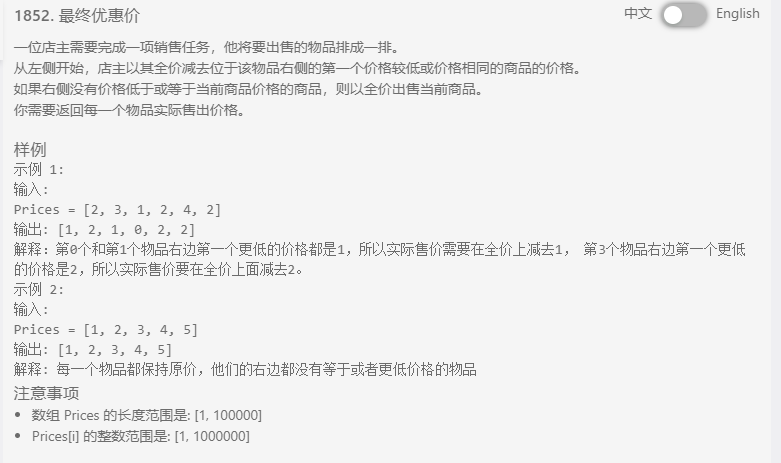
单调栈

1. 最终优惠价（leedcode1852题）【单调栈】

<https://www.jiuzhang.com/solution/final-discounted-price/#tag-highlight-lang-java>



public class Solution {

/\*\*

\* @param prices: a list of integer

\* @return: return the actual prices

\*/

public int[] FinalDiscountedPrice(int[] prices) {

// write your code here

int[] res = new int[prices.length];

Stack<Integer> s = new Stack<>();

for (int i = 0; i < prices.length; i++) {

res[i] = prices[i];

}

for (int i = 0; i < prices.length; i++) {

while (! s.isEmpty() && prices[s.peek()] >= prices[i] ) {

int index = s.pop();

res[index] = prices[index] - prices[i];

}

s.push(i);

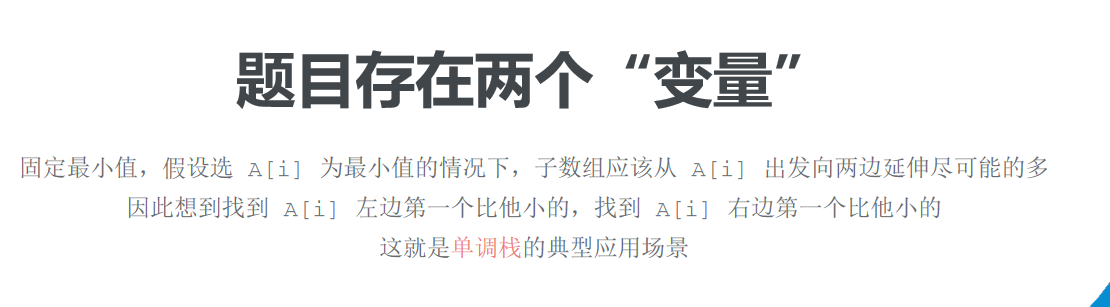
}

return res;

}

}

单调栈



1. 最大区间（lintcode1339）【单调栈】【区间中最小值\*区间所有数的和】



<https://www.jiuzhang.com/solution/maximum-range/#tag-highlight-lang-java>



public class Solution {

/\*\*

\* @param nums:

\* @return: Find the maxmum range value. A range value is defined as the minimum value times the sum of all values in a range.

\*/

public int maxRange(int[] nums) {

// write your code here

Stack<Integer> dataStack = new Stack<>();

Stack<Integer> preSum = new Stack<>();

int currSum = 0, res = 0, i = 0;

while (i < nums.length || !dataStack.isEmpty()) {

if (i < nums.length && (dataStack.isEmpty() || nums[i] >= dataStack.peek())){

dataStack.push(nums[i]);

preSum.push(currSum);

currSum = 0;

i ++;

}

else {

int leftmost = dataStack.peek();

dataStack.pop();

currSum += (leftmost + preSum.peek());

preSum.pop();

res = Math.max(res, currSum \* leftmost);

}

}

return res;

}

}

1. 逛街（腾讯2020校园招聘-后台第2题）【单调栈】



import java.util.\*;

public class Main{

public static int[] count(int n, int[] w){

int[] v = new int[n];

Stack<Integer> stack = new Stack<>();

//for (int i = 0; i < n; i++){

for (int i = n - 1; i >= 0; i--){

v[i] = stack.size();

while ((! stack.isEmpty()) && (w[i] >= w[stack.peek()])) {

stack.pop();

}

stack.push(i);

}

stack.clear();

