算法设计与应用基础 作业 2

19335174 施天予

1.【Leetcode200】岛屿数量

【算法思路】

这道题可以利用深度优先算法。首先，可以将二维网络看成无向图，相邻（水平或竖直）是 1 的可以看成有边相连。然后，扫描图。如果该位置是 '1'，则以该位置开始进行 DFS。注意在 DFS 过程中，每个遍历到的点都被重新改为 '0'。每次 DFS，都要递归进行 4 个方向的 DFS（在边界范围内）。最终的岛屿数就是进行的 DFS 的次数。

【复杂度分析】

时间复杂度：O(MN)，其中 M 和 N 分别为行数和列数。

空间复杂度：O(MN)，在最坏情况下，整个网格均为陆地，DFS 深度为 MN。

【代码】

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

```
                    dfs(grid, r, c);
                }
            }
        }
        return num;
    }
};
```

【Accepted 截图】

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	12 ms	9.3 MB	C++

2.【Leetcode127】单词接龙

【算法思路】

此题可用 BFS。在遍历单词的基础上遍历单词的每个字母，进行'a'-'z'的替换，看替换后的单词是否在原来的单词表里且未被访问过。如果在则将这个词加到队列中，同时把该节点移除掉。可以用 set 来储存，所以可以直接将已经考虑的单词从 set 中除去，而不需要引入额外的数据结构空间来标记 visit。

【复杂度分析】

时间复杂度：O(N)，N 为单词的总数。

空间复杂度：O(N)，N 为单词的总数。

【代码】

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<str
ing>& wordList) {
```

```
        unordered_set<string> s; //字符串的集合
        for (auto x : wordList)
            s.insert(x);
        queue<pair<string, int>> q; //用于广度优先搜索的队列
        q.push({beginWord, 1});
        while (!q.empty()) {
            if (q.front().first == endWord)
                return q.front().second;
            string tmp = q.front().first;
            int step = q.front().second;
            q.pop();
            for (int i = 0; i < tmp.length(); ++i) {
                char x = tmp[i];
                for (char c = 'a'; c <= 'z'; ++c){
                    if (x == c) continue;
                    tmp[i] = c ;
                    if (s.find(tmp) != s.end() ){
                        q.push({tmp, step+1});
                        s.erase(tmp) ;
                    }
                    tmp[i] = x;
                }
            }
        }
        return 0;
    }
};
```

【Accepted 截图】

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	116 ms	13.9 MB	C++

3.【Leetcode847】访问所有节点的最短路径

【算法思路】

本题可用多个节点同时开始 BFS，当有某一个节点的 BFS 到达终止条件时，该结果就是最短路径，而最短路径问题的难点之一在于如何标记已访问的节点。由于题目中有条件 “0<=N<=12” ，所以，总的状态数最多只有 2^12 种，也就是最多是 2^12 个状态，是可以用整型存下来的，所以可以使用位压缩，即使用一个 int visit 来表示已遍历的节点（包括之前遍历的节点+当前节点）。

然后，需要考虑的是如何储存当前节点和已访问节点的最短路径长度。可采用一个 vector<vector<int>> dist 数组，数组行是已访问的节点（位表示法），列是表示当前节点的索引，各单元内容为访问当前节点时的最短路径。采用队列来实现 BFS，队列中存的是 state。每次利用队头 pop 出的 state，更新 dist 数组。

【复杂度分析】

时间复杂度：O(2^N*N)，位运算表示状态一共有 2^N 个，而 cur 一共有 N 个。

空间复杂度：O(2^N*N)，对应状态数。

【代码】

```
class Solution {
public:
    struct state {
        int visit;  //当前节点+已访问节点(位运算符表示)
        int cur;    //当前节点
        state(int v, int c) {
            visit = v;
            cur = c;
        }
    };
    int shortestPathLength(vector<vector<int>>& graph) {
        int n = graph.size();
        if (n == 0) return 0;
        int num = 1 << n;
        vector<vector<int>> dist(1 << n, vector<int>(n, INT_MAX))
;
        int end = (1 << n) - 1;     //终止状态：用位运算表示
        queue<state> q;
        for (int i = 0; i < n; i++) {
            q.push(state(1 << i, i));
```

