积水\_LeetCode\_42\_I\_407\_II\_TrappingRainWater\_Hard

\* 积累雨水==>积累洪水

\* T1：LeetCode\_42\_I\_TrappingRainWater\_Hard<二维>

\* T2：LeetCode\_407\_II\_TrappingRainWater\_Hard<三维>

# T1：LeetCode\_42\_I\_TrappingRainWater\_Hard<二维>

## 题目介绍

/\*\*

\* LeetCode\_42\_TrappingRainWater\_Hard\_I

\* 难度：Hard

\* DateTime：2018-10-07 14:33

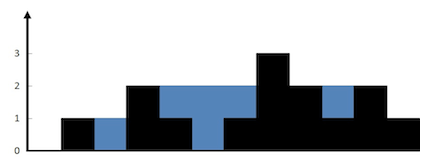
\* 题目介绍：

\* Given n non-negative integers representing an elevation map

\* where the width of each bar is 1, compute how much water

\* it is able to trap after raining.

\* LeetCode上右图片



\* The above elevation map is represented by array [0,1,0,2,1,0,1,3,2,1,2,1].

\* In this case, 6 units of rain water (blue section) are being trapped.

\* Thanks Marcos for contributing this image!

\* Example:

\* Input: [0,1,0,2,1,0,1,3,2,1,2,1]

\* Output: 6

## 思路分析

\* 思路分析：两端双指针法。

\* 将该问题模拟成水从左右两边逐渐涌上来，直至将所有方块淹没，

\* 在涌水逐渐上升的过程，计算积水的体积；

\* 最后水都退去，此时所能收集的雨水就是最终结果。

\* 既然是左右两边逐渐涌上来，那么两个指针left和right表示左右水前进的位置；

\* leftMax和rightMax表示左右两端方块的最大高度，

\* left和right的前进原则：肯定是哪个低，哪个先前进，因此需要

\* 判断 if(heights[left]<heights[right])决定左边还是右边。

\* 误区：该题虽然是积累雨水，但是思考问题的方式应该是洪水从两边进，

\* 而不是从中间往外溢出。同样思想适用于LeetCode\_407\_Trapping\_Rain\_Water\_II。

## Java代码

/\*\*

\* 两端双指针法

\*/

public int trap(int[] heights) {

//只有多余3个方块才有积水的可能

if (heights.length < 3) return 0;

//初始化

int left = 0, right = heights.length - 1;

int leftMax = 0, rightMax = 0;

int result = 0;

//while循环：左右夹击，哪边低哪边先进水

while (left < right) {

if (heights[left] <= heights[right]) {//洪水从左边进

if(heights[left] > leftMax) leftMax = heights[left];

result += leftMax - heights[left];

left++;

}else{//洪水从右边进

if(heights[right]>rightMax) rightMax = heights[right];

result += rightMax-heights[right];

right--;

}

}

return result;

}

# T2：LeetCode\_407\_II\_TrappingRainWater\_Hard<三维>

## 题目介绍

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*T2：LeetCode\_407\_II\_TrappingRainWater\_Hard<三维>\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

\* LeetCode\_407\_II\_TrappingRainWater\_Hard<三维>

\* 难度：Hard

\* DateTime：2018-10-07 16:24

\* https://leetcode.com/problems/trapping-rain-water-ii/description/

\*

\* 题目介绍：

\* Given an m x n matrix of positive integers representing the height of each unit cell

in a 2D elevation map, compute the volume of water it is able to trap after raining.

\*

\* Note:

\* Both m and n are less than 110. The height of each unit cell

is greater than 0 and is less than 20,000.