//与原来相反的遍历，保持递减

//for (int i = n - 1; i >= 0; i--){

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

v[i] = v[i] + 1 + stack.size();

while ((! stack.isEmpty()) && (w[i] >= w[stack.peek()])) {

stack.pop();

}

stack.push(i);

}

return v;

}

public static void main(String[] args){

Scanner sc = new Scanner(System.in);

int n = sc.nextInt();

int[] w = new int[n];

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

w[i] = sc.nextInt();

}

//Main m = new Main();

//int[] v = m.count(n, w);

int[] v = count(n, w); //把函数加上static以后可以直接调用

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

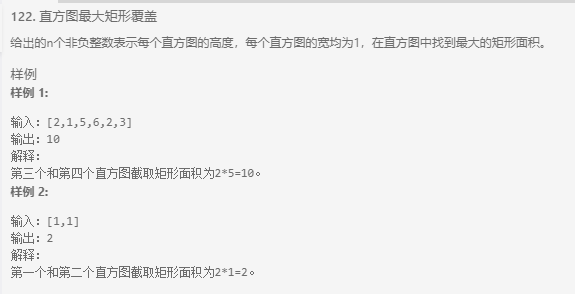
System.out.print(v[i] + " ");

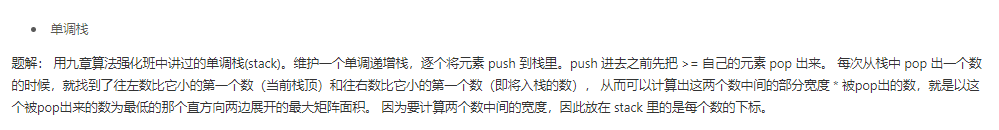
}

}

}

1. 柱状图中最大矩形（84）；直方图最大矩形覆盖（lintcode122）【单调栈】

<https://www.jiuzhang.com/solution/largest-rectangle-in-histogram/>





方法1：九章

**class** Solution {

**public** **int** largestRectangleArea(**int**[] heights) {

**if** (heights == null || heights.length == 0) **return** 0;

        Stack<Integer> sta = **new** Stack<>();

**int** max = 0;

**for** (**int** i = 0; i <= heights.length; i++){

**int** top = (i == heights.length) ? -1 : heights[i];

**while** (!sta.isEmpty() && top <= heights[sta.peek()]){

**int** h = heights[sta.pop()];

**int** w = sta.isEmpty() ? i : i - sta.peek() - 1;

                max = Math.max(max, h \* w);

            }

            sta.push(i);

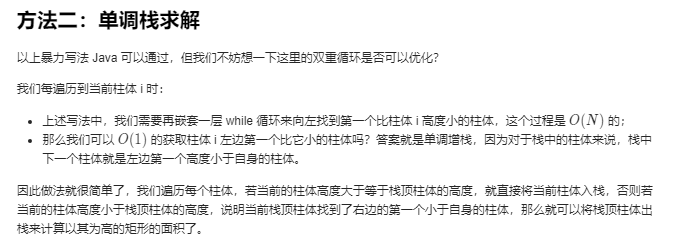
        }

**return** max;

    }

}

<https://leetcode-cn.com/problems/largest-rectangle-in-histogram/solution/xiang-jie-dan-diao-zhan-bi-xu-miao-dong-by-sweetie/>



方法2：甜姨

**class** Solution {

**public** **int** largestRectangleArea(**int**[] heights) {

*// 这里为了代码简便，在柱体数组的头和尾加了两个高度为 0 的柱体。*

**int**[] tmp = **new** **int**[heights.length + 2];

//public static native void arraycopy(Object src, int srcPos, Object dest, int destPos, int length);

        System.arraycopy(heights, 0, tmp, 1, heights.length);

        Deque<Integer> stack = **new** ArrayDeque<>();

**int** area = 0;

**for** (**int** i = 0; i < tmp.length; i++) {

*// 对栈中柱体来说，栈中的下一个柱体就是其「左边第一个小于自身的柱体」；*

*// 若当前柱体 i 的高度小于栈顶柱体的高度，说明 i 是栈顶柱体的「右边第一个小于栈顶柱体的柱体」。*

*// 因此以栈顶柱体为高的矩形的左右宽度边界就确定了，可以计算面积*

**while** (!stack.isEmpty() && tmp[i] < tmp[stack.peek()]) {

**int** h = tmp[stack.pop()];

                area = Math.max(area, (i - 1 - stack.peek()) \* h);

            }

            stack.push(i);

