

广度优先遍历（Java）

[liweiwei1419](https://leetcode-cn.com/u/liweiwei1419/)发布于 7 小时前2.0k广度优先搜索Java

分析：

* 二维表格上的问题，常用的算法是深度优先遍历、广度优先遍历和并查集，由于这里计算的结果和距离相关，显然使用广度优先遍历；
* 但是题目问的是「距离陆地区域最远的海洋区域」，这和我们的经验稍微有点出入。一般而言，「广度优先遍历」求的是最短路径，但仔细一想，发现其实广度优先遍历也是适用的：
  + 求最短路径的时候，只要找到目标值，返回即可；
  + 求最长路径的时候，**所有目标值都看完以后，才返回**。
* 这道题「广度优先遍历」的起点有多个，但完全不影响算法的正确性，可以假想一个虚拟的起点，初始的起点就是由虚拟起点扩散开来的，示意图可以参考 [题解](https://leetcode-cn.com/problems/as-far-from-land-as-possible/solution/zhen-liang-yan-sou-huan-neng-duo-yuan-kan-wan-miao/) ，特别形象。

这里题目中介绍的「曼哈顿距离」，其实就是对广度优先遍历（BFS）**逐层向外扩散**的精准数学解释，**每扩散一次，曼哈顿距离就加 1**。

* 编写广度优先遍历算法的注意事项：
  + 如果题目要求返回的结果和距离相关，需要在 while 循环内部一下子把当前列表的所有元素都依次取出来，这种「一下子」操作的次数就是我们需要的距离；
  + 如果一个单元格被添加到队列以后，**需要马上将它标记为已经访问**（根据情况，可以直接在原始输入数组上修改，也可以使用一个布尔数组 visited 进行标记），如果不这么做，很可能会出现死循环，这一点如果一开始没有注意到，也可以通过调试代码观察出。

在这里，我给出的是「常规写法」，大家可以参考[「官方题解」](https://leetcode-cn.com/problems/as-far-from-land-as-possible/solution/di-tu-fen-xi-by-leetcode-solution/) 学习这个问题的另外的思路。

使用「广度优先遍历」算法的问题还有：

* 「力扣」第 279 题：[完全平方数](https://leetcode-cn.com/problems/perfect-squares/)
* 「力扣」第 322 题：[零钱兑换](https://leetcode-cn.com/problems/coin-change/)

这些问题虽然看起来和最短路径无关，但只要在纸上写写画画，不难发现依然是求：在树或者图上的最短路径。可以通过这一类问题熟悉 BFS 的写法。

二维表格上的问题还有：

* 「力扣」第 200 题：[岛屿数量](https://leetcode-cn.com/problems/number-of-islands)
* 「力扣」第 994 题：[腐烂的橘子](https://leetcode-cn.com/problems/rotting-oranges/)
* 「力扣」第 130 题：[被围绕的区域](https://leetcode-cn.com/problems/surrounded-regions)
* 「力扣」第 79 题：[单词搜索](https://leetcode-cn.com/problems/word-search)
* 「剑指 Offer 」第 13 题：[机器人的运动范围](https://leetcode-cn.com/problems/ji-qi-ren-de-yun-dong-fan-wei-lcof)

大家可以通过做这些问题，熟悉在二维表格上编码代码的常用技巧：

* 设置方向数组，使得向「四面八方」扩散的代码更加紧凑；
* 设置是否越界的判断函数 inArea()；
* 根据情况，使用二维坐标和一维坐标相互转换的操作，因为二维坐标传入队列的时候，需要封装成数组，创建和销毁数组有一定性能消耗，有些问题如果需要判重，还可能有一点点工作量。

**参考代码**：

* Java

import java.util.LinkedList;

import java.util.Queue;

public class Solution {

public int maxDistance(int[][] grid) {

// 方向向量

int[][] directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

// 由于题目中给出了 grid 的范围，因此不用做输入检查

int N = grid.length;

Queue<Integer> queue = new LinkedList<>();

for (int i = 0; i < N; i++) {

for (int j = 0; j < N; j++) {

if (grid[i][j] == 1) {

queue.add(getIndex(i, j, N));

}

}

}

int size = queue.size();

if (size == 0 || size == N \* N) {

return -1;

}

int step = 0;

while (!queue.isEmpty()) {

int currentQueueSize = queue.size();

for (int i = 0; i < currentQueueSize; i++) {

Integer head = queue.poll();

int currentX = head / N;

int currentY = head % N;

for (int[] direction : directions) {

int newX = currentX + direction[0];

int newY = currentY + direction[1];

// 只关心有效范围内的海洋（0）

if (inArea(newX, newY, N) && grid[newX][newY] == 0) {

// 赋值成为一个不等于 0 的整数均可，因为后续逻辑只关心海洋（0）

grid[newX][newY] = 1;

queue.add(getIndex(newX, newY, N));

}

}

}

step++;

}

// 由于最后一步，没有可以扩散的的区域，但是 step 加了 1，故在退出循环的时候应该减 1

return step - 1;

}

/\*\*

\* **@param** x 二维表格单元格横坐标

\* **@param** y 二维表格单元格纵坐标

\* **@param** cols 二维表格列数

\* **@return**

\*/

private int getIndex(int x, int y, int cols) {

return x \* cols + y;

}

/\*\*

\* **@param** x 二维表格单元格横坐标

\* **@param** y 二维表格单元格纵坐标

\* **@param** N 二维表格行数（列数）

\* **@return** 是否在二维表格有效范围内

\*/

private boolean inArea(int x, int y, int N) {

return 0 <= x && x < N && 0 <= y && y < N;