\*

\* Example:

\*

\* Given the following 3x6 height map:

\* [

\* [1,4,3,1,3,2],

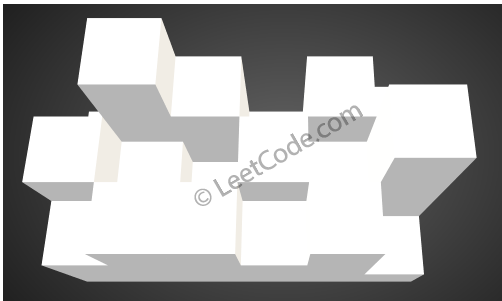
\* [3,2,1,3,2,4],

\* [2,3,3,2,3,1]

\* ]

\* Return 4.

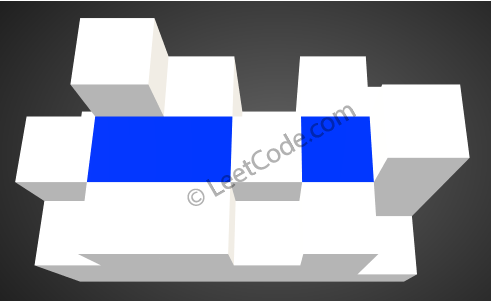
\* LeetCode有图片1



\* The above image represents the elevation map

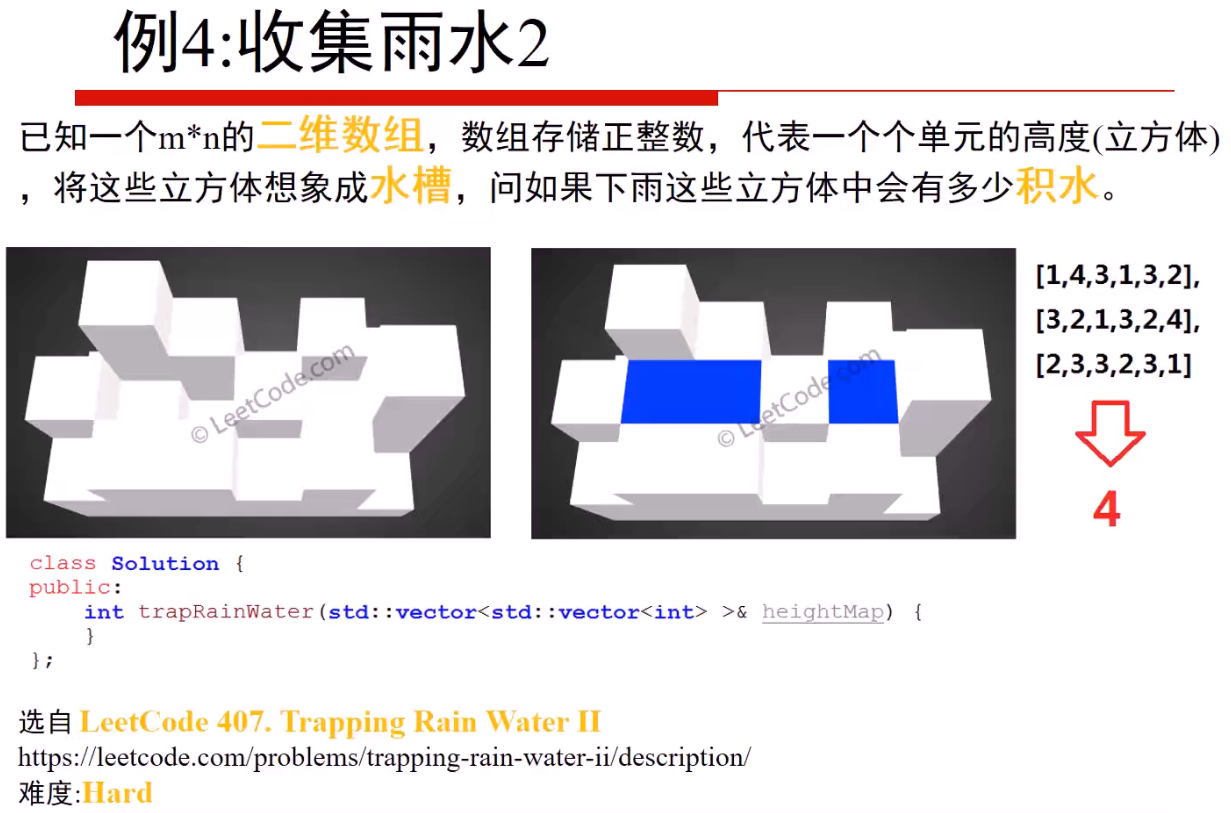
\* [[1,4,3,1,3,2],[3,2,1,3,2,4],[2,3,3,2,3,1]] before the rain.

