昵称：lichen782
园龄：1年2个月
粉丝：3
关注：0
+加关注

< 2014年1月 >

日	一	二	三	四	五	六
29	30	31	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

algorithm(23)
leetcode(22)

随笔分类

LeetCode(22)

随笔档案

2013年7月 (23)

最新评论

1. Re:LeetCode 笔记系列 17 Largest Rectangle in Histogram
有一个问题，3322这种，这样是不是计算出来的是6啊。不是8，倒数第三张图里的棕色框明显不是可以取到的最大面积了，应该加上左边的1才是对吧
--TiegerSong

2. Re:LeetCode 笔记系列 17 Largest Rectangle in Histogram
亲，这个算法错了，你试试2 1 3 4 2 3
--sr2013

3. Re:LeetCode 笔记系列 19 Scramble String [合理使用递归]
超时了吧。。。
--无敌的佩恩

4. Re:LeetCode 笔记系列二 Container With Most Water
第二种方法的剪枝不错啊！
第三种就是短板效应的具体体现吧
--WingTsubasa

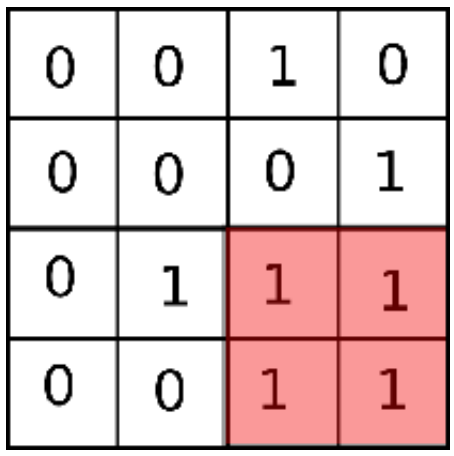
5. Re:LeetCode 笔记系列七 Substring with Concatenation of All Words
这题是压线通过，更好的优化应该用类似KMP的算法。请参考：

博客园 首页 博问 闪存 新随笔 联系 订阅XML 管理

随笔-23 评论-9 文章-0 trackbacks-0

LeetCode 笔记系列 18 Maximal Rectangle [学以致用]

题目： **Given a 2D binary matrix filled with 0's and 1's, find the largest rectangle containing all ones and return its area.**



leetcode的题目真是越来越经典了。比如这个题目，就是一道男默女泪的题。

一般人拿到这个题目，除非做过类似的，很难一眼找出一个方法来，更别说找一个比较优化的方法了。

首先一个难点就是，你怎么判断某个区域就是一个矩形呢？

其次，以何种方式来遍历这个**2D**的**matrix**呢？

一般来说，对这种"棋盘式"的题目，像什么**Queen**啦，象棋啦，数独啦，如果没有比较明显的遍历方式，可以采用一行一行地遍历。（好像废话哦。。。）

然后，当遍历到**(i, j)**的时候，该做什么样的事情呢？想想，嗯，那我可不可以简单看看，以**(i,j)**为矩形左上角，能不能形成一个矩形，能不能形成多个矩形？那形成的矩形中，我们能不能找一个最大的呢？（有同学问，为毛你要以这个点为左上角，不为左下角，或者其他脚哩？因为我们打算从左到右，从上到下一行一行遍历嘛，这样就不会漏掉，说不定还能做一些优化）

首先，如果**(i, j)**是**0**，那肯定没法是矩形了。

如果是**1**，那么我们怎么找以它为左上角的矩形呢？呼唤画面感！

。。。你**TM**在逗我？**==b**

图中圈圈表示左上角的**1**，那么矩形的可能性是。。。太多啦，怎么数嘛！

我们可以试探地从左上角的**1**所在的列开始，往下数数，然后呢，比如在第一行，例如是蓝色的那个矩形，我们看看在列上，它延伸了多远，这个面积是可以算出来的。

然后继续，第二行，例如是那个红色的矩形，再看它延伸到多远，哦，我们知道，比第一行近一些，我们也可以用当前离第一行的行数，乘以延伸的距离，得到当前行表示的矩形面积。

但是到了第一个虚线的地方，它远远超过了上面的其他所有行延伸的距离了，注意它的上方都是空心的哦，所以，

我们遇到这种情况，计算当前行和左上角**1**围成的面积的时候，**只能取所有前面最小的延伸**

http://n00tc0d3r.blogspot.com/2013/06/substring-with-concatenation-of-all.html

--WingTsubasa

距离乘以当前离第一行的行数。其实，这对所有情况都是这样的，是吧？于是，我们不是就有方法遍历这些所有的矩形了嘛。

代码如下：

```
    /**
     * 以给出的坐标作为左上角，计算其中的最大矩形面积
     * @param matrix
     * @param row 给出坐标的行
     * @param col 给出坐标的列
     * @return 返回最大矩形的面积
     */
    private int maxRectangle(char[][] matrix, int row, int col) {
        int minWidth = Integer.MAX_VALUE;
        int maxArea = 0;
        for (int i = row; i < matrix.length && matrix[i][col] == '1'; i++) {
            int width = 0;
            while (col + width < matrix[row].length
                    && matrix[i][col + width] == '1') {
                width++;
            }
            if (width < minWidth) {// 如果当前宽度小于了以前的最小宽度，更新它，为下面的矩形计算做准备
                minWidth = width;
            }
            int area = minWidth * (i - row + 1);
            if (area > maxArea)
                maxArea = area;
        }
        return maxArea;
    }
```

