Leetcode\_200\_NumberOfIslands\_岛屿数量\_深搜\_宽搜\_Medium

# Leetcode\_200\_NumberOfIslands\_岛屿数量\_深搜\_宽搜\_Medium

## 题目介绍

\* 难度：Medium

\* <https://leetcode.com/problems/number-of-islands/description/>

\* Given a 2d grid map of '1's (land) and '0's (water), count the number of islands.

\* An island is surrounded by water and is formed

\* by connecting adjacent lands horizontally or vertically.

\* You may assume all four edges of the grid are all surrounded by water.

\*

\* Example 1:

\* Input:

\* 11110

\* 11010

\* 11000

\* 00000

\* Output: 1

**\* Example 2：**

\* Input:

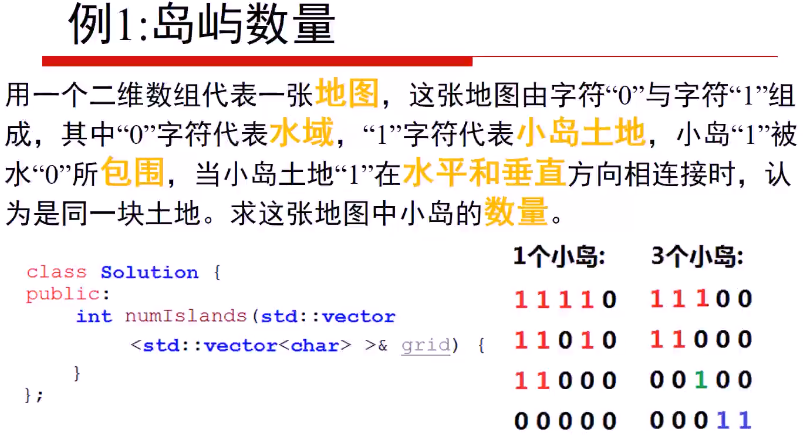
\* 11000

\* 11000

\* 00100

\* 00011

\* Output: 3



## 思路分析

\*思路方法：深度优先搜索与宽度优先搜索。

\* 方法1：深度优先搜索：DFS。

\* 创建一个与grid对应的visited布尔数组，用于标记mark该格子是否已经访问过；

\* 然后利用嵌套循环遍历每一个格子，只有当前格子为1且visited未访问过，

\* 调用DFS搜索该岛屿的所有格子，每一次调用DFS，都会自动递归地将该岛屿

\* 上下左右四个格子都标记为true，同时岛屿计数加一，最后返回岛屿总数即可。

\* 方法2：宽度优先搜索：BFS。

\* 利用一个队列数据结构结合while循环搜索并标记一个岛屿的所有的方格。

\* 首先岛屿的第一个方格进入队列，进入即标记为true；然后弹出该点的时候，

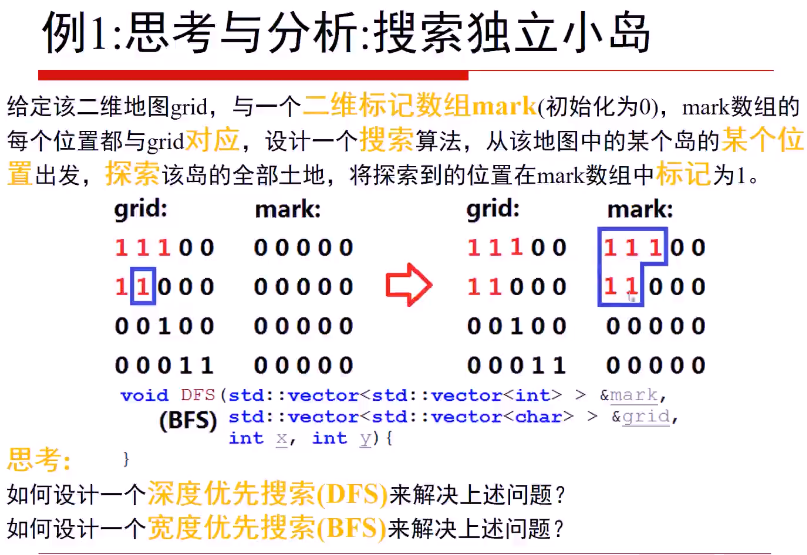
\* 将上下左右满足条件的四个方格进入队列，并标记为true。

\* 直到队列为空，至此一个岛屿搜索并标记完毕。

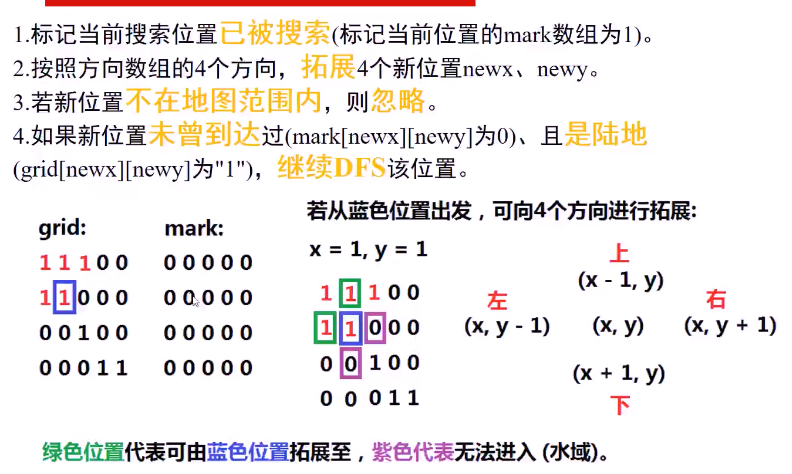
