**28203:【模板】单调栈**

给出项数为 n 的整数数列 a1...an。

定义函数 f(i) 代表数列中第 i 个元素之后第一个大于 ai 的元素的**下标**。若不存在，则 f(i)=0。 试求出 f(1...n)。

输入

第一行一个正整数 n。 第二行 n 个正整数 a1...an。

输出

一行 n 个整数表示 f(1), f(2), ..., f(n) 的值。 样例输入

5

1 4 2 3 5

样例输出

2 5 4 5 0

思路：用一个栈记录当前还没找到结果的对象所在的位置，并压入栈中，每次新遍历到序列中的一个 数，就跟栈中的每个数比较，找到了就弹出来，找不到就继续待着，最后一起赋值为0弹出来

代码

#

n = int(input())

a = list(map(int, input().split())) stack = []

for i in range(n):

while stack and a[stack[-1]] < a[i]: a[stack.pop()] = i + 1

stack.append(i) while stack:

a[stack[-1]] = 0 stack.pop()

print(\*a)

# 27205: 护林员盖房子 加强版

在一片保护林中，护林员想要盖一座房子来居住，但他不能砍伐任何树木。 现在请你帮他计算：保护林 中所能用来盖房子的矩形空地的最大面积。

**输入** 保护林用一个二维矩阵来表示，长宽都不超过20（即<=20）。 第一行是两个正整数m,n，表示矩阵 有m行n列。 然后是m行，每行n个整数，用1代表树木，用0表示空地。

**输出** 一个正整数，表示保护林中能用来盖房子的最大矩形空地面积。 样例输入

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | 5 |  | | |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 |

样例输出

5

提示

子矩阵边长可以为1，也就是说： 0 0 0 0 0 依然是一个可以盖房子的子矩阵。 看成是以不同的底竖直摆放的矩形的高度。

def maximalRectangle(matrix) -> int: if (rows := len(matrix)) == 0:

return 0

cols = len(matrix[0])

# 存储每一层的高度

height = [0 for \_ in range(cols + 1)] res = 0

for i in range(rows): # 遍历以哪一层作为底层

stack for j

#

h

= [-1]

in range(cols + 1):

计算j位置的高度，如果遇到1则置为0，否则递增

= 0 if j == cols

or matrix[i][j] == '1' else height[j] + 1

height[j] = h

# 单调栈维护长度

while len(stack) > 1 and h < height[stack[-1]]:

res = max(res, (j - stack[-2] - 1) \* height[stack[-1]]) stack.pop()

stack.append(j)

return res

rows, \_ = map(int, input().split())

a = [input().split() for \_ in range(rows)]

print(maximalRectangle(a))

# 26977: 接雨水

<http://cs101.openjudge.cn/practice/09202/>

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。 示例：



height = [0,1,0,2,1,0,1,3,2,1,2,1]

由数组表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

## 输入

第一行包含一个整数n。1 <= n <= 2 \* 10^4 第二行包含n个整数，相邻整数间以空格隔开。0 <= ratings[i] <= 2 \* 10^5

**输出** 一个整数 样例输入

sample1 input: 12

0 1 0 2 1 0 1 3 2 1 2 1

sample1 output: 6

样例输出

sample2 6

4 2 0 3

input:

2 5

sample2 9

output:

# 用于计算接雨水问题的函数，采用了单调栈的思想

def trap6(height): total\_sum = 0 stack = [] current = 0

while current < len(height):

while stack and height[current] > height[stack[-1]]: h = height[stack[-1]]

stack.pop()

if not stack: break

distance = current - stack[-1] - 1

min\_val = min(height[stack[-1]], height[current]) total\_sum += distance \* (min\_val - h)

stack.append(current) current += 1

return total\_sum

input()

\*h, = map(int, input().split())

#h = [0,1,0,2,1,0,1,3,2,1,2,1]

ans = trap6(h) print(ans)

# 04137:最小新整数

给定一个十进制正整数n(0 < n < 1000000000)，每个数位上数字均不为0。n的位数为m。 现在从m位中 删除k位(0<k < m)，求生成的新整数最小为多少？ 例如: n = 9128456, k = 2, 则生成的新整数最小为