```
            dist[1 << i][i] = 0;
        }
        while (!q.empty()) {
            state pre = q.front();
            q.pop();
            int visit1 = pre.visit;
            int dist1 = dist[visit1][pre.cur];
            if (visit1 == end) return dist1;
            for (int node : graph[pre.cur]) {
                int visit_new = visit1 | (1 << node);    //新状态
                if (dist[visit_new][node] > dist1 + 1) {
                    dist[visit_new][node] = dist1 + 1;
                    q.push(state(visit_new, node));
                }
            }
        }
        return -1;
    }
};
```

【Accepted 截图】

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	16 ms	11.8 MB	C++

4.【Leetcode55】跳跃游戏

【算法思路】

此题可用贪心算法。对数组中的任意一个位置 y，能否跳跃到达 y，意味着是否

存在一个位置 x，有 x+nums[x]>=y。如果存在，则 y 可以到达。所以，可以在遍历每一个位置的时候，实时维护当前可以达到的最远的位置 k。如果在遍历到 x 的时候，k<x，则说明不能到达 x，返回 false；否则，说明可以到达 x，继续下一轮遍历。

【复杂度分析】

时间复杂度：O(n)，其中 n 为数组的大小（最多遍历 nums 数组一遍）。

空间复杂度：O(1)，不需要额外的空间开销。

【代码】

```
class Solution {
public:
    bool canJump(vector<int>& nums) {
        int n = nums.size();
        int k = 0;
        for (int i = 0; i < n - 1; ++i) {
            k = max(k, i + nums[i]);
            if (k <= i) return false;
        }
        return k >= n-1;
    }
};
```

【Accepted 截图】

提交时间	提交结果	运行时间	内存消耗	语言
几秒前	通过	12 ms	12.6 MB	C++

5.【Leetcode134】加油站

【算法思路】

此题可以用双指针求解。对于任意一个位置 x，他能否到达 x 意思是当前汽油剩余量 temp+gas[x]-cost[x]是否大于等于 0。然而，关键在于找到循环行使的起点。所以我设置了双指针，初始的时候，left=0，right=1 。当 left!=right 的时候，一直 while 循环。循环内判断当前汽油剩余量 temp 是否大于 0。是，则说明以 left 作为行使起点，可以继续往后走，right++；否则，说明以 left 作为行使起点，无法继续往后走，当前 left 不符合起点的条件，此时 left--，也就是检查看 left 之前的节点是否满足起点的条件，使得 temp>0 。

在开始可以用 stl 的 accumulate 函数计算可以加油的总量 gas_sum ，和消耗的总量 cost_sum，如果 cost_sum>gas_sum，直接返回 -1。

【复杂度分析】

时间复杂度：O(N)，其中 N 为数组的大小（数值只遍历 1 遍）。

空间复杂度：O(1)，只用到了常数个变量。

【代码】

```
class Solution {
public:
    int canCompleteCircuit(vector<int>& gas, vector<int>& cost) {
        int n = gas.size();
        if (n == 1) return gas[0] >= cost[0] ? 0 : -1;
        int gas_sum = accumulate(gas.begin(), gas.end(), 0);
        int cost_sum = accumulate(cost.begin(), cost.end(), 0);
        if (gas_sum < cost_sum) return -1;
        int left = 0, right = 1;
        int temp = gas[0] - cost[0];
        while (left != right) {
            if (temp >= 0) {
                temp += gas[right] - cost[right];
                right = right == n-1 ? 0 : right+1;
            }
            else {
                left = left == 0 ? n-1 : left-1;
                temp += gas[left] - cost[left];
            }
        }
        return left;
    }
};
```

【Accepted 截图】

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	9.6 MB	C++