\*

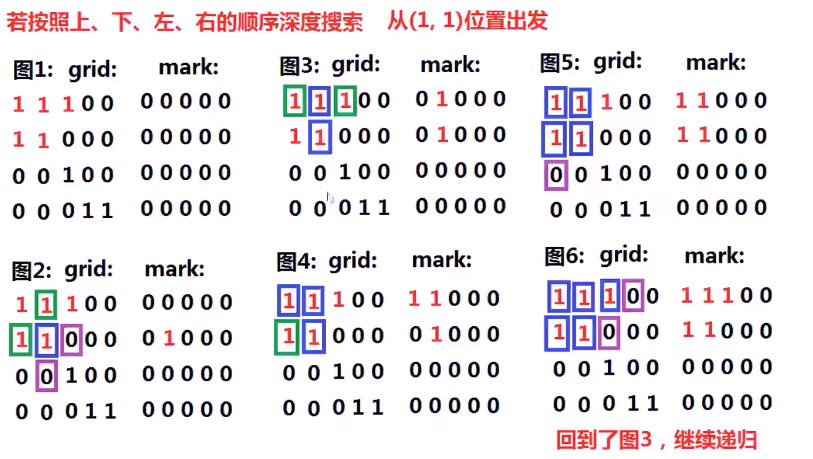
\* **DFS与BFS的区别**：DFS利用递归的方式实现搜索并标记；

\* BFS利用队列数据结构并结合while循环实现搜索并标记。

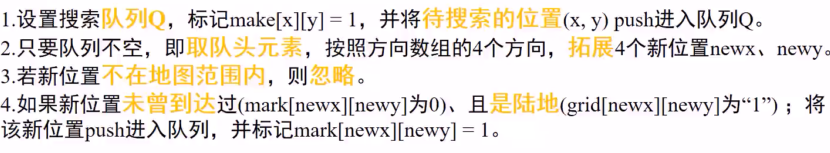


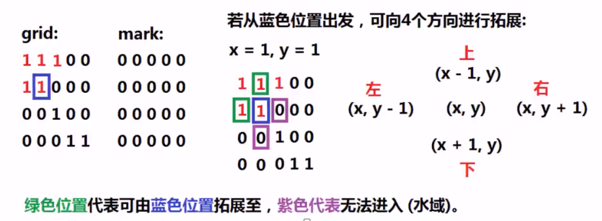
### 深度优先搜索DFS



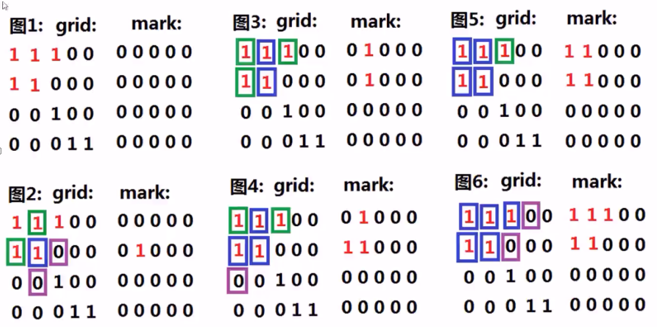


### **宽度优先搜索BFS**





若按照上下左右的顺序进行宽度搜索，从(1,1)出发：

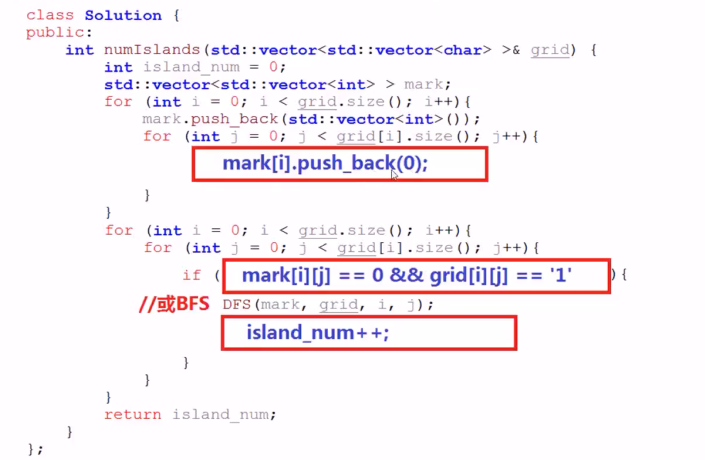


### 通用方法：



## Java代码

通用方法：



/\*\*

\* 通用方法：无论DFS或BFS，都是需要遍历每个点，判断是否是一个新岛屿的开始。

\* 若是一个新岛屿的开始，则会利用DFS或BFS标记该岛屿的每个方格。

\*/

public int numIslands(char[][] grid) {

if(grid == null||grid.length == 0||grid[0].length == 0) return 0;

int rows = grid.length,columns = grid[0].length;

boolean[][] visited = new boolean[rows][columns];

int count = 0;

//嵌套循环，遍历每一个点

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

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

if(grid[i][j] == '1' && !visited[i][j]){//新岛屿的起点

// DFS(grid,visited,i,j);//通过递归调用将该岛屿的所有格标记为true

BFS(grid,visited,i,j);//通过递归调用将该岛屿的所有格标记为true

count++;

}

}

}

return count;

}

### **深度优先搜索DFS**

//方法1：DFS深度优先搜索

public void **DFS**(char[][] grid,boolean[][] visited,int i,int j ){