12456

## 输入

第一行t, 表示有t组数据； 接下来t行，每一行表示一组测试数据，每组测试数据包含两个数字n, k。

**输出** t行，每行一个数字，表示从n中删除k位后得到的最小整数。 样例输入

2

9128456 2

1444 3

样例输出

12456

1

何为单调栈？顾名思义，单调栈即满足单调性的栈结构。例如：维护一个整数的单调递增栈。

def removeKDigits(num, k): stack = []

for digit in num:

while k and stack and stack[-1] > digit: stack.pop()

k -= 1

stack.append(digit) while k:

stack.pop() k -= 1

return int(''.join(stack)) t = int(input())

results = []

for \_ in range(t):

n, k = input().split() results.append(removeKDigits(n, int(k)))

for result in results: print(result)

# 28190: 奶牛排队

现在，奶牛们想知道，如果找出一些连续的奶牛，要求最左边的奶牛 A 是最矮的，最右边的 B 是最高 的，且 B 高于 A 奶牛。中间如果存在奶牛，则身高不能和 A,B 奶牛相同。问这样的奶牛最多会有多少 头？

从左到右给出奶牛的身高，请告诉它们符合条件的最多的奶牛数（答案可能是 0,2，但不会是 1）

## 输入

第一行一个正整数 N，表示奶牛的头数。(2<=N<=10000)

接下来 N 行，每行一个正整数，从上到下表示从左到右奶牛的身高 hi (1<=hi<=50000000) 。

## 输出

一行一个整数，表示最多奶牛数。

sample input1: 5

1

2

3

4

1

sample output1: 4

#取第 1 头到第 4 头奶牛，满足条件且为最多。

sample input2: 10

15

15

2

13

7

4

11

5

11

12

sample output2: 5

利用单调栈， left\_bound用于记录以当前点为最右端，满足条件的最左端的索引减1； right\_bound用于 记录以当前节点为最左端，满足条件的最右端的索引加1，最终答案就是两段拼起来之后的最长长度。

求一个区间，使得区间左端点最矮，区间右端点最高，且区间内不存在与两端相等高度的奶牛，输出这个区间的 长度。

我们设左端点为 A ,右端点为 B

因为 A 是区间内最矮的，所以 [A.B]中，都比 A 高。所以只要 A 右侧第一个 ≤A的奶牛位于 B 的右侧， 则 A 合法

同理，因为B是区间内最高的，所以 [A.B]中，都比 B 矮。所以只要 B 左侧第一个 ≥B 的奶牛位于 A的左 侧，则 B合法

对于 “ 左/右侧第一个 ≥/≤ ” 我们可以使用单调栈维护。用单调栈预处理出 zz数组表示左，r 数组表示 右。

然后枚举右端点 B寻找 A，更新 ans 即可。 这个算法的时间复杂度为 O(n)，其中 n 是奶牛的数量。

N = int(input())

heights = [int(input()) for \_ in range(N)] left\_bound = [-1] \* N

right\_bound = [N] \* N

stack = [] # 单调栈，存储索引

# 求左侧第一个≥h[i]的奶牛位置

for i in range(N):

while stack and heights[stack[-1]] < heights[i]: stack.pop()

if stack:

left\_bound[i] = stack[-1] stack.append(i)

stack = [] # 清空栈以供寻找右边界使用

# 求右侧第一个≤h[i]的奶牛位

for i in range(N-1, -1, -1):

while stack and heights[stack[-1]] > heights[i]: stack.pop()

if stack:

right\_bound[i] = stack[-1] stack.append(i)

ans = 0

# for i in range(N-1, -1, -1): # 从大到小枚举是个技巧

# for j in range(left\_bound[i] + 1, i):

# if right\_bound[j] > i:

# ans = max(ans, i - j + 1)

# break

#

# if i <= ans:

# break

for i in range(N): # 枚举右端点 B寻找 A，更新 ans for j in range(left\_bound[i] + 1, i):

if right\_bound[j] > i:

ans = max(ans, i - j + 1) break

print(ans)