王争的算法训练营

习题课：栈

王争的算法训练营

习题课：栈

配套习题（12+4）：

剑指 Offer 09. 用两个栈实现队列（简单） （已讲）

225. 用队列实现栈（简单）

面试题 03.05. 栈排序（中等）

155. 最小栈（简单）

面试题 03.01. 三合一（简单）

20. 有效的括号（简单）

面试题 16.26. 计算器（中等） （已讲）

772. 基本计算器 III（困难 力扣会员，比上一题多了括号）

1047. 删除字符串中的所有相邻重复项（简单）

剑指 Offer 31. 栈的压入、弹出序列（中等）

739. 每日温度（中等） 单调栈 （已讲）

42. 接雨水（困难）单调栈

以下选做，留给精力充沛的同学自刷

84. 柱状图中最大的矩形（困难）单调栈

面试题 03.06. 动物收容所（中等） 队列

剑指 Offer 59 - II. 队列的最大值（中等） 单调队列

剑指 Offer 59 - I. 滑动窗口的最大值 （困难）单调队列

习题课：栈

剑指 Offer 09. 用两个栈实现队列（简单） （已讲）

用两个栈实现一个队列。队列的声明如下，请实现它的两个函数 `appendTail` 和 `deleteHead` ，分别完成在队列尾部插入整数和在队列头部删除整数的功能。(若队列中没有元素， `deleteHead` 操作返回 -1)

示例 1：

```
输入：
["CQueue","appendTail","deleteHead","deleteHead"]
[[],[3],[],[]]
输出：[null,null,3,-1]
```

示例 2：

```
输入：
["CQueue","deleteHead","appendTail","appendTail","deleteHead","deleteHead"]
[[],[],[5],[2],[],[]]
输出：[null,-1,null,null,5,2]
```

提示：

- `1 <= values <= 10000`
- `最多会对 appendTail、deleteHead 进行 10000 次调用`

习题课：栈

剑指 Offer 09. 用两个栈实现队列（简单） （已讲）

```
class CQueue {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> tmpStack = new Stack<>();

    public CQueue() {}

    public void appendTail(int value) {
        stack.push(value);
    }

    public int deleteHead() {
        if (stack.empty()) return -1;
        while (!stack.empty()) {
            tmpStack.push(stack.pop());
        }
        int result = tmpStack.pop();
        while (!tmpStack.empty()) {
            stack.push(tmpStack.pop());
        }
        return result;
    }
}
```

习题课：栈

剑指 Offer 09. 用两个栈实现队列（简单） （已讲）

```
class CQueue {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> tmpStack = new Stack<>();

    public CQueue() {}

    public void appendTail(int value) {
        while (!stack.empty()) {
            tmpStack.push(stack.pop());
        }
        stack.push(value);
        while (!tmpStack.empty()) {
            stack.push(tmpStack.pop());
        }
    }

    public int deleteHead() {
        if (stack.empty()) return -1;
        return stack.pop();
    }
}
```

王争的算法训练营

习题课：栈

225. 用队列实现栈（简单）

请你仅使用两个队列实现一个后入先出（LIFO）的栈，并支持普通队列的全部四种操作（`push`、`top`、`pop` 和 `empty`）。

实现 `MyStack` 类：

- `void push(int x)` 将元素 x 压入栈顶。
- `int pop()` 移除并返回栈顶元素。
- `int top()` 返回栈顶元素。
- `boolean empty()` 如果栈是空的，返回 `true` ；否则，返回 `false` 。

注意：

- 你只能使用队列的基本操作 —— 也就是 `push to back`、`peek/pop from front`、`size` 和 `is empty` 这些操作。
- 你所使用的语言也许不支持队列。 你可以使用 list （列表）或者 deque（双端队列）来模拟一个队列 , 只要是标准的队列操作即可。

习题课：栈

225. 用队列实现栈（简单）

用栈实现队列：

1. 需要用两个栈
2. push、**pop**、**peek**、isEmpty()

```
class MyStack {

    /** Initialize your data structure here. */
    public MyStack() {}

    /** Push element x onto stack. */
    public void push(int x) {}

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {}

    /** Get the top element. */
    public int top() {}

    /** Returns whether the stack is empty. */
    public boolean empty() {}
}
```

