Leetcode(c++c++) 广度优先搜索 简单题

994腐烂的橘子

题目：

在给定的网格中，每个单元格可以有以下三个值之一：

值 0 代表空单元格；

值 1 代表新鲜橘子；

值 2 代表腐烂的橘子。

每分钟，任何与腐烂的橘子（在 4 个正方向上）相邻的新鲜橘子都会腐烂。

返回直到单元格中没有新鲜橘子为止所必须经过的最小分钟数。如果不可能，返回 -1。

思路：

代码：

1. **class** Solution {
2. **public**:
3. **int** orangesRotting(vector<vector<**int**>>& grid) {
4. **int** xx[4]={0,0,-1,1};
5. **int** yy[4]={1,-1,0,0};
6. **int** n=grid.size();
7. **int** m=grid[0].size();
8. vector<vector<**int**>> new\_grid;
9. **int** ans=0;
10. **while**(1){
11. new\_grid=grid;
12. **for**(**int** i=0;i<n;i++){
13. **for**(**int** j=0;j<m;j++){
14. **if**(grid[i][j]==2){
15. **for**(**int** k=0;k<4;k++){
16. **int** x=i+xx[k];
17. **int** y=j+yy[k];
18. **if**(x>=0 && x<n && y>=0 &&y<m){
19. **if**(new\_grid[x][y]==1){
20. new\_grid[x][y]=2;
21. }
22. }
23. }
24. }
25. }
26. }
27. **if**(new\_grid==grid)
28. **break**;
29. ans++;
30. grid=new\_grid;
31. }
32. **for**(**int** i=0;i<n;i++){
33. **for**(**int** j=0;j<m;j++){
34. **if**(grid[i][j]==1)
35. **return** -1;
36. }
37. }
38. **return** ans;
39. }
40. };

中等题：

127单词接龙

题目：

给定两个单词（beginWord 和 endWord）和一个字典，找到从 beginWord 到 endWord 的最短转换序列的长度。转换需遵循如下规则：

每次转换只能改变一个字母。

转换过程中的中间单词必须是字典中的单词。

说明:

如果不存在这样的转换序列，返回 0。

所有单词具有相同的长度。

所有单词只由小写字母组成。

字典中不存在重复的单词。

你可以假设 beginWord 和 endWord 是非空的，且二者不相同。

示例 1:

输入:

beginWord = "hit",

endWord = "cog",

wordList = ["hot","dot","dog","lot","log","cog"]

输出: 5

解释: 一个最短转换序列是 "hit" -> "hot" -> "dot" -> "dog" -> "cog",

返回它的长度 5。

示例 2:

输入:

beginWord = "hit"

endWord = "cog"

wordList = ["hot","dot","dog","lot","log"]

输出: 0

解释: endWord "cog" 不在字典中，所以无法进行转换。

思路：

代码：

1. **class** Solution {
2. public:
3. int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
4. queue<string> q;
5. map<string,int> m1;
6. map<string,int> re;
7. int n=wordList.size();
8. **for**(int i=0;i<n;i++){
9. m1[wordList[i]]=1;
10. }
11. re[beginWord]=1;
12. q.push(beginWord);
14. **while** ((!q.empty()) && m1.size())
15. {
16. string now = q.front();
17. q.pop();
18. int num = re[now];
19. int llen = now.size();
20. **for** (int i = 0; i < llen; i ++)
21. {
22. string temp = now;
23. **for**(char c = 'a' ; c <= 'z'; c ++)
24. {
25. **if**(temp[i] == c)
26. **continue**;
27. **else**
28. temp[i] = c;
29. **if**(m1.find(temp) != m1.end())
30. {
31. **if**(temp == endWord)
32. **return** num + 1;
33. q.push(temp);
34. re[temp] = num + 1;
35. m1.erase(temp);
36. }
37. }
38. }
39. }
40. **return** 0;

43. }
44. };

279完全平方数

题目：

给定正整数 n，找到若干个完全平方数（比如 1, 4, 9, 16, ...）使得它们的和等于 n。你需要让组成和的完全平方数的个数最少。

示例 1:

输入: n = 12

输出: 3

解释: 12 = 4 + 4 + 4.

示例 2:

输入: n = 13

输出: 2

解释: 13 = 4 + 9.

思路：动态规划的题型，首先构造一个大小为n的一维数组。数组表示的是一个图，图的节点直接相连的条件是两数之间相差为平方数。

代码：

1. **class** Solution {
2. **public**:
3. **int** numSquares(**int** n) {
4. vector<**int**> dp(n+1,n);
5. dp[0]=0;
6. **for**(**int** i=0;i<=n;i++){
7. **for**(**int** j=1;i+j\*j<=n;j++){
8. dp[i+j\*j]=min(dp[i+j\*j],dp[i]+1);
9. }
10. }
11. **return** dp[n];
12. }
13. };

529扫雷游戏

题目：

让我们一起来玩扫雷游戏！

给定一个代表游戏板的二维字符矩阵。 'M' 代表一个未挖出的地雷，'E' 代表一个未挖出的空方块，'B' 代表没有相邻（上，下，左，右，和所有4个对角线）地雷的已挖出的空白方块，数字（'1' 到 '8'）表示有多少地雷与这块已挖出的方块相邻，'X' 则表示一个已挖出的地雷。

现在给出在所有未挖出的方块中（'M'或者'E'）的下一个点击位置（行和列索引），根据以下规则，返回相应位置被点击后对应的面板：

如果一个地雷（'M'）被挖出，游戏就结束了- 把它改为 'X'。

如果一个没有相邻地雷的空方块（'E'）被挖出，修改它为（'B'），并且所有和其相邻的方块都应该被递归地揭露。

如果一个至少与一个地雷相邻的空方块（'E'）被挖出，修改它为数字（'1'到'8'），表示相邻地雷的数量。

如果在此次点击中，若无更多方块可被揭露，则返回面板。

思路：

代码：

1. **class** Solution {
2. **public**:
3. vector<vector<**char**>> updateBoard(vector<vector<**char**>>& board, vector<**int**>& click) {
4. **int** x = click[0], y = click[1];
5. **const** **int** M = board.size(), N = board[0].size();
6. **if** (board[x][y] == 'M') {
7. board[x][y] = 'X';
8. } **else** **if** (board[x][y] == 'E') {
9. **int** mcount = 0;
10. **for** (auto p : dirs) {
11. **int** nx = x + p.first, ny = y + p.second;
12. **if** (nx >= 0 && nx < M && ny >= 0 && ny < N && board[nx][ny] == 'M') {
13. mcount ++;
14. }
15. }
16. **if** (mcount > 0) {
17. board[x][y] = '0' + mcount;
18. } **else** {
19. board[x][y] = 'B';
20. **for** (auto p : dirs) {
21. **int** nx = x + p.first, ny = y + p.second;
22. **if** (nx >= 0 && nx < M && ny >= 0 && ny < N && board[nx][ny] == 'E') {
23. vector<**int**> pos = {nx, ny};
24. updateBoard(board, pos);
25. }
26. }
27. }
28. }
29. **return** board;
30. }
31. **private**:
32. vector<pair<**int**, **int**>> dirs = {{0, 1}, {0, -1}, {1, 1}, {1, 0}, {1, -1}, {-1, 1}, {-1, 0}, {-1, -1}};
33. };