这样，我们再对每个点都调用一下上面的这个方法，不是就能求出最大面积了么。

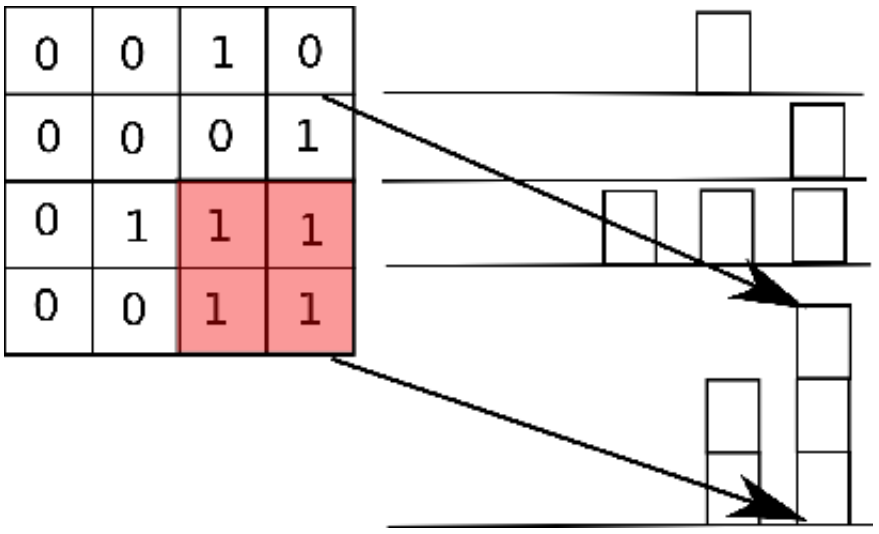
解法一：

```
public int maximalRectangle(char[][] matrix) {
        // Start typing your Java solution below
        // DO NOT write main() function
        int m = matrix.length;
        int n = m == 0 ? 0 : matrix[0].length;
        int maxArea = 0;
        for(int i = 0; i < m; i++){//row
            for(int j = 0; j < n; j++){//col
                if(matrix[i][j] == '1'){
                    int area = maxRectangle(matrix, i, j);
                    if(area > maxArea) maxArea = area;
                }
            }
        }
        return maxArea;
    }
```

这个需要**O(n^3)**，所以没有通过大集合的测试。

leetcode的讨论组给出了一个比较难理解的方法，这里就不采用了。

说说第三个方法。前一个笔记，我们讨论了柱状图的最大矩形面积，那可以**O(n)**的，学以致用呀！**btw**，**leetcode**的这两题也是挨一块儿的，用心良苦。。。。



如果我们把每一行看成**x**坐标，那高度就是从那一行开始往上数的**1**的个数。带入我们的**maxAreaInHist**方法，在**O(n^2)**时间内就可以求出每一行形成的"柱状图"的最大矩形面积了。它们之中最大的，就是我们要的答案。

代码如下：

```
1 public int maximalRectangle2(char[][] matrix) {
2         int m = matrix.length;
3         int n = m == 0 ? 0 : matrix[0].length;
4         int[][] height = new int[m][n + 1];
5         //actually we know that height can just be a int[n+1],
6         //however, in that case, we have to write the 2 parts together in row traverse,
7         //which, leetcode just doesn't make you pass big set
8         //所以啊，leetcode是喜欢分开写循环的，即使时间复杂度一样，即使可以节约空间
9         int maxArea = 0;
10        for(int i = 0; i < m; i++){
11            for(int j = 0; j < n; j++) {
12                if(matrix[i][j] == '0'){
13                    height[i][j] = 0;
14                }else {
15                    height[i][j] = i == 0 ? 1 : height[i - 1][j] + 1;
16                }
17            }
18        }
19        for(int i = 0; i < m; i++){
20            int area = maxAreaInHist(height[i]);
21            if(area > maxArea){
22                maxArea = area;
23            }
24        }
25        return maxArea;
26    }
27
28    private int maxAreaInHist(int[] height){
29        Stack<Integer> stack = new Stack<Integer>();
30        int i = 0;
31        int maxArea = 0;
32        while(i < height.length){
33            if(stack.isEmpty() || height[stack.peek()] <= height[i]){
34                stack.push(i++);
35            }else {
36                int t = stack.pop();
37                maxArea = Math.max(maxArea, height[t] * (stack.isEmpty() ? i : i -
stack.peek() - 1));
38            }
39        }
40        return maxArea;
41    }
```

这里有一个和**leetcode**相关的细节。就是本来在计算**height**数组的时候，我们没有必要分配成代码中的那个样子，一维就可以了，然后在遍历每一行的时候计算当前行的**height**数组，然后再计算**maxArea**。这种情况下还是过不了大集合，所以不得不为每一行都保存一个**height**，先期计算该二维数组。

总结：

1. 学到的新知识要用；

2. 画面感和逻辑分析都很重要，不可偏非。

分类: **LeetCode**

标签: **leetcode**, **algorithm**