【华为OD机考 统一考试机试C卷】 转盘寿司（C++ Java JavaScript Python C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述

寿司店周年庆，正在举办优惠活动回馈新老客户。

寿司转盘上总共有 n 盘寿司，prices[i] 是第 i 盘寿司的价格，

如果客户选择了第 i 盘寿司，寿司店免费赠送客户距离第 i 盘寿司最近的下一盘寿司 j，前提是 prices[j] < prices[i]，如果没有满足条件的 j，则不赠送寿司。

每个价格的寿司都可无限供应。

输入描述

输入的每一个数字代表每盘寿司的价格，每盘寿司的价格之间使用空格分隔，例如:

```
3 15 6 14
```

表示:

- 第 0 盘寿司价格 prices[0] 为 3

- 第 1 盘寿司价格 prices[1] 为 15
- 第 2 盘寿司价格 prices[2] 为 6
- 第 3 盘寿司价格 prices[3] 为 14
- 寿司的盘数 n 范围为：1 ≤ n ≤ 500

每盘寿司的价格 price 范围为：1 ≤ price ≤ 1000

输出描述

输出享受优惠后的一组数据，每个值表示客户选择第 i 盘寿司时实际得到的寿司的总价格。使用空格进行分隔，例如：

```
3 21 9 17
```

用例

输入	3 15 6 14
输出	3 21 9 17
说明	无

用例解析

根据题目的描述，客户选择了第 i 盘寿司，寿司店免费赠送距离第 i 盘寿司最近的下一盘寿司 j，且 prices[j] < prices[i]。如果没有满足条件的 j，则不赠送寿司。因此，对于每一盘寿司，我们需要找到其价格右侧第一个比它小的寿司的价格，并将其加到当前寿司的价格上。

给定输入 `3 15 6 14`，我们来逐个分析：

1. 第 0 盘寿司价格为 3，没有比它更小的价格，所以输出为 3。
2. 第 1 盘寿司价格为 15，它右侧最近的更小价格是 6（第 2 盘寿司），所以输出为 15 + 6 = 21。
3. 第 2 盘寿司价格为 6，它右侧最近的更小价格是 3（第 0 盘寿司），所以 输出为 9。
4. 第 3 盘寿司价格为 14，它右侧最近的更小价格是 3（第 0 盘寿司），所以 输出为 17。

综合以上，输出结果为 `3 21 9 17`。

通过这个用例，可以得出数组是可以循环到头部继续寻找

解题思路

优惠规则是每个寿司盘可以用它右边第一个比它便宜的寿司盘的价格来享受优惠。如果右边没有更便宜的寿司盘，则保持原价。寿司盘是循环的，即最后一个寿司盘的右边是第一个寿司盘。

解题思路如下：

1. 数据处理：将读取的字符串按空格分割，转换成整数数组 `prices` ，这个数组包含了所有寿司盘的价格。同时，获取数组的长度 `n` ，代表寿司盘的总数。
2. 初始化数据结构：创建一个与价格数组等长的结果数组 `res` 来存储每个寿司盘享受优惠后的价格，并初始化一个双端队列 `stack` 作为栈，用来跟踪价格的索引。
3. 遍历价格数组：由于寿司盘是循环的，需要遍历 `2 * n - 1` 次来确保每个寿司盘都能找到它右边的第一个更便宜的寿司盘。使用模运算来处理循环数组的索引。
4. 处理栈：在遍历过程中，对于每个价格，检查栈顶元素（即之前遍历过的价格）是否大于当前价格。如果是，则说明找到了一个更便宜的寿司盘，计算栈顶元素对应寿司盘的优惠价格并更新到结果数组 `res` 中，然后将该元素出栈。重复此过程直到栈为空或栈顶元素小于当前价格。第一轮遍历（即 `j < n` 时）将每个索引加入栈中。
5. 处理剩余元素：遍历结束后，栈中可能还有元素，这些元素代表它们右侧没有更小的价格，因此直接将它们的价格作为结果。

整个解题思路利用了栈这一数据结构来有效地跟踪和更新每个寿司盘的优惠价格，通过一次遍历来优化算法的时间复杂度。

C++

```cpp
#include <iostream>
#include <vector>
#include <stack>
#include <sstream>

int main() {
    // 使用iostream从控制台读取输入
    std::string line;
    std::getline(std::cin, line);
    std::istringstream iss(line);

    // 读取一行输入，按空格分割，然后将每个数字字符串转换为整数，并收集到vector中