解法一：

push直接塞

pop、peek倒腾

```java
class MyStack {
    Queue<Integer> queue;
    /** Initialize your data structure here. */
    public MyStack() {
        queue = new LinkedList<Integer>();
    }

    /** Push element x onto stack. */
    public void push(int x) {
        queue.offer(x);
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        int n = queue.size();
        for (int i = 0; i < n-1; i++) {
            queue.offer(queue.poll());
        }
        return queue.poll();
    }

    /** Get the top element. */
    public int top() {
        int n = queue.size();
        for (int i = 0; i < n-1; i++) {
            queue.offer(queue.poll());
        }
        int ret = queue.poll();
        queue.offer(ret);
        return ret;
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
        return queue.isEmpty();
    }
}
```

算法训练营

解法二:
push倒腾
pop、peek直接取

```
class MyStack {
    Queue<Integer> queue= new LinkedList<Integer>();
    /** Initialize your data structure here. */
    public MyStack() {}

    /** Push element x onto stack. */
    public void push(int x) {
        int n = queue.size();
        queue.offer(x);
        for (int i = 0; i < n; i++) {
            queue.offer(queue.poll());
        }
    }

    /** Removes the element on top of the stack and returns that element. */
    public int pop() {
        return queue.poll();
    }

    /** Get the top element. */
    public int top() {
        return queue.peek();
    }

    /** Returns whether the stack is empty. */
    public boolean empty() {
        return queue.isEmpty();
    }
}
```

王争的算法训练营

习题课：栈

面试题 03.05. 栈排序（中等）

栈排序。 编写程序，对栈进行排序使最小元素位于栈顶。最多只能使用一个其他的临时栈存放数据，但不得将元素复制到别的数据结构（如数组）中。该栈支持如下操作：`push` 、`pop` 、`peek` 和 `isEmpty` 。当栈为空时，`peek` 返回 -1。

示例1:

```
 输入:
["SortedStack", "push", "push", "peek", "pop", "peek"]
[[], [1], [2], [], [], []]
 输出:
[null,null,null,1,null,2]
```

套路：两个栈之间倒腾

解法一：push的时候直接塞，pop、peek的时候倒腾

解法二：类似插入排序，一直让栈中的元素从大到小有序的（从栈底到栈顶）

解法一

```
class SortedStack {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> tmpStack = new Stack<>();

    public SortedStack() {}

    public void push(int val) {
        stack.push(val);
    }

    public void pop() {
        if (stack.isEmpty()) return;
        int minVal = Integer.MAX_VALUE;
        while (!stack.isEmpty()) {
            int val = stack.pop();
            if (val < minVal) minVal = val;
            tmpStack.push(val);
        }
        boolean removed = false;
        while (!tmpStack.isEmpty()) {
            int val = tmpStack.pop();
            if ((val != minVal) ||
                (val == minVal && removed==true)) {
                stack.push(val);
            } else {
                removed = true;
            }
        }
    }

    public int peek() {
        if (stack.isEmpty()) return -1;
        int minVal = Integer.MAX_VALUE;
        while (!stack.isEmpty()) {
            int val = stack.pop();
            if (val < minVal) minVal = val;
            tmpStack.push(val);
        }
        while (!tmpStack.isEmpty()) {
            int val = tmpStack.pop();
            stack.push(val);
        }
        return minVal;
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }
}
```

王道的算法训练营

解法二

```
class SortedStack {
    private Stack<Integer> stack = new Stack<>();
    private Stack<Integer> tmpStack = new Stack<>();

    public SortedStack() {}

    public void push(int val) {
        while (!stack.isEmpty() && stack.peek() < val) {
            tmpStack.push(stack.pop());
        }
        stack.push(val);
        while (!tmpStack.isEmpty()) {
            stack.push(tmpStack.pop());
        }
    }

    public void pop() {
        if (!stack.isEmpty()) {
            stack.pop();
        }
    }

    public int peek() {
        if (stack.isEmpty()) return -1;
        return stack.peek();
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }
}
```

