

1、

算法思路：

将二维数组视作一个无向图，两个 1 相邻视作两个相连的点。然后对这个数组进行一次遍历，采用深度优先搜索的方法，将遍历过的点设置为 0，那么最后所有的点都会被设置为 0，其中作为深度优先搜索的起始点有多少个就有多少个岛屿

复杂度分析：

将这个数组遍历一次，时间复杂度为  $O(n*m)$ ，空间复杂度为  $O(1)$ ，其中， $n*m$  为数组的大小

代码：

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid,int a,int b)
    {
        int width,length;
        width=grid.size();
        length=grid[0].size();
        grid[a][b]='0';
        //a-1,b
        if(a-1>=0&&grid[a-1][b]=='1')
        {
            dfs(grid,a-1,b);
        }
        //a,b-1
        if(b-1>=0&&grid[a][b-1]=='1')
        {
            dfs(grid,a,b-1);
        }
        //a+1,b
        if(a+1<width&&grid[a+1][b]=='1')
        {
            dfs(grid,a+1,b);
        }
        //a,b+1
        if(b+1<length&&grid[a][b+1]=='1')
        {
            dfs(grid,a,b+1);
        }
    }

    int numIslands(vector<vector<char>>& grid) {
        int island_number=0;
        int width,length;
        width=grid.size();
```

```

        if(width==0)
            return 0;
        length=grid[0].size();
        for(int i=0;i<width;i++)
        {
            for(int j=0;j<length;j++)
            {
                if(grid[i][j]=='1')
                {
                    island_number++;
                    dfs(grid,i,j);
                }
            }
        }
        return island_number;
    }
};

```

截图：

力扣 探索 题库 圈子 竞赛 面试 职位 商店

新功能 下载 App 会员中心 中 8 1

题目描述 评论 (819) 题解 (834) 提交记录

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 30.50% 的用户

内存消耗：8.6 MB，在所有 C++ 提交中击败了 100.00% 的用户

标签一下：

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	8.6 MB	C++
3 分钟前	执行出错	N/A	N/A	C++
25 分钟前	执行出错	N/A	N/A	C++

题目列表 随机一题 < 上一题 223 下一题 > 控制台 贡献

```

1 class Solution {
2 public:
3     void dfs(vector<vector<char>>& grid, int a, int b)
4     {
5         int width, length;
6         width = grid.size();
7         length = grid[0].size();
8         grid[a][b] = '0';
9         // a-1, b
10        if(a-1 >= 0 && grid[a-1][b] == '1')
11        {
12            dfs(grid, a-1, b);
13        }
14        // a, b-1
15        if(b-1 >= 0 && grid[a][b-1] == '1')
16        {
17            dfs(grid, a, b-1);
18        }
19        // a+1, b
20        if(a+1 < width && grid[a+1][b] == '1')
21        {
22            dfs(grid, a+1, b);
23        }
24        // a, b+1
25        if(b+1 < length && grid[a][b+1] == '1')

```

2、

算法思路：

我们将在数组中的所有单词视作一个点，然后如果两个单词之间只有一个字母的差别的话，就将其视作两点之间存在一条边。那么我们可以把问题简化成要求的两点之间最短路径的值是多少。

复杂度分析：

时间复杂度： $O(n)$ ，空间复杂度： $O(n)$ ， $n$  为数组的长度（也就是单词的个数）。

代码：

```

class Solution {
public:

```

```

    int ladderLength(string beginWord, string endWord, vector<string>&
wordList)    {
        //将所有节点加入 s 中
        unordered_set<string> s;
        for(auto &i: wordList)
            s.insert(i);

        queue<pair<string,int>> q;
        //beginWord 加进去

        q.push({beginWord,1});