\* 图片2



\* After the rain, water is trapped between the blocks.

\* The total volume of water trapped is 4.



## 思路分析

\* 思路介绍：带有优先级的广度优先搜索BFS。Breadth-First-Search。

\* 使用到优先级队列,一个boolean二维数组对应是否标记过；

\* 具体操作步骤：

\* S1：将四周元素push到优先级队列中，然后四周mark标记为true;

\* S2：从优先级队列中获取一个最小的方块，记当为前高度为h，

\* 然后依次遍历其上下左右四个方块；<这四个没有先后顺序>

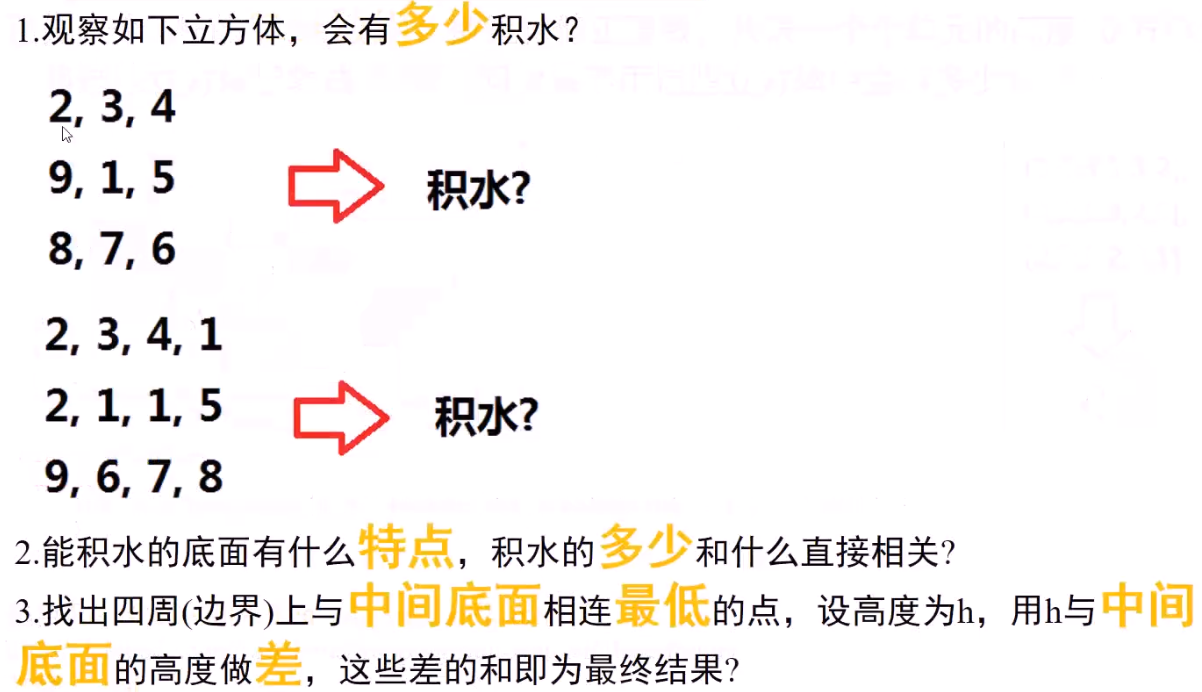
\* 首先判断该方块有没有标记过，若已经标记true，直接continue，否则继续：