王争的算法训练营

习题课：栈

155. 最小栈（简单）

设计一个支持 `push` ， `pop` ， `top` 操作，并能在常数时间内检索到最小元素的栈。

- `push(x)` —— 将元素 x 推入栈中。
- `pop()` —— 删除栈顶的元素。
- `top()` —— 获取栈顶元素。
- `getMin()` —— 检索栈中的最小元素。

1. 倒腾：无法做到O(1)时间复杂度
2. ？？

解释:

```
MinStack minStack = new MinStack();
minStack.push(-2);
minStack.push(0);
minStack.push(-3);
minStack.getMin();   --> 返回 -3.
minStack.pop();
minStack.top();      --> 返回 0.
minStack.getMin();   --> 返回 -2.
```

```java
class MinStack {
    private Stack<Integer> data = new Stack<>();
    private Stack<Integer> minval = new Stack<>();

    /** initialize your data structure here. */
    public MinStack() {}

    public void push(int x) {
        if (data.empty()) {
            data.push(x);
            minval.push(x);
        } else {
            int curminval = minval.peek();
            if (x < curminval) {
                minval.push(x);
            } else {
                minval.push(curminval);
            }
            data.push(x);
        }
    }

    public void pop() {
        data.pop();
        minval.pop();
    }

    public int top() {
        return data.peek();
    }

    public int getMin() {
        return minval.peek();
    }
}
```

王争的算法训练营

习题课：栈

面试题 03.01. 三合一（简单）

三合一。描述如何只用一个数组来实现三个栈。

你应该实现 `push(stackNum, value)` 、 `pop(stackNum)` 、 `isEmpty(stackNum)` 、 `peek(stackNum)` 方法。 `stackNum` 表示栈下标， `value` 表示压入的值。

构造函数会传入一个 `stackSize` 参数，代表每个栈的大小。

```
class TripleInOne {
    public TripleInOne(int stackSize) {}

    public void push(int stackNum, int value) {}

    public int pop(int stackNum) {}

    public int peek(int stackNum) {}

    public boolean isEmpty(int stackNum) {}
}
```

```
class TripleInOne {

    private int[] array;
    private int n;
    private int[] top; //保存每个栈的栈顶下标

    public TripleInOne(int stackSize) {
        array = new int[3*stackSize];
        n = 3*stackSize;
        top = new int[3];
        top[0] = -3;
        top[1] = -2;
        top[2] = -1;
    }

    public void push(int stackNum, int value) {
        if (top[stackNum] + 3 >= n) {
            return;
        }
        top[stackNum] += 3;
        array[top[stackNum]] = value;
    }

    public int pop(int stackNum) {
        if (top[stackNum] < 0) {
            return -1;
        }
        int ret = array[top[stackNum]];
        top[stackNum] -= 3;
        return ret;
    }

    public int peek(int stackNum) {
        if (top[stackNum] < 0) {
            return -1;
        }
        return array[top[stackNum]];
    }

    public boolean isEmpty(int stackNum) {
        return top[stackNum] < 0;
    }
}
```

王争的算法训练营

20. 有效的括号（简单）

给定一个只包括 `'('`，`')'`，`'{'`，`'}'`，`'['`，`']'` 的字符串 `s`，判断字符串是否有效。

有效字符串需满足：

1. 左括号必须用相同类型的右括号闭合。
2. 左括号必须以正确的顺序闭合。

消消乐/连连消问题

示例 1：

```
输入：s = "()"
输出：true
```

示例 2：

```
输入：s = "()[]{}"
输出：true
```

王争的算法训练营

```java
class Solution {
    public boolean isValid(String s) {
        Stack<Character> stack = new Stack<>();
        for (int i = 0; i < s.length(); ++i) {
            char c = s.charAt(i);
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            } else { //右括号
                if (stack.empty()) return false;
                char popC = stack.pop();
                if (c == ')' && popC != '(') {
                    return false;
                }
                if (c == ']' && popC != '[') {
                    return false;
                }
                if (c == '}' && popC != '{') {
                    return false;
                }
            }
        }
        return stack.empty();
    }
}
```

