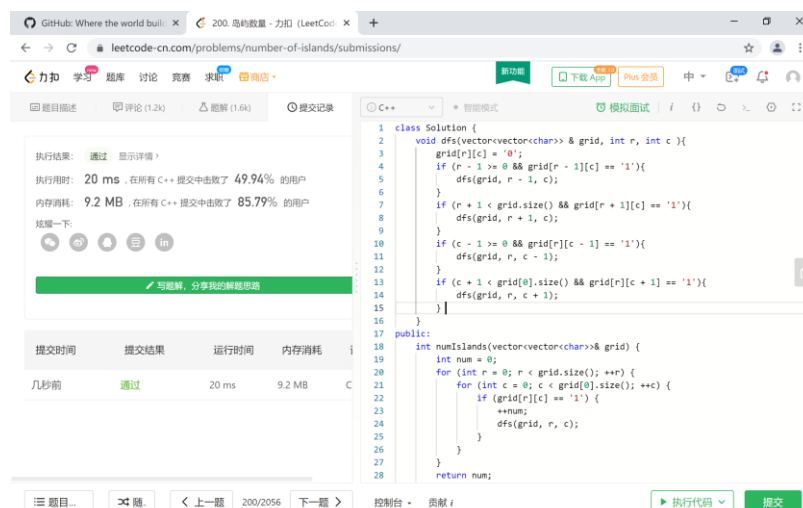


1. 算法思路：深度优先搜索，对于陆地上的点，搜索其相邻的陆地节点，并置 0
复杂度分析： $O(MN)$ ，其中 M 和 N 分别为行数和列数

```
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;
    }
};
```

127. 单词接龙 - 力扣 (LeetCode)

leetcode-cn.com/problems/word-ladder/submissions/

执行结果: 通过 显示详情

执行用时: 676 ms, 在所有 C++ 提交中击败了 25.69% 的用户

内存消耗: 13.5 MB, 在所有 C++ 提交中击败了 76.44% 的用户

炫耀一下:

写题解, 分享我的解题思路

提交时间	提交结果	运行时间	内存消耗
几秒前	通过	676 ms	13.5 MB
几秒前	通过	700 ms	13.4 MB
2 分钟前	解答错误	N/A	N/A

```

1 class Solution {
2 public:
3     int ladderLength(string beginWord, string endWord, vector<string>&
4 wordList) {
5         unordered_set<string> wordSet(wordList.begin(), wordList.end());
6         queue<string> q;
7         q.push(beginWord);
8         wordSet.erase(beginWord);
9         int num = 0;
10        while (!q.empty()) {
11            num++;
12            int s = q.size();
13            for (int i = 0; i < s; i++) {
14                string front = q.front();
15                // ...
16            }
17        }
18    }
19 };

```

测试用例 代码执行结果 调试器 Beta

已完成 执行用时: 0 ms

输入: "hit", "cog"

输出: 5

预期结果: 5

控制台 填入示例 如何创建一个测试用例?

执行代码 提交

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;
}
};

```

执行结果: 通过 显示详情

执行用时: 20 ms, 在所有 C++ 提交中击败了 67.19% 的用户

内存消耗: 11.9 MB, 在所有 C++ 提交中击败了 41.15% 的用户

炫耀一下:

写题解, 分享我的解题思路

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	20 ms	11.9 MB	C++
几秒前	通过	20 ms	11.7 MB	C++
几秒前	编译出错	N/A	N/A	C++
3分钟前	通过	24 ms	11.8 MB	C++

```

1 class Solution {
2 public:
3     int shortestPathLength(vector<vector<int>>& graph) {
4         int num = graph.size();
5         vector<vector<int>> dist(1 << num, vector<int>(num,
6             INT_MAX));
7         int visited = (1 << num) - 1;
8         queue<pair<int, int>> q;
9         for (int i = 0; i < num; ++i) {
10             q.push(make_pair(1 << i, 1));
11             dist[1 << i][i] = 0;
12         }
13         while(!q.empty()) {
14             pair<int, int> previous = q.front();
15             q.pop();

```

测试用例 代码执行结果 调试器 Beta

已完成 执行用时: 0 ms

输入: [[1,2,3],[0],[0],[0],[0]]

输出: 4

预期结果: 4

控制台 填入示例 如何创建一个测试用例? 执行代码 提交

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;
    }
};

```

Success Details >

Runtime: 4 ms, faster than 99.00% of C++ online submissions for Jump Game.

Memory Usage: 12.8 MB, less than 37.03% of C++ online submissions for Jump Game.

Next challenges: [Jump Game II](#) [Jump Game III](#)

Show off your acceptance: [f](#) [t](#) [in](#)

Time Submitted	Status	Runtime	Memory	Language
04/20/2021 20:13	Accepted	4 ms	12.8 MB	cpp

```

1 class Solution {
2     public:
3         bool canJump(vector<int>& nums) {
4             int i = 0;
5             int maximum = 0;
6             while( i <= maximum && maximum < nums.size() ){
7                 maximum = (nums[i] + i) > maximum ? (nums[i] + i) :
8                     maximum;
9                 if ( maximum >= nums.size() - 1 ){
10                     return true;
11                 }
12                 i++;
13             }
14             return false;
15 };

```

Your previous code was restored from your local storage. [Reset to default](#)

Console [Contribute i](#)

[Run Code ^](#) [Submit](#)

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 arriving (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;
}
};

```

Homework_2_1 | (7) Gas Station | (7) Gas Station | GitHub: Where | [GitHub] Pleas | 微信网页版 | + | - | X

leetcode.com/problems/gas-station/submissions/

LeetCode Explore **Day 30** Problems Mock Contest Discuss Store We're hiring, apply today! Premium

Description Solution Discuss (...) Submissi...




Success Details >

Runtime: 4 ms, faster than 85.78% of C++ online submissions.

Memory Usage: 9.9 MB, less than 42.94% of C++ online submissions.

Next challenges:

- Create Maximum Number
- Maximum Non Negative Product in a Matrix
- Number Of Rectangles That Can Form The Largest Square

Show off your acceptance:   

Time Submitted	Status	Runtime
134/1835		

```

1  class Solution {
2      public:
3      int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
4          int start = 0;
5          int current = 0;
6          int tank = 0;
7          bool flag = false;
8          while (start - current != 1 && current - start != gas.size() - 1){
9              tank = tank + gas[current] - cost[current]; //gas in tank when
10             arriving (i+1)th station
11             current = (current + 1) % gas.size();
12             if (current == 0){
13                 flag = true;
14             }
15             if (tank < 0){
16                 start = current;
17                 tank = 0;
18                 if (start == 0 || (flag && start >= current)){
19                     return -1;
20                 }
21             }
22             if (tank + gas[current] - cost[current] < 0){
23                 return -1;
24             }
25             return start;
26         }
27     };

```

Console | Contribute | Run Code | Submit