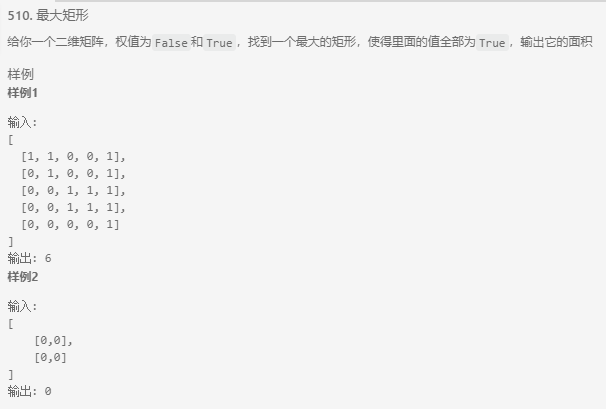
        }

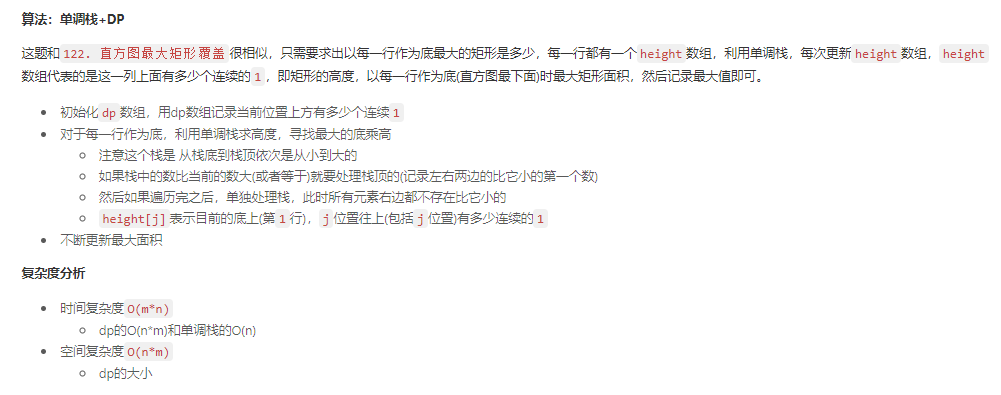
**return** area;

    }

}

1. 最大矩形（85）（lintcode510）【单调栈+DP】





Lintcode510代码

public class Solution {

/\*\*

\* @param matrix: a boolean 2D matrix

\* @return: an integer

\*/

public int maximalRectangle(**boolean[][] matrix**) {

if (matrix == null || matrix.length == 0 || matrix[0].length == 0) {

return 0;

}

int ans = 0;

int row = matrix.length, col = matrix[0].length;

int[][] dp = new int[row][col+1];

for (int i = 0; i < row; i++) {//每个位置上方有多少连续的1

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

if (i == 0 && **matrix[i][j]**) {

dp[i][j] = 1;

continue;

}

if (**matrix[i][j]**) {

dp[i][j] = dp[i - 1][j] + 1;

}

}

}

for (int i = 0; i < row; i++) { //把每一行作为底找最大矩形

ans = Math.max(ans, largestRectangleArea(dp[i]));

}

return ans;

}

private int largestRectangleArea(int[] height) {

int[] tmp = new int[height.length + 2];

System.arraycopy(height, 0, tmp, 1, height.length);

Deque<Integer> sta = new ArrayDeque<>();

int area = 0;

for (int i = 0; i < tmp.length; i++){

while (!sta.isEmpty() && tmp[i] < tmp[sta.peek()]){

int h = tmp[sta.pop()];

area = Math.max(area, (i - sta.peek() - 1) \* h);

}

sta.push(i);

}

return area;

}

}

与在leetcode中的不同用加粗表示。以下是leetcode85代码

**class** Solution {

**public** **int** maximalRectangle(char[][] matrix) {

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

**int** res = 0;

**int** row = matrix.length, col = matrix[0].length;

**int**[][] dp = **new** **int**[row][col+1];

**for** (**int** i = 0; i < row; ++i){

**for** (**int** j = 0; j < col; ++j){

**if** (i == 0 && **matrix[i][j] == '1'**){

                    dp[i][j] = 1;

**continue**;

                }

**if** (**matrix[i][j] == '1'**){

                    dp[i][j] = dp[i-1][j] + 1;

                }

            }

        }

**for** (**int** i = 0; i < row; ++i){

            res = Math.max(res, largeArea(dp[i]));

        }

**return** res;

    }

**public** **int** largeArea(**int**[] height){

**int**[] tmp = **new** **int**[height.length + 2];

        System.arraycopy(height, 0, tmp, 1, height.length);

        Deque<Integer> sta = **new** ArrayDeque<>();

**int** area = 0;

**for** (**int** i = 0; i < tmp.length; i++){

**while** (!sta.isEmpty() && tmp[i] < tmp[sta.peek()]){

**int** h = tmp[sta.pop()];

                area = Math.max(area, (i - sta.peek() - 1) \* h);