王争的算法训练营

习题课：栈

面试题 16.26. 计算器（中等） （已讲）

给定一个包含正整数、加(+)、减(-)、乘(*)、除(/)的算数表达式(括号除外)，计算其结果。

表达式仅包含非负整数，`+`， `-`， `*`， `/` 四种运算符和空格 ` `。 整数除法仅保留整数部分。

示例 1:

```
输入: "3+2*2"
输出: 7
```

示例 2:

```
输入: " 3/2 "
输出: 1
```

示例 3:

```
输入: " 3+5 / 2 "
输出: 5
```

王争的算法课堂

```java
class Solution {
    public int calculate(String s) {
        Stack<Integer> nums = new Stack<>();
        Stack<Character> ops = new Stack<>();
        int i = 0;
        int n = s.length();
        while (i < n) {
            char c = s.charAt(i);
            if (c == ' ') { // 跳过空格
                i++;
            } else if (isDigit(c)) { //数字
                int number = 0;
                while (i < n && isDigit(s.charAt(i))) {
                    number = number*10+(s.charAt(i)-'0');
                    i++;
                }
                nums.push(number);
            } else { // 运算符
                if (ops.isEmpty() || prior(c, ops.peek())) {
                    ops.push(c);
                } else {
                    while (!ops.isEmpty() && !prior(c, ops.peek())) {
                        fetchAndCal(nums, ops);
                    }
                    ops.push(c);
                }
                i++;
            }
        }

        while (!ops.isEmpty()) {
            fetchAndCal(nums, ops);
        }

        return nums.pop();
    }
```

王争的算法课堂

```
    private boolean prior(char a, char b) {
        if ((a == '*' || a == '/')
            && (b == '+' || b == '-')) {
                return true;
        }
        return false;
    }

    private int cal(char op, int number1, int number2) {
        switch(op) {
            case '+': return number1+number2;
            case '-': return number1-number2;
            case '*': return number1*number2;
            case '/': return number1/number2;
        }
        return -1;
    }

    private boolean isDigit(char c) {
        return c >= '0' && c <= '9';
    }

    private void fetchAndCal(Stack<Integer> nums, Stack<Character> ops) {
        int number2 = nums.pop();
        int number1 = nums.pop();
        char op = ops.pop();
        int ret = cal(op, number1, number2);
        nums.push(ret);
    }

}
```

习题课：栈

王争的算法训练营

772. 基本计算器 III（困难 包含括号 力扣会员）

实现一个基本的计算器来计算简单的表达式字符串。

表达式字符串只包含非负整数，算符 `+`、`-`、`*`、`/`，左括号 `(` 和右括号 `)` 。整数除法需要 **向下截断** 。

你可以假定给定的表达式总是有效的。所有的中间结果的范围为 $[-2^{31}, 2^{31} - 1]$ 。

示例 1：

```
输入：s = "1+1"
输出：2
```

示例 2：

```
输入：s = "6-4/2"
输出：4
```

示例 3：

```
输入：s = "2*(5+5*2)/3+(6/2+8)"
输出：21
```

示例 4：

```
输入：s = "(2+6*3+5-(3*14/7+2)*5)+3"
输出：-12
```

王争的算法训练营

```java
class Solution {
    public int calculate(String s) {
        Stack<Integer> nums = new Stack<>();
        Stack<Character> ops = new Stack<>();
        int i = 0;
        int n = s.length();
        while (i < n) {
            char c = s.charAt(i);
            if (c == ' ') { // 跳过空格
                i++;
            } else if (isDigit(c)) { //数字
                int number = 0;
                while (i < n && isDigit(s.charAt(i))) {
                    number = number*10+(s.charAt(i)-'0');
                    i++;
                }
                nums.push(number);
            } else if (c == '(') {
                ops.push(c);
                i++;
            } else if (c == ')') {
                while (!ops.isEmpty() && ops.peek() != '(') {
                    fetchAndCal(nums, ops);
                }
                ops.pop(); // 弹出'('
                i++;
            } else { // 运算符
                if (ops.isEmpty() || prior(c, ops.peek())) {
                    ops.push(c);
                } else {
                    while (!ops.isEmpty() && !prior(c, ops.peek())) {
                        fetchAndCal(nums, ops);
                    }
                    ops.push(c);
                }
                i++;
            }
        }
        while (!ops.isEmpty()) {
            fetchAndCal(nums, ops);
        }
        return nums.pop();
    }
```