}

public static void main(String[] args) {

// int[][] grid = {{1, 0, 1}, {0, 0, 0}, {1, 0, 1}};

int[][] grid = {{1, 0, 0}, {0, 0, 0}, {0, 0, 0}};

Solution solution = new Solution();

int res = solution.maxDistance(grid);

System.out.println(res);

}

}

**复杂度分析**：

* 时间复杂度：O(N^2)*O*(*N*2)，这里 N*N* 是方格的边长。二维表格里所有的元素都会被看一遍；
* 空间复杂度：O(N^2)*O*(*N*2)，最坏情况下，方格里全部是陆地（1）的时候，元素全部会进队列。

下一篇：套路大解密：搞懂题目考察什么，剩下的任务就是套模板！

© 著作权归作者所有

25

条评论

最热



编辑

预览

评论

[[](https://leetcode-cn.com/u/tiantianliu2018/)Kyra](https://leetcode-cn.com/u/tiantianliu2018/)

6 小时前

做完题目再来看weiwei大佬的分析，每次都有总结，简直不要太完美！

3踩回复

[[](https://leetcode-cn.com/u/pwrliang/)pwrliang](https://leetcode-cn.com/u/pwrliang/)

6 小时前

为什么从1往0搜？如果写成0往1搜会出现什么问题呢？

赞踩查看 5 条回复回复

[[](https://leetcode-cn.com/u/tang-ji-he-de-yu-sang-qiu/)堂吉诃德与桑丘](https://leetcode-cn.com/u/tang-ji-he-de-yu-sang-qiu/)

5 小时前

讲解的很清楚，赞！最后一步没有扩散区域step加了1，这个可以直接在step++前面加个判断队列是否为空，到最后一步队列已经为空就不再执行+1操作，这样直接返回step就是结果。我参考楼主的给个C++版的：

class Solution {

public:

int maxDistance(vector<vector<int>>& grid) {

if (grid.empty()) return 0;

int n = grid.size();

vector<int> dx {1,-1,0,0};

vector<int> dy {0,0,1,-1};

queue<pair<int,int>> q;

for (int i=0;i<n;i++)

{

for (int j=0;j<n;j++)

{

if (grid[i][j]==1)

{

q.push(make\_pair(i,j));

}

}

}

int m = q.size();

if (m==n\*n || m==0) return -1;

int step = 0;

while(!q.empty())

{

int size = q.size();

for (int i=0;i<size;i++)

{

pair<int,int> d = q.front();

q.pop();

for (int k=0;k<4;k++)

{

int x = d.first + dx[k];

int y = d.second + dy[k];

if (x>=0 && x<n && y>=0 && y<n && grid[x][y]==0)

{

grid[x][y] = 1;

q.push(make\_pair(x,y));

}

}

}

if(!q.empty()) step++;

}

return step;

}

};

赞踩查看 4 条回复回复

[[](https://leetcode-cn.com/u/smilecandy-2/)smilecandy](https://leetcode-cn.com/u/smilecandy-2/)

6 小时前

请问getIndex方法是什么意思呢？

赞踩查看 3 条回复回复

[[](https://leetcode-cn.com/u/zwyin/)zwyin](https://leetcode-cn.com/u/zwyin/)

6 小时前

queue里直接存数组即可，这里计算index是为了节省空间？

赞踩查看 1 条回复回复

[[](https://leetcode-cn.com/u/jiaojing/)jiaojing](https://leetcode-cn.com/u/jiaojing/)

1 小时前

if (inArea(newX, newY, N) && grid[newX][newY] == 0) 这步判断出是海洋后 为什么还要 queue.add(getIndex(newX, newY, N)); 把这个加回到queue队列里呢， 那么最后是怎么判断最短距离的呀，求赐教

赞踩回复

[[](https://leetcode-cn.com/u/wang_ni_ma/)王尼玛](https://leetcode-cn.com/u/wang_ni_ma/)

3 小时前

感觉这种二维表格问题出现的频率挺高的啊，3月打卡的题目中有好几道都是这种类型的。

赞踩回复

[[](https://leetcode-cn.com/u/yayakoblessing/)扭秧歌的一只泱](https://leetcode-cn.com/u/yayakoblessing/)

4 小时前

发现看weiwei大佬题解的另一大好处 可以不用自己写测试用例了（构建二维数组好麻烦

赞踩回复

[[](https://leetcode-cn.com/u/fuxuemingzhu/)负雪明烛](https://leetcode-cn.com/u/fuxuemingzhu/)

4 小时前

赞weiwei的分析，看懂你的一个题解就会了一大片题哈哈。（点击我的主页有每日一题交流群，欢迎大家加入。）

赞踩回复

[[](https://leetcode-cn.com/u/demondaydream/)DemonDayDream](https://leetcode-cn.com/u/demondaydream/)

（编辑过）4 小时前

step++;

}

// 由于最后一步，没有可以扩散的的区域，但是 step 加了 1，故在退出循环的时候应该减 1

return step - 1;

也可以写成：

if(queue.size()>0)step++;

}

return step;

即：每一次bfs后，如果queue中有下一层available的节点，才+1。

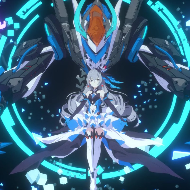
赞踩回复

[[](https://leetcode-cn.com/u/ageovb/)刘辉](https://leetcode-cn.com/u/ageovb/)

5 小时前

大神还是大神，学习了，有些人题解是写了，可是代码连个注释都没有（说的就是官方），真难看

赞踩回复

[[](https://leetcode-cn.com/u/comprehensive/)comprehensive](https://leetcode-cn.com/u/comprehensive/)

6 小时前

nice，wei哥