            }

            sta.push(i);

        }

**return** area;

    }

}

1. 最大正方形（221）（lintcode436）【单调的+DP，上一题改造了两行】【只DP】



方法1：单调栈+DP

**class** Solution {

**public** **int** maximalSquare(**char**[][] matrix) {

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

**int** res = 0;

**int** row = matrix.length, col = matrix[0].length;

**int**[][] dp = **new** **int**[row][col+1];

**for** (**int** i = 0; i < row; ++i){

**for** (**int** j = 0; j < col; ++j){

**if** (i == 0 && matrix[i][j] == '1'){

                    dp[i][j] = 1;

**continue**;

                }

**if** (matrix[i][j] == '1'){

                    dp[i][j] = dp[i-1][j] + 1;

                }

            }

        }

**for** (**int** i = 0; i < row; ++i){

            res = Math.max(res, largeArea(dp[i]));

        }

**return** res;

    }

**public** **int** largeArea(**int**[] height){

**int**[] tmp = **new** **int**[height.length + 2];

        System.arraycopy(height, 0, tmp, 1, height.length);

        Deque<Integer> sta = **new** ArrayDeque<>();

**int** area = 0;

**for** (**int** i = 0; i < tmp.length; i++){

**while** (!sta.isEmpty() && tmp[i] < tmp[sta.peek()]){

*//这边找到高h，宽w里面最小的作为正方形的边长*

**int** h = tmp[sta.pop()];

**int** w = i - sta.peek() - 1;

**int** min = Math.min(h, w);

                area = Math.max(area, min \* min);

            }

            sta.push(i);

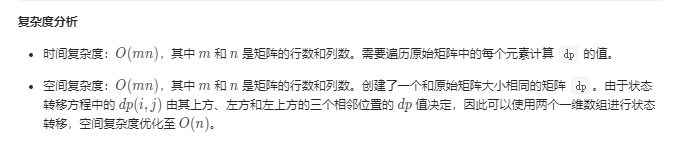
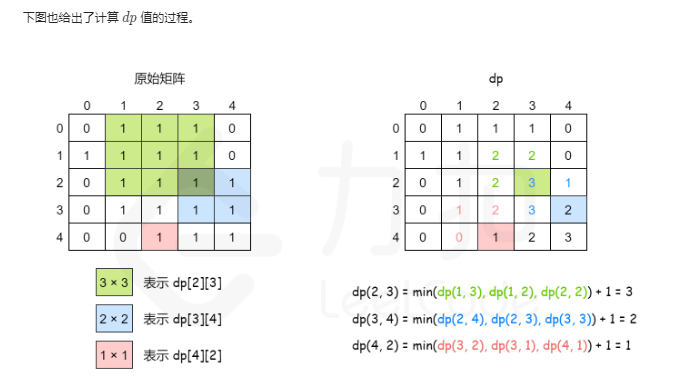
        }

**return** area;

    }

}

https://leetcode-cn.com/problems/maximal-square/solution/zui-da-zheng-fang-xing-by-leetcode-solution/



方法2：只用动规

**class** Solution {

**public** **int** maximalSquare(**char**[][] matrix) {

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

**int** res = 0, len = 0;

**int** row = matrix.length, col = matrix[0].length;

**int**[][] dp = **new** **int**[row][col];

**for** (**int** i = 0; i < row; ++i){

**for** (**int** j = 0; j < col; ++j){

**if** (matrix[i][j] == '1'){

**if** (i == 0 || j == 0){

                        dp[i][j] = 1;

                    } **else** {

                        dp[i][j] = Math.min(Math.min(dp[i-1][j], dp[i][j-1]), dp[i-1][j-1]) + 1;

                    }

                    len = Math.max(len, dp[i][j]);

                }

            }

        }

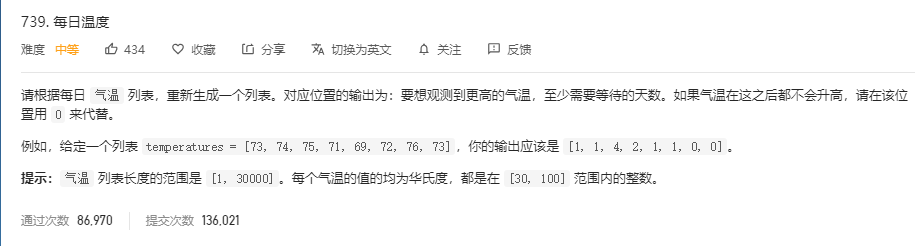
        res = len \* len;