1. 数字直接入栈
2. 运算符：
   1）栈空 或者 c > 栈顶运算符　c入栈
   2）c <= 栈顶运算符　出栈计算
3. 如果是’(’，直接入栈
4. 如果是’)’，出栈计算，直到碰到’(’为止

王道的算法训练营

```java
   private void fetchAndCal(Stack<Integer> nums, Stack<Character> ops) {
        int number2 = nums.pop();
        int number1 = nums.pop();
        char op = ops.pop();
        int ret = cal(op, number1, number2);
        nums.push(ret);
    }

    private boolean prior(char a, char b) {
        if ((a == '*' || a == '/')
           && (b == '+' || b == '-')) {
               return true;
        }
        if (b == '(') return true;
        return false;
    }

    private int cal(char op, int number1, int number2) {
        switch(op) {
            case '+': return number1+number2;
            case '-': return number1-number2;
            case '*': return number1*number2;
            case '/': return number1/number2;
        }
        return -1;
    }

    private boolean isDigit(char c) {
        return c >= '0' && c <= '9';
    }
}
```

王争的算法训练营

习题课：栈

1047. 删除字符串中的所有相邻重复项（简单）

给出由小写字母组成的字符串 `S`，**重复项删除操作**会选择两个相邻且相同的字母，并删除它们。

在 S 上反复执行重复项删除操作，直到无法继续删除。

在完成所有重复项删除操作后返回最终的字符串。答案保证唯一。

示例：

输入： "abbaca"
输出： "ca"
解释：
例如，在 "abbaca" 中，我们可以删除 "bb" 由于两字母相邻且相同，这是此时唯一可以执行删除操作的重复项。之后我们得到字符串 "aaca"，其中又只有 "aa" 可以执行重复项删除操作，所以最后的字符串为 "ca"。

提示：

1. `1 <= S.length <= 20000`
2. `S` 仅由小写英文字母组成。

习题课：栈

1047. 删除字符串中的所有相邻重复项（简单）

```
class Solution {
    public String removeDuplicates(String S) {
        Deque<Character> deque = new LinkedList<>();
        for (int i = 0; i < S.length(); ++i) {
            char c = S.charAt(i);
            if (deque.isEmpty() || deque.peekLast() != c) {
                deque.addLast(c);
            } else {
                deque.pollLast();
            }
        }

        StringBuilder sb = new StringBuilder();
        while (!deque.isEmpty()) {
            sb.append(deque.pollFirst());
        }
        return sb.toString();
    }
}
```

习题课：栈

剑指 Offer 31. 栈的压入、弹出序列（中等）

输入两个整数序列，第一个序列表示栈的压入顺序，请判断第二个序列是否为该栈的弹出顺序。假设压入栈的所有数字均不相等。例如，序列 {1,2,3,4,5} 是某栈的压栈序列，序列 {4,5,3,2,1} 是该压栈序列对应的一个弹出序列，但 {4,3,5,1,2} 就不可能是该压栈序列的弹出序列。

示例 1：

```
输入：pushed = [1,2,3,4,5], popped = [4,5,3,2,1]
输出：true
```

示例 2：

```
输入：pushed = [1,2,3,4,5], popped = [4,3,5,1,2]
输出：false
解释：1 不能在 2 之前弹出。
```

pushed=1、2、3、4、5、6，popped=3、2、5、6、4、1

```java
class Solution {
    public boolean validateStackSequences(int[] pushed, int[] popped) {
        Stack<Integer> stack = new Stack<>();
        int i = 0;
        int j = 0;
        int k = 0;
        while (k < pushed.length + popped.length) {
            k++;
            // 出栈
            if (!stack.isEmpty() && j < popped.length && stack.peek() == popped[j]) {
                stack.pop();
                j++;
                continue;
            }
            // 入栈
            if (i < pushed.length) {
                stack.push(pushed[i]);
                i++;
                continue;
            }
            return false;
        }
        return true;
    }
}
```