\* 若遍历的方块高度小于h，则表示可以积水，累加result，并将该方块处高度设置为h，若遍历的方块高度大于等于h，则表示不可以积水；

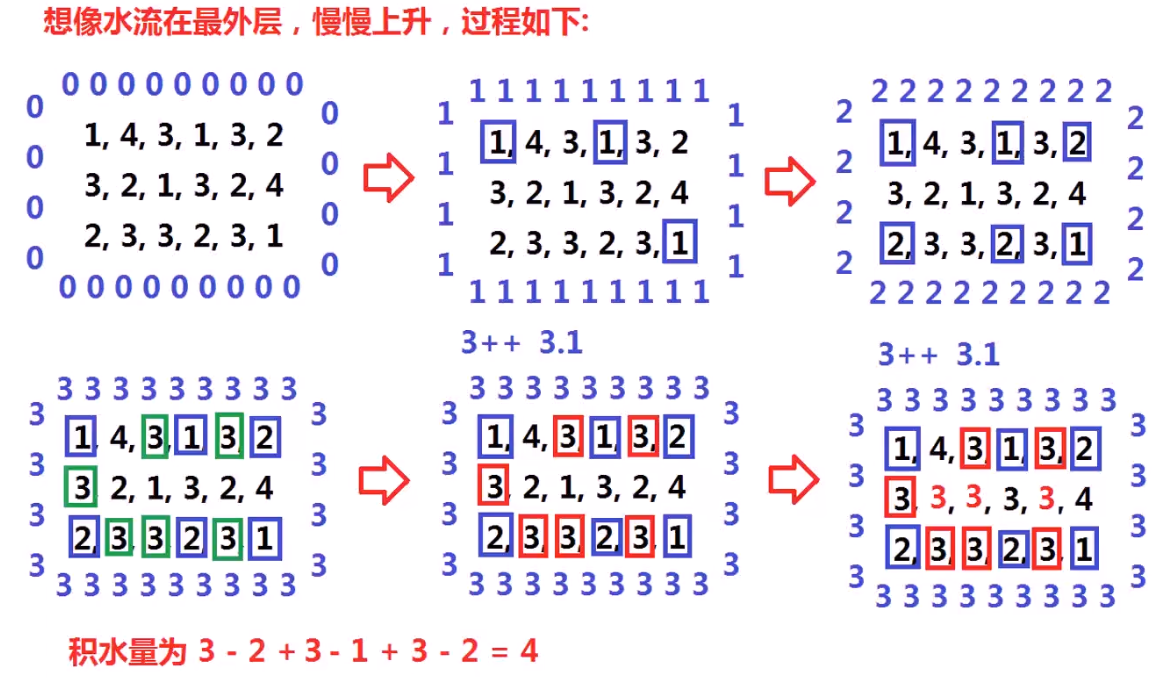
\* 无论可不可以积水，都需要将该方块push到优先级队列中，并对应mark处设为true。