**return** res;

    }

}

1. 每日温度（739）（lintcode1060）【方法2是单调栈】



<https://leetcode-cn.com/problems/daily-temperatures/solution/java-by-sdwwld/>

方法1：从后面开始找，效率比较高（4ms）

**class** Solution {

**public** **int**[] dailyTemperatures(**int**[] T) {

**int**[] res = **new** **int**[T.length];

*//从后面开始查找*

**for** (**int** i = res.length - 1; i >= 0; i--) {

**int** j = i + 1;

**while** (j < res.length) {

**if** (T[j] > T[i]) {

*//如果找到就停止while循环*

                    res[i] = j - i;

**break**;

                } **else** **if** (res[j] == 0) {

*//如果没找到，并且res[j]==0。说明第j个元素后面没有*

*//比第j个元素大的值，因为这一步是第i个元素大于第j个元素的值，*

*//那么很明显这后面就更没有大于第i个元素的值。直接终止while循环。*

**break**;

                } **else** {

*//如果没找到，并且res[j]！=0说明第j个元素后面有比第j个元素大的值，*

*//然后我们让j往后挪res[j]个单位，找到那个值，再和第i个元素比较*

                    j += res[j];

                }

            }

        }

**return** res;

    }

}

方法2：单调栈（17ms）

**class** Solution {

**public** **int**[] dailyTemperatures(**int**[] T) {

**int** len = T.length;

        Deque<Integer> stack = **new** ArrayDeque<>();

**int**[] res = **new** **int**[len];

**for** (**int** i = 0; i < len; i++){

**int** h = T[i];

**if** (stack.isEmpty() || h **<=** T[stack.peek()]){

                stack.push(i);

            } **else** {

**int** top = stack.pop();

                res[top] = i - top;

                i--;

            }

        }

**return** res;

    }

}

1. 股票价格跨度（901）（lintcode1740）【单调栈】



单调栈问题 题目中提到股票价格小于或等于今天价格的最大连续日数。 由于这是一个在线问题，所以我们必然是要将输入的price给存储起来，而且同时我们也需要保留这是第几次输入的信息。 需要注意的是边界问题，当我们输入第一个price的时候，此时stack空。 所以这里拿出来判断特殊处理一下。

**class** StockSpanner {

**public** StockSpanner() {

    }

**private** Stack<**int**[]> stack = **new** Stack<>();

**public** **int** next(**int** price) {

**int** res = 1;

**while** (!stack.isEmpty() && stack.peek()[0] <= price)

            res += stack.pop()[1];

        stack.push(**new** **int**[]{price, res});

**return** res;

    }

}

*/\*\**

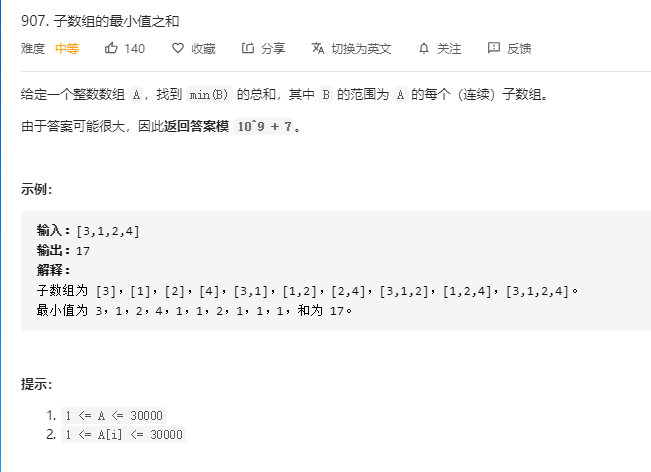
*\* Your StockSpanner object will be instantiated and called as such:*

*\* StockSpanner obj = new StockSpanner();*

*\* int param\_1 = obj.next(price);*

*\*/*

1. 子数组的最小值之和（907）（lintcode1734）【单调栈】



单调栈解决； 假设从数组的头部遍历，如果后面的数大于前面的数，那么这两个数构成的子数组取得肯定是前面的小的数； 如果后面的数小于前面的数，那么子数组就得取后面的数。 那么当前元素的贡献就是： Ai 与其前一个较少的元素之间的距离 Ai与其下一个较小元素之间的距离将所有元素的贡献相加即可。

**class** Solution {

**public** **int** sumSubarrayMins(**int**[] A) {

        Deque<Integer> stack = **new** ArrayDeque<>();

**int** n = A.length, res = 0, mod = (**int**) 1e9 + 7;

**for** (**int** i = 0; i <= n; i++){

**int** cur = (i == n) ? 0 : A[i];

**while** (!stack.isEmpty() && A[stack.peek()] > cur){

**int** j = stack.pop();

**int** k = stack.isEmpty() ? -1 : stack.peek();

                res = (res + A[j] \* (i - j) \* (j - k)) % mod;