王争的算法训练营

习题课：栈

王争的算法训练营

739. 每日温度（中等） 单调栈 （已讲）

请根据每日 `气温` 列表 `temperatures` ，请计算在每一天需要等几天才会有更高的温度。如果气温在这之后都不会升高，请在该位置用 `0` 来代替。

示例 1:

```
输入: temperatures = [73,74,75,71,69,72,76,73]
输出: [1,1,4,2,1,1,0,0]
```

示例 2:

```
输入: temperatures = [30,40,50,60]
输出: [1,1,1,0]
```

示例 3:

```
输入: temperatures = [30,60,90]
输出: [1,1,0]
```

提示：

- `1 <= temperatures.length <= 10^5`
- `30 <= temperatures[i] <= 100`

习题课：栈

739. 每日温度（中等） 单调栈 （已讲）

解法一：暴力解法

```
class Solution {
    public int[] dailyTemperatures(int[] T) {
        int n = T.length;
        int result[] = new int[n];
        for (int i = 0; i < n; ++i) {
            for (int j = i+1; j < n; ++j) {
                if (T[j] > T[i]) {
                    result[i] = j-i;
                    break;
                }
            }
        }
        return result;
    }
}
```

739. 每日温度（中等） 单调栈 （已讲）

解法二：单调栈

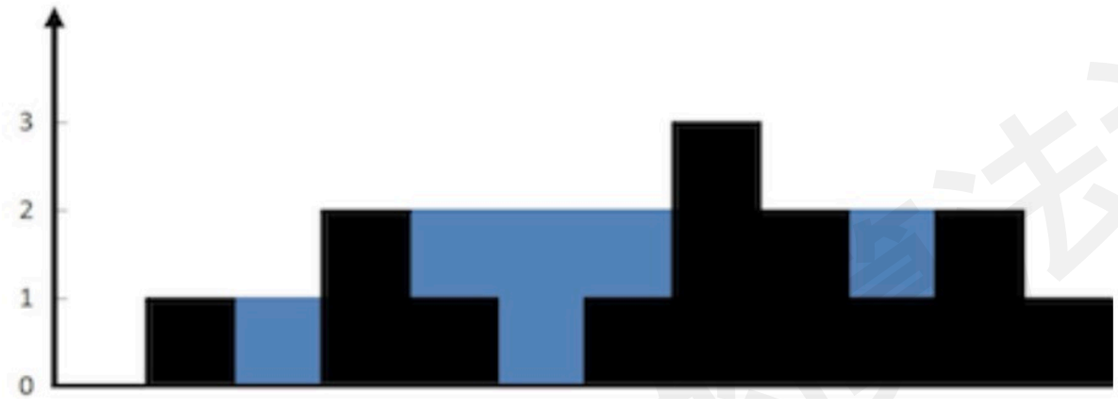
```
class Solution {
    public int[] dailyTemperatures(int[] T) {
        int n = T.length;
        int result[] = new int[n];
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < n; ++i) {
            while (!stack.empty() && T[stack.peek()] < T[i]) {
                int idx = stack.peek();
                result[idx] = i - idx;
                stack.pop();
            }
            stack.push(i);
        }
        return result;
    }
}
```

习题课：栈

42. 接雨水（困难）

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 1：




```
输入：height = [0,1,0,2,1,0,1,3,2,1,2,1]
输出：6
解释：上面是由数组 [0,1,0,2,1,0,1,3,2,1,2,1] 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）。
```

