算法分析和复杂性理论

第 7 次作业

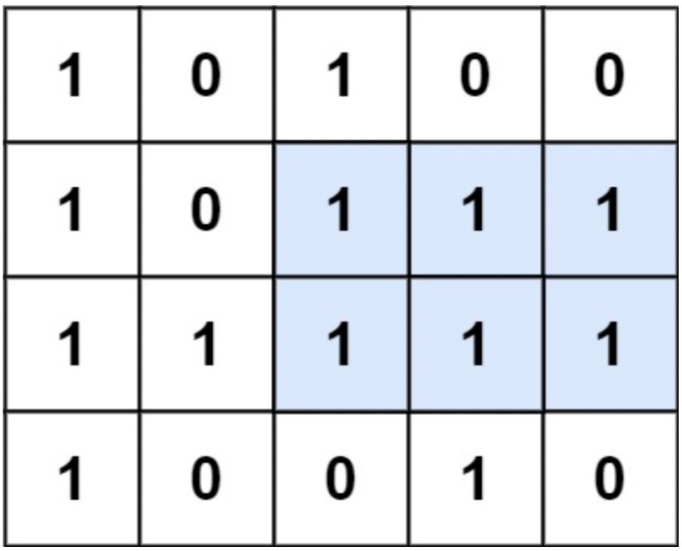
张瀚文 2201212865

1 最大矩形（085）

题目描述：

给定一个仅包含 `0` 和 `1` 、大小为 `rows x cols` 的二维二进制矩阵，找出只包含 `1` 的最大矩形，并返回其面积。

示例 1：



输入： matrix = [["1","0","1","0","0"],["1","0","1","1","1"],
["1","1","1","1","1"],["1","0","0","1","0"]]
输出： 6
解释： 最大矩形如上图所示。

解题思路：在 84 题当中，题目给出的是一个个竖直类型的矩形，要求这些矩形组合当中能够找到的最大面积。在这题当中我们可以对 01 的数字矩阵也做这么一个类似的变形，将从底部开始连续延伸的 1 的数量看成是竖直摆放的矩形的高度，这样我们这题就可以使用上一题的思路进行求解了。在上一题我们计算矩形面积的时候用到了两个单调栈，分别计算了某一个高度向左、向右能够延伸到的最远距离，其实这并没有必要。因为我们用一个栈也可以同时计算出两边的边界。举个例子：[1, 3, 6, 7]，当前元素是 5。我们需要把 6，7 出栈，5 入栈。我们知道了 5 的左边界是 3，但仔细想一想，对于 7 来说，我们知道了它的左右边界。7 的左边界是 6，右边界是 5。也就是说对于栈顶的元素而言，

它的左边界是 stack[top-1]，右边界是当前的位置 i，宽就是 i - stack[top-1] - 1。

运行结果：

测试代码：

```
class Solution:
    def maximalRectangle(self, matrix: List[List[str]]) -> int:
        m,n = len(matrix),len(matrix[0])
        for i in range(m):
            for j in range(n):
                if j==0:
                    matrix[i][j] = int(matrix[i][j])
                else:
                    matrix[i][j] = matrix[i][j-1]+1 if matrix[i][j]=='1' else 0
        ans = 0
        for j in range(n):
            monostack = list()
            left = [-1]*m
            right = [m]*m
            for i in range(m):
                while monostack and matrix[monostack[-1]][j]>matrix[i][j]:
                    right[monostack[-1]] = i
                    monostack.pop()
                left[i] = monostack[-1] if monostack else -1
                monostack.append(i)
            tmp = max((right[i]-left[i]-1)*matrix[i][j] for i in range(m))
            ans = max(ans,tmp)
        return ans
```

2 乘积最大子数组（152）

题目描述：

给你一个整数数组 `nums` ，请你找出数组中乘积最大的非空连续子数组（该子数组中至少包含一个数字），并返回该子数组所对应的乘积。

测试用例的答案是一个 **32-位** 整数。

子数组 是数组的连续子序列。

示例 1:

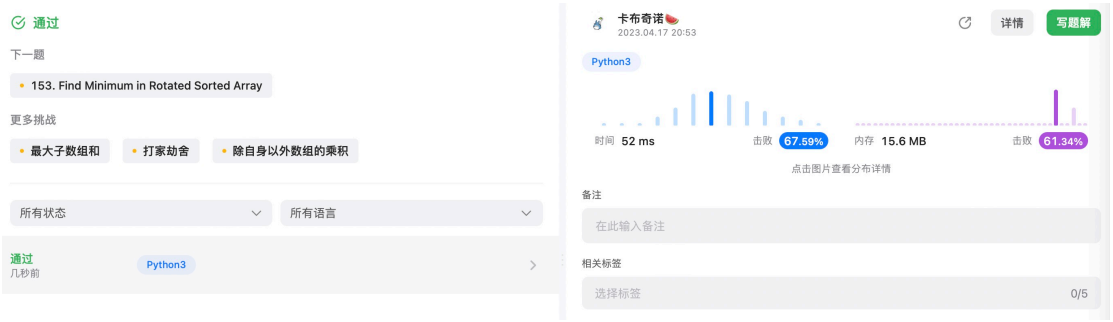
```
输入: nums = [2,3,-2,4]
输出: 6
解释: 子数组 [2,3] 有最大乘积 6。
```

示例 2:

```
输入: nums = [-2,0,-1]
输出: 0
解释: 结果不能为 2, 因为 [-2,-1] 不是子数组。
```

解题思路： 只要记录前 i 的最小值和最大值，那么 dp[i] = max(nums[i] * pre_max, nums[i] * pre_min, nums[i])

运行结果：



测试代码：

```
class Solution:
    def maxProduct(self, nums: List[int]) -> int:
        if not nums: return
        res = nums[0]
        pre_max = nums[0]
        pre_min = nums[0]
        for num in nums[1:]:
            cur_max = max(pre_max * num, pre_min * num, num)
            cur_min = min(pre_max * num, pre_min * num, num)
            res = max(res, cur_max)
            pre_max = cur_max
            pre_min = cur_min
```

```
return res
```