            }

            stack.push(i);

        }

**return** res;

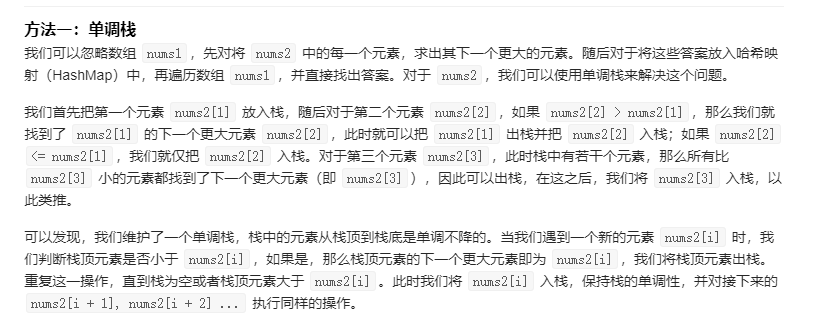
    }

}

1. 下一个更大元素1（496）【单调栈】



<https://leetcode-cn.com/problems/next-greater-element-i/solution/xia-yi-ge-geng-da-yuan-su-i-by-leetcode/>



**class** Solution {

**public** **int**[] nextGreaterElement(**int**[] nums1, **int**[] nums2) {

        Deque<Integer> stack = **new** ArrayDeque<>();

        HashMap<Integer, Integer> map =  **new** HashMap<>();

**int**[] res = **new** **int**[nums1.length];

**for** (**int** i = 0; i < nums2.length; i++){

**while** (!stack.isEmpty() && nums2[i] > stack.peek()){

                map.put(stack.pop(), nums2[i]);

            }

            stack.push(nums2[i]);

        }

**while** (!stack.isEmpty()){

            map.put(stack.pop(), -1);

        }

**for** (**int** i = 0; i < nums1.length; i++){

            res[i] = map.get(nums1[i]);

        }

**return** res;

    }

}

1. 去除重复字母（316）【单调栈】



class Solution {

    public String removeDuplicateLetters(String s) {

        if (s == null || s.length() < 2) return s;

        Deque<Character> stack = new ArrayDeque<>();

        char[] ch = s.toCharArray();

        StringBuilder res = new StringBuilder();

        for (int i = 0; i < ch.length; i++){

            if (!stack.isEmpty() && stack.contains(ch[i])) continue;

            while (!stack.isEmpty() && ch[i] < stack.peekLast() && s.indexOf(stack.peekLast(), i) > 0){

                stack.removeLast();

            }

            stack.addLast(ch[i]);

        }

        for (Character cur : stack){

            res.append(cur);

        }

        return res.toString();

    }

}

1. 不同字符的最小子序列（1081）【单调栈，与上一题的意思一样】



class Solution {

    public String smallestSubsequence(String text) {

        if (text == null || text.length() < 2) return text;

        Deque<Character> stack = new ArrayDeque<>();

        char[] ch = text.toCharArray();

        StringBuilder res = new StringBuilder();

        for (int i = 0; i < ch.length; i++){

            if (!stack.isEmpty() && stack.contains(ch[i])) continue;

            while (!stack.isEmpty() && ch[i] < stack.peekLast() && text.indexOf(stack.peekLast(), i) > 0){

                stack.removeLast();

            }

            stack.addLast(ch[i]);

        }

        for (Character cur : stack){

            res.append(cur);

        }

        return res.toString();

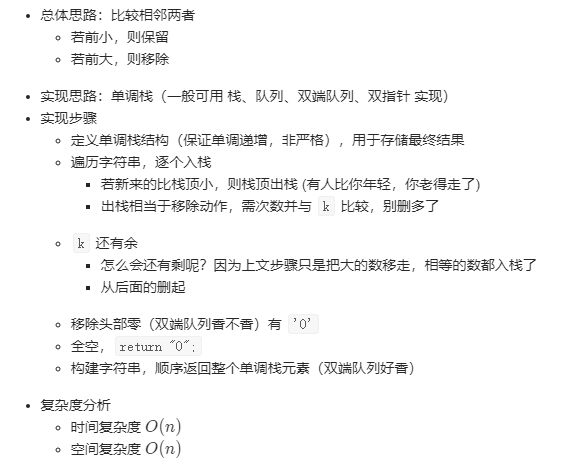
    }

}

1. 移掉k位数字（402）【单调栈】【双指针】



<https://leetcode-cn.com/problems/remove-k-digits/solution/java-dan-diao-dui-lie-shuang-zhi-zhen-by-lzhlyle/>