示例 2：

```
输入：height = [4,2,0,3,2,5]
输出：9
```

多种解法：

1）暴力解法

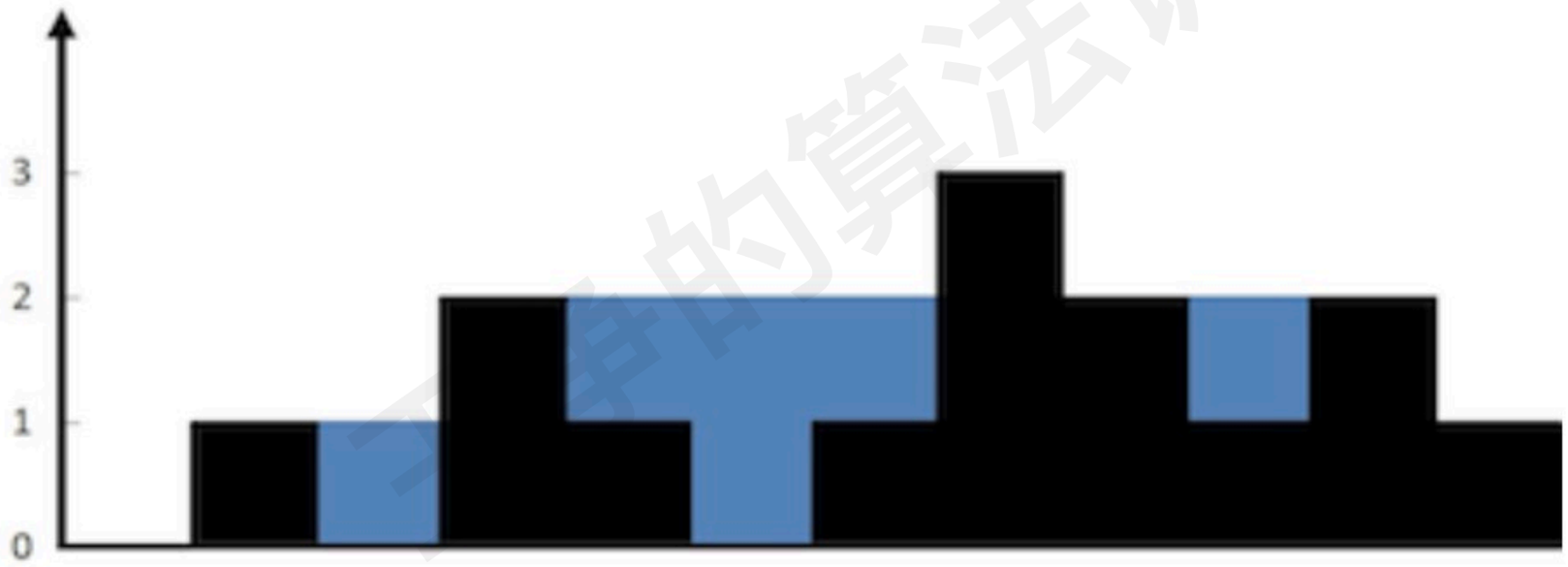
2）前缀/后缀统计解法

3）单调栈解法

4）双指针解法

1）暴力解法

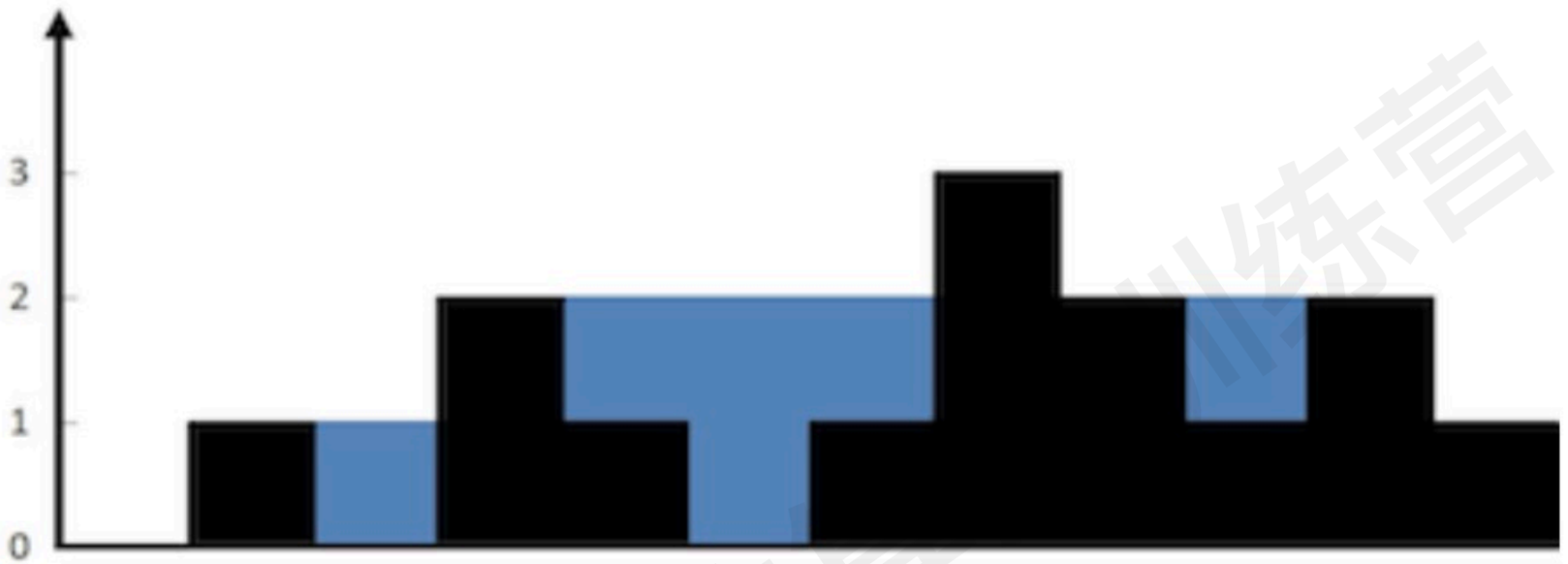
核心思想：
每个柱子之上承载的水量 = min(左侧最高柱子lh，右侧最高柱子rh) - 这个柱子的高度h
总的接水量=每个柱子之上承载水量的总和



时间复杂度是O(n^2)
空间复杂度是O(1)

```
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        int result = 0;
        // 遍历每个柱子h，查找它左边的最高柱子lh，和右边的最高柱子rh
        // 柱子上能承载的雨水=min(lh, rh)-h
        for (int i = 1; i < n-1; ++i) {
            int lh = 0;
            for (int j = 0; j < i; ++j) { // 左侧最高lh
                if (height[j] > lh) lh = height[j];
            }
            int rh = 0;
            for (int j = i+1; j < n; ++j) { // 右侧最高rh
                if (height[j] > rh) rh = height[j];
            }
            int carry = Math.min(lh, rh) - height[i];
            if (carry < 0) carry = 0;
            result += carry;
        }
        return result;
    }
}
```

2）前缀/后缀统计解法



王争的算法训练营

```java
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        // 前缀max
        int[] leftMax = new int[n];
        int max = 0;
        for (int i = 0; i < n; ++i) {
            leftMax[i] = Math.max(max, height[i]);
            max = leftMax[i];
        }
        // 后缀max
        int[] rightMax = new int[n];
        max = 0;
        for (int i = n-1; i >= 0; --i) {
            rightMax[i] = Math.max(max, height[i]);