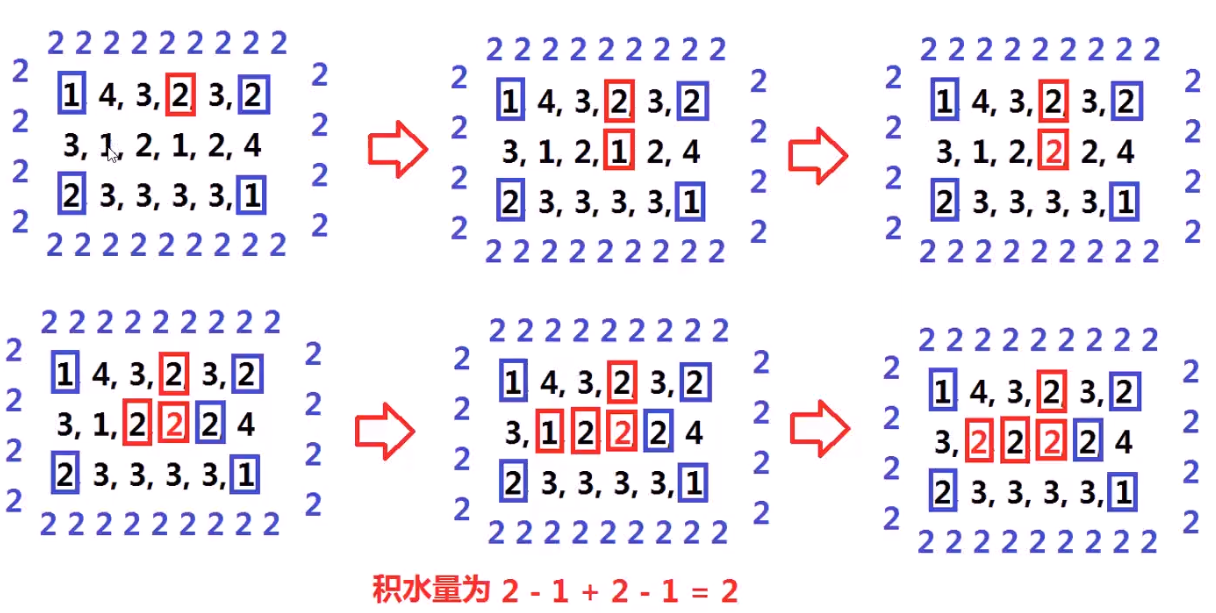
\* while循环至优先级队列为空即可。

### 分析过程



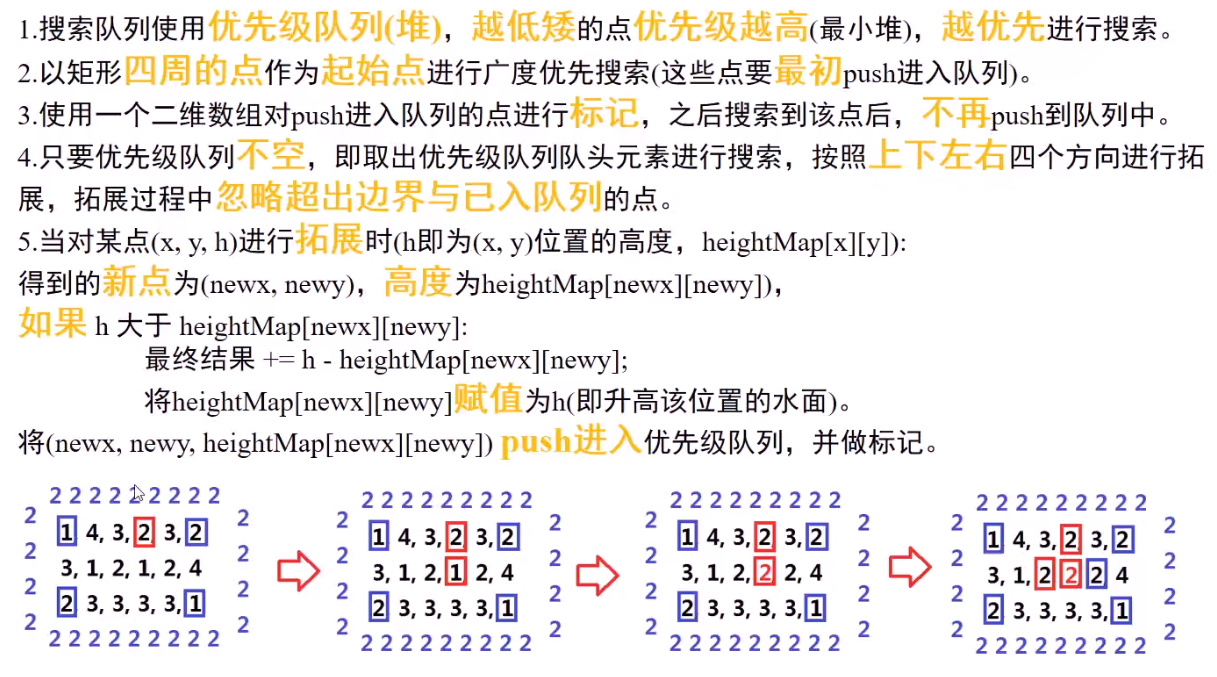




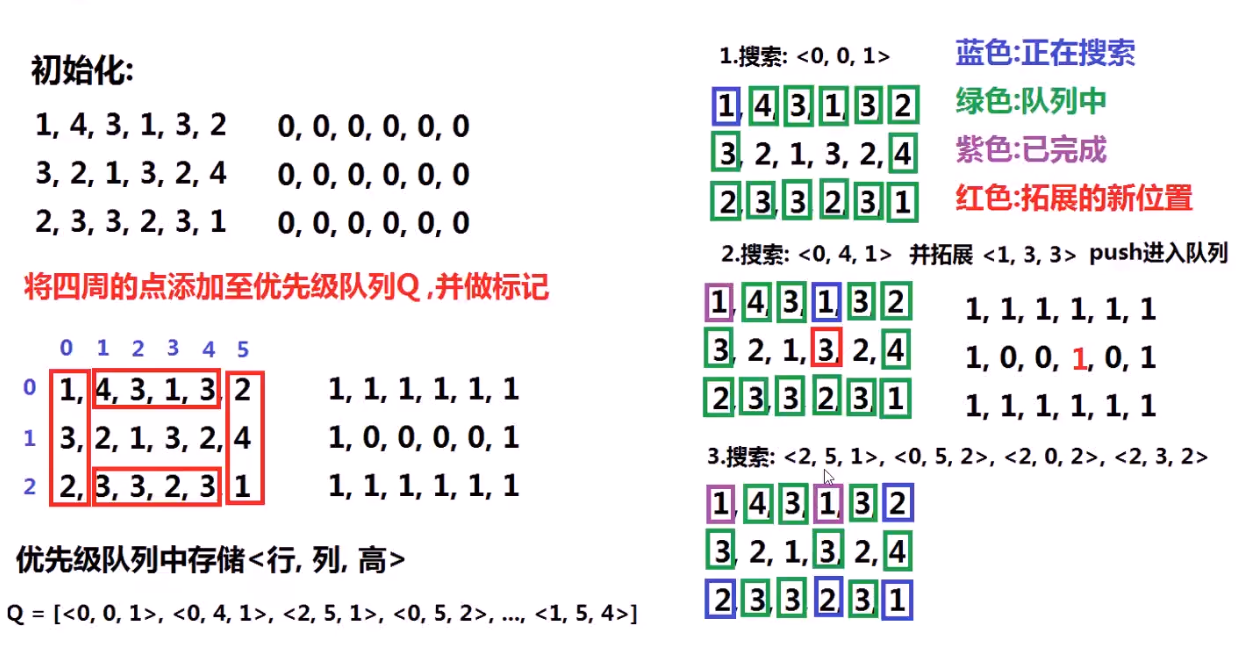


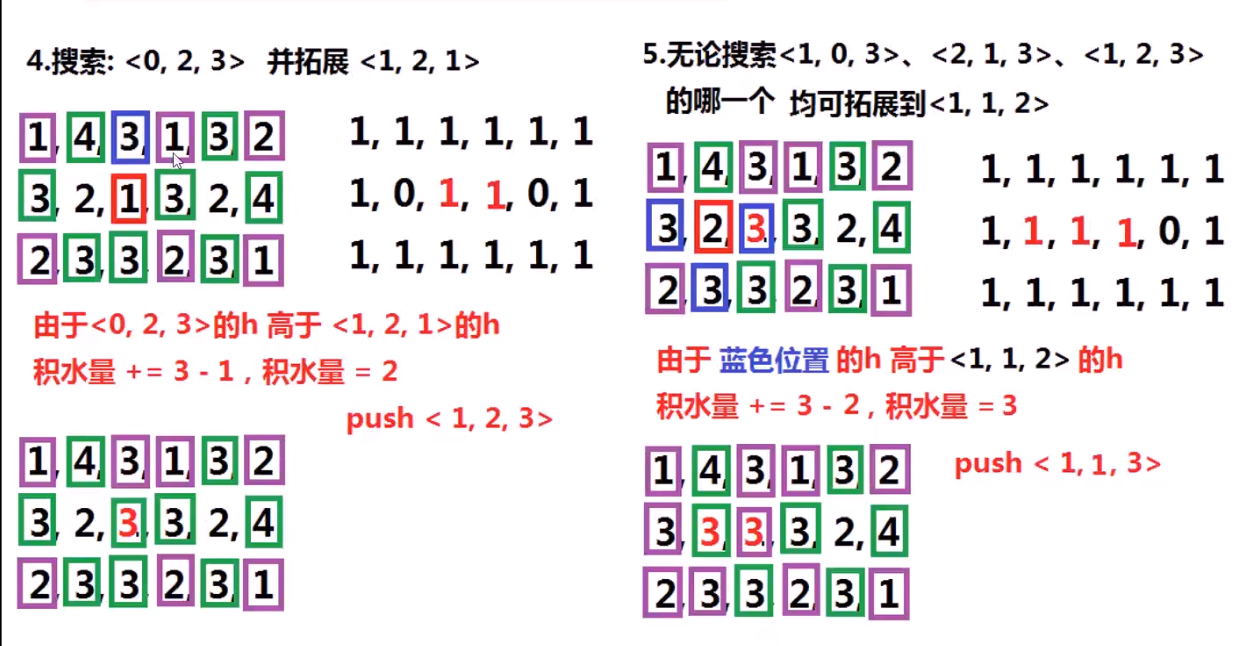
### 思路整理：

带有优先级的广度优先搜索。



### 示例分析：







## Java代码

### Java代码

public int trapRainWater(int[][] heightMap) {

//S1:行数和列数必须大于等于3才可以积水

if (heightMap.length < 3 || heightMap[0].length < 3) return 0;

//S2:初始化

int result = 0;

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

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

int rows = heightMap.length;

int colums = heightMap[0].length;

boolean[][] mark = new boolean[rows][colums];//默认为false

//优先级队列：第一次使用