方法1：单调栈

class Solution {

    public String removeKdigits(String num, int k) {

        Deque<Character> stack = new ArrayDeque<>();

        char[] ch = num.toCharArray();

        // 遍历，逐个入队

        for (char c : ch) {

            while (!stack.isEmpty() && k > 0 && stack.getLast() > c){

                k--;

                stack.removeLast();

            }

            stack.addLast(c);

        }

        // k 还有余，后面大的删起

        while (k > 0 && !stack.isEmpty()){

            k--;

            stack.removeLast();

        }

        //移除头部零

        while (!stack.isEmpty() && stack.getFirst() == '0'){

            stack.removeFirst();

        }

        if (stack.isEmpty()) return "0";

        // 构建，返回

        StringBuilder res = new StringBuilder();

        for (Character tmp : stack){

            res.append(tmp);

        }

        return res.toString();

    }

}

方法2：双指针

class Solution {

    public String removeKdigits(String num, int k) {

        int r = 0, n = num.length();

        char[] res = new char[n];

        // 遍历，逐个入数组

        for (char c : num.toCharArray()) {

            while (r > 0 && k > 0 && res[r - 1] > c) {

                k--;

                r--;

            }

            res[r++] = c;

        }

        // k 还有余，后面大的删起

        while (k-- > 0 && r > 0) r--;

        // 移除头部零

        int l = 0; // 此时才需要 left 指针

        while (l < r && res[l] == '0') l++;

        if (l == r) return "0";

        // 构建，返回

        return new String(res, l, r - l);

    }

}

1. 拼接最大数（321）



<https://leetcode-cn.com/problems/create-maximum-number/solution/qiu-jie-zi-wen-ti-zai-he-bing-zi-wen-ti-de-jie-by-/>

public class Solution {

    public int[] maxNumber(int[] nums1, int[] nums2, int k) {

        int len1 = nums1.length, len2 = nums2.length;

        int[] res = new int[k];

        for (int i = Math.max(0, k - len2); i <= len1 && i <= k; i++)        {

            int[] arr = mergeMaxArr(maxArr(nums1, i), maxArr(nums2, k-i), k);

            if(compare(arr, res, k)) res = arr;

        }

        return res;

    }

    //从num数组中找pos个数（按相对顺序）组建成新数组（可以不连续），使该数组构成的数最大

    private int[] maxArr(int[] num,int pos){

        int len = num.length;

        if (len == pos) return num;

        int[] res = new int[pos];

        int top = -1;

        for (int cur = 0; cur < len; cur++){

            while (top + 1 > 0 && num[cur] > res[top] && top + 1 + len - cur > pos)

                top--;

            if(top < pos - 1)

                res[++top] = num[cur];

        }

        return res;

    }

    //与归并类似，但是这个排序后的结果不改变元素在原数组的相对顺序

    private int[] mergeMaxArr(int[] arr1,int[] arr2,int pos){

        int p1=0,p2=0;

        int[] res=new int[pos];

        int index=0;

        while (index<pos){

            if (compare(arr1,arr2,p1,p2))

                res[index++]=arr1[p1++];

            else res[index++]=arr2[p2++];

        }

        return res;

    }

    private boolean compare(int[] arr1,int[] arr2,int k){

        int p = 0;

        while (p < k && arr1[p] == arr2[p])

            p++;

        return p == k || arr1[p] >= arr2[p];

    }

    private boolean compare(int[] arr1,int[] arr2,int p1,int p2){

        int len1 = arr1.length, len2 = arr2.length;

        while (p1 < len1 && p2 < len2 && arr1[p1] == arr2[p2]){

            p1++;

            p2++;

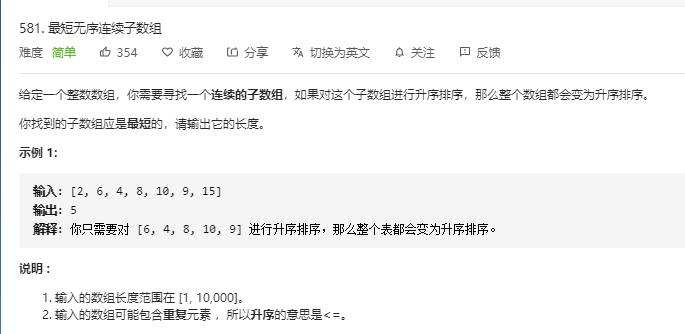
        }

        return p2 == len2 || (p1 < len1 && arr1[p1] > arr2[p2]);