if(i < 0 ||i >= grid.length|| j <0 || j >= grid[i].length||grid[i][j] != '1'|| visited[i][j]) return;

visited[i][j] = true;//标记已经访问过

DFS(grid,visited,i-1,j);//上

DFS(grid,visited,i,j+1);//右

DFS(grid,visited,i+1,j);//下

DFS(grid,visited,i,j-1);//左

}



### **宽度优先搜索**

//方法2：BFS宽度优先搜索

public void BFS(char[][] grid,boolean[][] visited,int x,int y ){

if(x < 0 ||x >= grid.length|| y <0 || y >= grid[x].length||grid[x][y] != '1'|| visited[x][y]) return;

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

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

ArrayDeque<Integer> deque = new ArrayDeque<Integer>();

deque.addLast(x);

deque.addLast(y);

visited[x][y] = true;**//一定要在进入队列的时候就做标记，否则会使同一个点多次进入队列**

**//为什么？因为队列弹出一个点可能会进去4个点，若是弹出再标记的话，**

**// 则4个点都是未访问的，会多次进入队列中**

while(!deque.isEmpty()){

x = deque.removeFirst();

y = deque.removeFirst();

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

int newX = x + dx[k];

int newY = y + dy[k];

if(newX < 0 ||newX >= grid.length|| newY <0 || newY >= grid[x].length||grid[newX][newY] != '1'|| visited[newX][newY])

continue;

deque.addLast(newX);

deque.addLast(newY);

visited[newX][newY] = true;

}

}

}