```

```cpp
    std::vector<int> prices;
    int price;
    while (iss >> price) {
        prices.push_back(price);
    }

    // 获取寿司价格数组的长度，代表寿司盘数
    int n = prices.size();

    // 创建一个数组来存储结果，即每个寿司盘享受优惠后的总价格
    std::vector<int> res(n, 0);
    // 创建一个栈来跟踪寿司价格的索引
    std::stack<int> stack;

    // 遍历每个寿司盘的价格，由于寿司盘是循环的，需要遍历两倍长度减一次
    for (int j = 0; j < n * 2 - 1; ++j) {
        // 计算当前索引，由于数组是循环的，使用模运算得到实际索引
        int index = j % n;
        // 当栈不为空且栈顶元素的价格大于当前索引对应的价格时
        while (!stack.empty() && prices[stack.top()] > prices[index]) {
            // 弹出栈顶元素的索引
            int topIndex = stack.top();
            stack.pop();
            // 计算栈顶元素享受优惠后的价格，并更新结果数组
            res[topIndex] = prices[topIndex] + prices[index];
        }
        // 第一轮遍历时，将索引压入栈中
        if (j < n) {
            stack.push(index);
        }
    }

    // 遍历完成后，栈中剩余的元素代表它们右侧没有更小的价格
    // 直接将它们自身的价格作为结果
    while (!stack.empty()) {
        int topIndex = stack.top();
        stack.pop();
        res[topIndex] = prices[topIndex];
    }


```

```
    // 输出结果，每个价格后加上空格
    for (int num : res) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        // 使用Scanner从控制台读取输入
        Scanner sc = new Scanner(System.in);
        // 读取一行输入，按空格分割，然后将每个数字字符串转换为整数，并收集到数组中
        int[] prices = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        // 获取寿司价格数组的长度，代表寿司盘数
        int n = prices.length;

        // 创建一个数组来存储结果，即每个寿司盘享受优惠后的总价格
        int[] res = new int[n];
        // 创建一个双端队列作为栈使用，用于跟踪寿司价格的索引
        Deque<Integer> stack = new ArrayDeque<>();

        // 遍历每个寿司盘的价格，由于寿司盘是循环的，需要遍历两倍长度减一次
        for (int j = 0; j < n * 2 - 1; j++) {
            // 计算当前索引，由于数组是循环的，使用模运算得到实际索引
            int index = j % n;
            // 当栈不为空且栈顶元素的价格大于当前索引对应的价格时
            while (!stack.isEmpty() && prices[stack.peek()] > prices[index]) {
                // 弹出栈顶元素的索引
                int topIndex = stack.pop();
                // 计算栈顶元素享受优惠后的价格，并更新结果数组
                res[topIndex] = prices[topIndex] + prices[index];
            }
            // 第一轮遍历时，将索引压入栈中
            if (j < n) {
```

```
                stack.push(index);
            }
        }

        // 遍历完成后，栈中剩余的元素代表它们右侧没有更小的价格
        // 直接将它们自身的价格作为结果
        while (!stack.isEmpty()) {
            int topIndex = stack.pop();
            res[topIndex] = prices[topIndex];
        }

        // 使用StringBuilder构建输出结果
        StringBuilder sb = new StringBuilder();
        for (int num : res) {
            // 将每个价格添加到StringBuilder中，并加上空格
            sb.append(num).append(" ");
        }
        // 输出结果，并去除末尾的空格
        System.out.println(sb.toString().trim());
    }
}
```

javaScript

```
const readline = require('readline');

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

// 从控制台读取一行输入
rl.on('line', (line) => {
  // 将输入的字符串按空格分割并转换成整数数组
  const prices = line.split(' ').map(Number);

  // 获取寿司价格数组的长度，代表寿司盘数
  const n = prices.length;