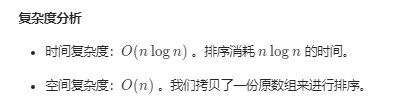
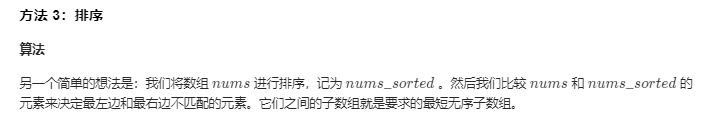
    }

}

1. 最短无序连续子数组（581）【内置函数】【单调栈】【双指针】



<https://leetcode-cn.com/problems/shortest-unsorted-continuous-subarray/solution/zui-duan-wu-xu-lian-xu-zi-shu-zu-by-leetcode/>



方法1：用内置排序（8ms）

**class** Solution {

**public** **int** findUnsortedSubarray(**int**[] nums) {

**int**[] arr = nums.clone();

        Arrays.sort(arr);

**int** start = arr.length, end = 0;

**for** (**int** i = 0; i < arr.length; i++) {

**if** (arr[i] != nums[i]) {

                start = Math.min(start, i);

                end = Math.max(end, i);

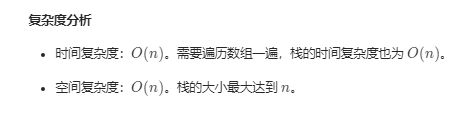
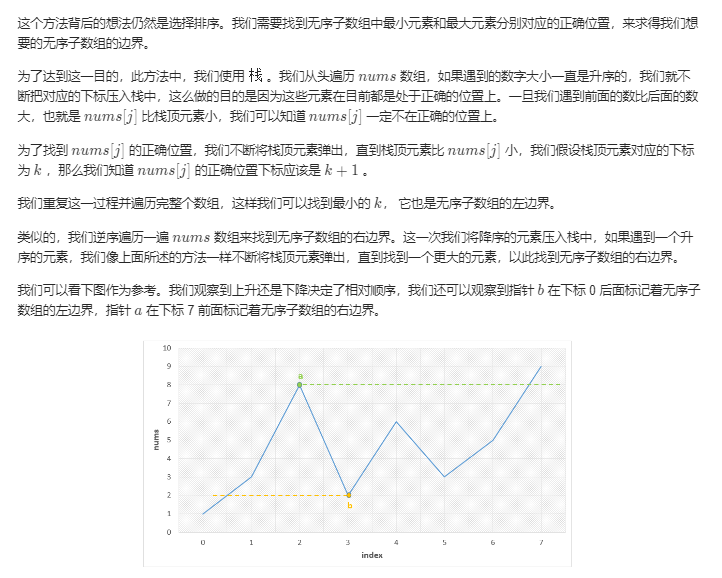
            }

        }

**return** (end - start) >= 0 ? end - start + 1 : 0;

    }

}



方法2：单调栈（10s）

**class** Solution {

**public** **int** findUnsortedSubarray(**int**[] nums) {

        Deque<Integer> stack = **new** ArrayDeque<>();

**int** left = nums.length, right = 0;

**for** (**int** i = 0; i < nums.length; i++){

**while** (!stack.isEmpty() && nums[stack.peek()] > nums[i])

                left = Math.min(left, stack.pop());

            stack.push(i);

        }

        stack.clear();

**for** (**int** i = nums.length - 1; i >= 0; i--) {

**while** (!stack.isEmpty() && nums[stack.peek()] < nums[i])

                right = Math.max(right, stack.pop());

            stack.push(i);

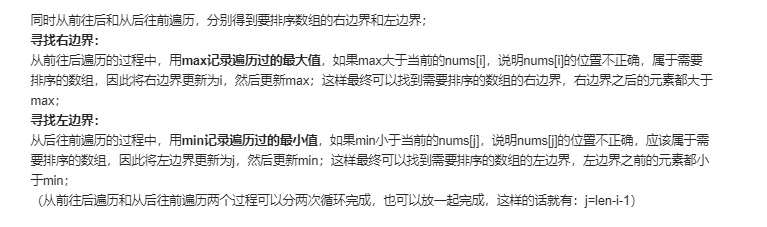
        }

**return** right - left > 0 ? right - left + 1 : 0;

    }

}

方法3：双指针，同时遍历



**class** Solution {

**public** **int** findUnsortedSubarray(**int**[] nums) {

**int** len = nums.length;

**int** max = nums[0], min = nums[len - 1];

**int** left = 0, right = -1;

**for** (**int** i = 0; i < len; i++) {

**if** (max > nums[i]) right = i;

**else** max = nums[i];

**if** (min < nums[len - 1 - i]) left = len - 1 - i;

**else** min = nums[len - 1 - i];

        }

**return** right - left + 1;

    }

}