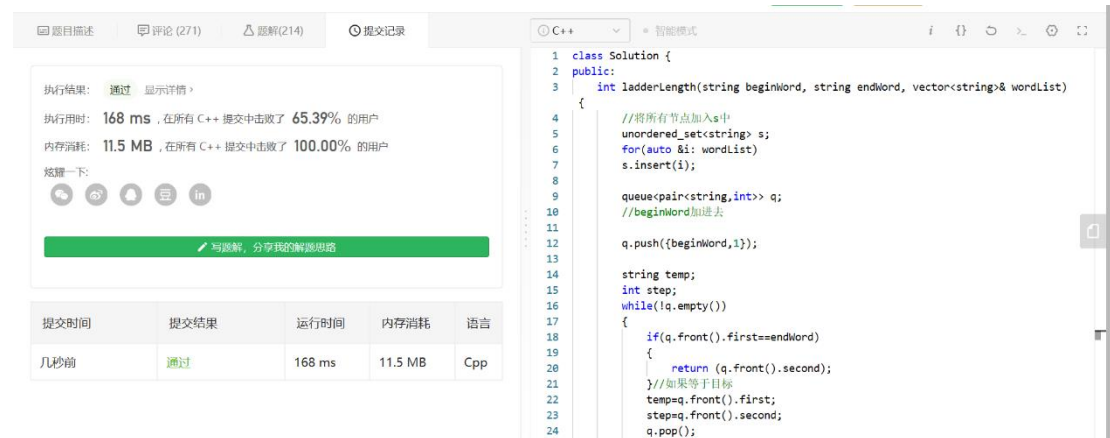
        string temp;
        int step;
        while(!q.empty())
        {
            if(q.front().first==endWord)
            {
                return (q.front().second);
            }//如果等于目标
            temp=q.front().first;
            step=q.front().second;
            q.pop();

            char ch;
            for(int i=0;i<temp.length();i++)
            {
                ch=temp[i];
                for(char c='a';c<='z';c++)
                {
                    if(ch==c)
                        continue;
                    temp[i]=c;
                    if(s.find(temp)!=s.end())
                    {
                        q.push({temp,step+1});
                        s.erase(temp);
                    }
                    temp[i]=ch;
                }
            }
        }
        return 0;
    }
}

```

```
};
```

截图：



3、

算法思路：

对每个点作为起始点进行 BFS，终点是所有的点都被遍历过了，然后在这里面求最短路问题，并且采用位压缩的方式存储节点是否被遍历的状态。

复杂度分析：

时间复杂度：  $O(2^N * N)$ 。

空间复杂度：  $O(2^N * N)$ 。

N 是节点个数。

代码：

```
struct state{
    int visited;    //第 i 位为 1 表示第 i 个节点已经访问
    int index;
    state(int v,int i){
        visited = v;
        index = i;
    }
};

class Solution {
public:
    int shortestPathLength(vector<vector<int>>& adj) {
        const int n = adj.size();
        const int end_state = (1<<n) - 1;    //表示遍历所有节点的状态

        queue<state> box;
        bool visited[1<<n][n];
        memset(visited,0,sizeof(bool)*(1<<n)*n);
```

```

    for(int i=0;i<n;i++){
        box.push(state(1<<i,i));//
        visited[1<<i][i] = true;
    }

    int len = 0;
    while(!box.empty()){
        int size = box.size(); //注意在遍历过程中 box 会发生变化
        for(int i=0;i<size;i++){
            state tmp = box.front();
            box.pop();

            for(int j=0;j<adj[tmp.index].size();j++){//BFS 的下一层
                int next = adj[tmp.index][j];
                int s = tmp.visited | (1<<next); //表示访问节点
next, 更新 state
                if(s==end_state){
                    return len+1;
                }
                if(!visited[s][next]){ //未访问过
                    visited[s][next] = true;
                    box.push(state(s,next));
                }
            }
        }
        len++;
    }
    return 0;
}
};

```

截图：

力扣 探索 题库 圈子 竞赛 面试 职位 商店

新功能 下载 App 会员中心 中

题目描述 评论 (23) 题解 (27) 提交记录

执行结果: **通过** 显示详情

执行用时: **8 ms**, 在所有 C++ 提交中击败了 **97.03%** 的用户

内存消耗: **8.5 MB**, 在所有 C++ 提交中击败了 **100.00%** 的用户

标签: 图论, 广度优先搜索, 深度优先搜索, 动态规划

写题解, 分享你的解题思路

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	8 ms	8.5 MB	Cpp