            max = rightMax[i];
        }
        // 每个柱子之上承载的雨水
        int result = 0;
        for (int i = 1; i < n-1; i++) {
            result += Math.min(leftMax[i], rightMax[i])-height[i];
        }
        return result;
    }
}
```

时间复杂度是O(n)
空间复杂度是O(n)

王争的算法训练营

3）单调栈解法

左侧（垂直）分了多少层，一层一层的计算承载水量

时间复杂度是O(n)

空间复杂度是O(n)

```java
class Solution {
    public int trap(int[] height) {
        int n = height.length;
        int result = 0;
        Stack<Integer> stack = new Stack<>();
        for (int i = 0; i < n; ++i) {
            if (stack.isEmpty()) {
                stack.push(i); //存下标哈
                continue;
            }
            while (!stack.isEmpty()) {
                int top = stack.peek();
                if (height[top] >= height[i]) { //单调入栈
                    stack.push(i);
                    break;
                } else { // 找到凹槽了
                    top = stack.pop();
                    if (stack.isEmpty()) {
                       stack.push(i);
                       break;
                    }
                    int left = stack.peek();
                    int h = Math.min(height[left], height[i])-height[top];
                    int w = i-left-1;
                    result += h*w;
                }
            }
        }
        return result;
    }
}
```

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料