PriorityQueue<QItem> prioQueue = new PriorityQueue<QItem>(new Comparator<QItem>() {

@Override

public int compare(QItem o1, QItem o2) {

return o1.h - o2.h;//保证h越小，优先级越高

}

});

//S3:将四周方块push到优先级队列中

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

prioQueue.add(new QItem(0, i, heightMap[0][i]));//第一行

mark[0][i] = true;

prioQueue.add(new QItem(rows - 1, i, heightMap[rows - 1][i]));//最后一行

mark[rows - 1][i] = true;

}

for (int i = 1; i < rows - 1; i++) {

prioQueue.add(new QItem(i, 0, heightMap[i][0]));//第一列

mark[i][0] = true;

prioQueue.add(new QItem(i, colums - 1, heightMap[i][colums - 1]));//最后一列

mark[i][colums - 1] = true;

}

//S4:核心

while (!prioQueue.isEmpty()) {

QItem item = prioQueue.remove();//获取对头元素(h最小的元素)

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

int newx = item.x + dx[i];

int newy = item.y + dy[i];

if (newx < 0 || newx >= rows ||

newy < 0 || newy >= colums || mark[newx][newy]) {

continue;

}

int h = item.h;//此时外围最低高度

if(h > heightMap[newx][newy]){

result += h - heightMap[newx][newy];

heightMap[newx][newy] = h;

}

mark[newx][newy] = true;

prioQueue.add(new QItem(newx,newy,heightMap[newx][newy]));//push到队列中

}

}

return result;

}

/\*\*

\* 自定义优先级队列元素的数据结构

\*/

private class QItem {

int x;

int y;

int h;

public QItem(int x, int y, int h) {

this.x = x;

this.y = y;

this.h = h;

}

}

### C语言