```

1 struct state{
2     int visited; //第i位为1表示第i个节点已经访问
3     int index;
4     state(int v,int i){
5         visited = v;
6         index = i;
7     }
8 };
9
10 class Solution {
11 public:
12     shortestPathLength(vector<vector<int>>& adj) {
13         const int n = adj.size();
14         const int end_state = (1<<n) - 1; //表示遍历所有节点的状态
15
16         queue<state> box;
17         bool visited[1<<n][n];
18         memset(visited,0,sizeof(bool)*(1<<n)*n);
19
20         for(int i=0;i<n;i++){
21             box.push(state(1<<i,i));
22             visited[1<<i][i] = true;
23         }
24
25         int len = 0;
26         while(!box.empty()){

```

4、

算法思路: 用一个 k 来代表当前可以达到的最远距离, 每次 i++ 时, 更新 k 的值, 如果遍历完整数组 i>k 均不成立那么就可以达到数组末尾, 否则无法到达。

复杂度分析:

时间复杂度:  $O(n)$ , 空间复杂度:  $O(1)$ , n 为数组大小。

代码:

```

class Solution {
public:
    bool canJump(vector<int>& nums) {
        int k=0;
        for(int i=0;i<nums.size();i++)
        {
            if(i>k)
                return false;
            k=max(k,i+nums[i]);
        }
        return true;
    }
};

```

截图:

力扣 探索 题库 圈子 竞赛 面试 职位 商店

新功能 下载 App 会员中心 中

题目描述 评论 (933) 题解 (1043) 提交记录

提交时间	提交结果	运行时间	内存消耗	语言
4 个月前	通过	16 ms	14.9 MB	Cpp
4 个月前	解答错误	N/A	N/A	C

```

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

```

5、

算法思路：

采用最笨的方法，每次从一个起点开始计算，看下能否在符合规则的情况下跑完全程，如果可以就退出，不然就继续遍历，直到遍历完所有加油站为止。

复杂度分析：

时间复杂度： $O(n^2)$ ，空间复杂度： $O(1)$ 。n 为 gas/cost 数组的大小。

代码：

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int n=gas.size();
        int pos=-1;
        for(int i=0;i<n;i++)
        {
            int sum=0;//记录耗油和加油总量
            for(int j=i;j<=n+i;j++)
            {
                if(j==i)
                {
                    pos=j;
                    sum+=gas[j%n];
                    continue;
                }
                if(sum<cost[(j-1)%n])
                {
                    pos=-1;
                    break;
                }
                else
                {
                    sum=sum+gas[j%n]-cost[(j-1)%n];
                }
                if(j==n+i)
                {
                    goto A;
                }
            }
        }
        A:
        return pos;
    }
};
```

截图：

题目描述

评论 (219)

题解 (227)

提交记录

执行结果: 通过 [显示详情](#)

执行用时: **328 ms** , 在所有 C++ 提交中击败了 **11.12%** 的用户

内存消耗: **9.6 MB** , 在所有 C++ 提交中击败了 **7.14%** 的用户

炫耀一下:

与题解, 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	328 ms	9.6 MB	Cpp
几秒前	通过	296 ms	9.6 MB	Cpp
几秒前	解答错误	N/A	N/A	Cpp

C++

智能模式

```
1 class Solution {
2 public:
3     int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
4         int n=gas.size();
5         int pos=-1;
6         for(int i=0;i<n;i++)
7         {
8             int sum=0;//记录耗油和加油总量
9             for(int j=i;j<=n+i;j++)
10             {
11                 if(j==1)
12                 {
13                     pos=j;
14                     sum+=gas[j%n];
15                     continue;
16                 }
17                 if(sum<cost[(j-1)%n])
18                 {
19                     pos=-1;
20                     break;
21                 }
22                 else
23                 {
24                     sum=sum+gas[j%n]-cost[(j-1)%n];
25                 }
26                 if(j==n+i)
27                     return pos;
28             }
29         }
30         return -1;
31     }
32 }
```