```

```
  // 创建一个数组来存储结果，即每个寿司盘享受优惠后的总价格
  const res = new Array(n).fill(0);
  // 创建一个栈来跟踪寿司价格的索引
  const stack = [];

  // 遍历每个寿司盘的价格，由于寿司盘是循环的，需要遍历两倍长度减一次
  for (let j = 0; j < n * 2 - 1; j++) {
    // 计算当前索引，由于数组是循环的，使用模运算得到实际索引
    const index = j % n;
    // 当栈不为空且栈顶元素的价格大于当前索引对应的价格时
    while (stack.length > 0 && prices[stack[stack.length - 1]] > prices[index]) {
      // 弹出栈顶元素的索引
      const topIndex = stack.pop();
      // 计算栈顶元素享受优惠后的价格，并更新结果数组
      res[topIndex] = prices[topIndex] + prices[index];
    }
    // 第一轮遍历时，将索引压入栈中
    if (j < n) {
      stack.push(index);
    }
  }

  // 遍历完成后，栈中剩余的元素代表它们右侧没有更小的价格
  // 直接将它们自身的价格作为结果
  while (stack.length > 0) {
    const topIndex = stack.pop();
    res[topIndex] = prices[topIndex];
  }

  // 输出结果，每个价格后加上空格
  console.log(res.join(' '));

  // 关闭readline接口
  rl.close();
});
```

Python

```

# 读取一行输入，按空格分割，然后将每个数字字符串转换为整数，并收集到列表中
```

```python
prices = list(map(int, input().split()))
# 获取寿司价格列表的长度，代表寿司盘数
n = len(prices)

# 创建一个列表来存储结果，即每个寿司盘享受优惠后的总价格
res = [0] * n
# 创建一个列表作为栈使用，用于跟踪寿司价格的索引
stack = []

# 遍历每个寿司盘的价格，由于寿司盘是循环的，需要遍历两倍长度减一次
for j in range(n * 2 - 1):
    # 计算当前索引，由于列表是循环的，使用模运算得到实际索引
    index = j % n
    # 当栈不为空且栈顶元素的价格大于当前索引对应的价格时
    while stack and prices[stack[-1]] > prices[index]:
        # 弹出栈顶元素的索引
        top_index = stack.pop()
        # 计算栈顶元素享受优惠后的价格，并更新结果列表
        res[top_index] = prices[top_index] + prices[index]
    # 第一轮遍历时，将索引压入栈中
    if j < n:
        stack.append(index)

# 遍历完成后，栈中剩余的元素代表它们右侧没有更小的价格
# 直接将它们自身的价格作为结果
while stack:
    top_index = stack.pop()
    res[top_index] = prices[top_index]

# 输出结果，每个价格后加上空格
print(' '.join(map(str, res)))

```

C语言

```c
#include <stdio.h>

#define MAX_N 500

```

```c
int main() {
    int prices[MAX_N]; // 创建一个数组，用于存储每盘寿司的价格
    int n = 0; // 初始化寿司盘数为0

    // 从控制台读取每盘寿司的价格，直到读取到换行符
    while (scanf("%d", &prices[n]) == 1) {
        n++; // 每读取一个价格，寿司盘数加1
    }

    int res[MAX_N]; // 创建一个数组，用于存储每个寿司盘享受优惠后的总价格
    int stack[MAX_N]; // 创建一个数组作为栈使用，用于跟踪寿司价格的索引
    int top = -1; // 初始化栈顶指针为-1，表示栈为空

    // 遍历每个寿司盘的价格，由于寿司盘是循环的，需要遍历两倍长度减一次
    for (int j = 0; j < n * 2 - 1; j++) {
        // 计算当前索引，由于数组是循环的，使用模运算得到实际索引
        int index = j % n;
        // 当栈不为空且栈顶元素的价格大于当前索引对应的价格时
        while (top >= 0 && prices[stack[top]] > prices[index]) {
            // 弹出栈顶元素的索引
            int topIndex = stack[top--];
            // 计算栈顶元素享受优惠后的价格，并更新结果数组
            res[topIndex] = prices[topIndex] + prices[index];
        }
        // 第一轮遍历时，将索引压入栈中
        if (j < n) {
            stack[++top] = index;
        }
    }

    // 遍历完成后，栈中剩余的元素代表它们右侧没有更小的价格
    // 直接将它们自身的价格作为结果
    while (top >= 0) {
        int topIndex = stack[top--];
        res[topIndex] = prices[topIndex];
    }

    // 输出每个寿司盘享受优惠后的总价格
    for (int i = 0; i < n; i++) {
        printf("%d ", res[i]);
```

```
    }

    return 0;
}
```

完整用例

用例1

3 15 6 14

用例2

1 2 3 4 5

用例3

5 4 3 2 1

用例4

2 2 2 2 2

用例5

10 20 30 40 50 40 30 20 10

用例6

1

用例7

1000 999 998 997 996

用例8

500 500 500 500 500

用例9

1 1000 1 1000 1

用例10

1000 1 1000 1 1000

文章目录


机考真题
华为OD
No No No
CSDN @算法大师