【华为OD机考 统一考试机试C卷】会议室占用时间段（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

现有若干个会议，所有会议共享一个会议室，用数组表示各个会议的开始时间和结束时间，格式为：

```
[[会议1开始时间，会议1结束时间]，[会议2开始时间，会议2结束时间]]
```

请计算会议室占用时间段。

输入描述

[[会议1开始时间, 会议1结束时间], [会议2开始时间, 会议2结束时间]]

备注：

- 会议室个数范围：[1, 100]
- 会议室时间段：[1, 24]

输出描述

输出格式预输入一致,具体请看用例。

```
[[会议开始时间，会议结束时间],[会议开始时间，会议结束时间]]
```

用例1

输入：

```
[[1,4],[2,5],[7,9],[14,18]]
```

输出：

```
[[1,5],[7,9],[14,18]]
```

说明：

> 时/间段[1,4]和[2,5]重叠，合并为[1,5]

用例2

输入：

```
[[1,4],[4,5]]
```

输出：

```
[[1,5]]
```

说明：

> 时间段[1,4]和[4,5]连续

解题思路

本题为lettocodo模式，不需要处理输入,只需要实现函数节课。

格式为：

```
int[][] merge(int[][] roomTimes) {}
```

1. 使用排序算法，根据每个会议时间段的开始时间（即每个内部数组的第一个元素）对 `roomTimes` 数组进行排序。
2. 创建一个数组，用于存储合并后的会议时间段。
3. 将排序后的第一个会议时间段作为当前会议时间段，并将其添加到合并列表中。
4. 从第二个会议时间段开始，遍历 `roomTimes` 数组中的每个会议时间段。
5. 对于每个遍历到的会议时间段，比较其开始时间与当前会议时间段的结束时间：
   - 如果当前会议时间段的结束时间大于等于遍历到的会议时间段的开始时间，则说明两个会议时间段有重叠。此时，需要更新当前会议时间段的结束时间为两个时间段中较晚的结束时间，以此来合并会议时间段。
   - 如果当前会议时间段的结束时间小于遍历到的会议时间段的开始时间，则说明两个会议时间段没有重叠。此时，将遍历到的会议时间段设置为新的当前会议时间段，并将其添加到合并列表中。
6. 继续步骤6的过程，直到遍历完所有会议时间段。

C++

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

// 定义用于存储会议时间段的结构体
struct Meeting {
    int start;
    int end;
};

// 比较函数，用于按会议开始时间排序
bool compareMeetings(const Meeting &a, const Meeting &b) {
    return a.start < b.start;
}

// 合并会议时间段的函数
```

```cpp
std::vector<Meeting> merge(std::vector<Meeting> &meetings) {
    // 如果会议列表为空或只有一个会议，则不需要合并，直接返回
    if (meetings.size() <= 1) {
        return meetings;
    }

    // 使用标准库中的sort函数，根据会议开始时间对会议进行排序
    std::sort(meetings.begin(), meetings.end(), compareMeetings);

    // 创建一个新的向量存储合并后的会议时间段
    std::vector<Meeting> merged;
    // 将第一个会议添加到合并后的列表中
    merged.push_back(meetings[0]);

    // 遍历所有会议
    for (const auto &nextMeeting : meetings) {
        // 引用合并列表中的最后一个会议
        Meeting &lastMerged = merged.back();

        // 如果当前会议的开始时间小于等于最后一个合并会议的结束时间，则合并
        if (nextMeeting.start <= lastMerged.end) {
            // 更新最后一个合并会议的结束时间为当前会议和最后一个合并会议中较晚的结束时间
            lastMerged.end = std::max(lastMerged.end, nextMeeting.end);
        } else {
            // 如果没有重叠，则将当前会议添加到合并列表中
            merged.push_back(nextMeeting);
        }
    }

    // 返回合并后的会议列表
    return merged;
}

int main() {
    std::string input;
    std::getline(std::cin, input); // 读取一行输入

    // 移除输入字符串中的所有方括号
    input.erase(std::remove(input.begin(), input.end(), '['), input.end());
    input.erase(std::remove(input.begin(), input.end(), ']'), input.end());

```

```
    // 使用字符串流分割输入字符串，获取会议时间
    std::istringstream iss(input);
    std::vector<Meeting> meetings;
    std::string start, end;
    while (std::getline(iss, start, ',') && std::getline(iss, end, ',')) {
        meetings.push_back({std::stoi(start), std::stoi(end)});
    }

    // 合并会议时间段
    std::vector<Meeting> mergedMeetings = merge(meetings);

    // 输出合并后的会议时间段
    std::cout << "[";
    for (size_t i = 0; i < mergedMeetings.size(); ++i) {
        std::cout << "[" << mergedMeetings[i].start << "," << mergedMeetings[i].end << "]";
        if (i < mergedMeetings.size() - 1) {
            std::cout << ",";
        }
    }
    std::cout << "]" << std::endl;

    return 0;
}
```

Java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        input = input.replaceAll("\\[", "").replaceAll("\\]", "");
        String[] inputArray = input.split(",");
        int[][] meetings = new int[inputArray.length / 2][2];
```

```
        for (int i = 0; i < inputArray.length; i += 2) {
            int start = Integer.parseInt(inputArray[i]);
            int end = Integer.parseInt(inputArray[i + 1]);
            meetings[i / 2] = new int[]{start, end};
        }

        int[][] mergedMeetings = merge(meetings);
        System.out.print("[");
        for (int i = 0; i < mergedMeetings.length; i++) {
            int[] time = mergedMeetings[i];
            System.out.print("[" + time[0] + "," + time[1] + "]");
            if (i < mergedMeetings.length - 1) {
                System.out.print(",");
            }
        }
        System.out.println("]");
        scanner.close();
    }

    public static int[][] merge(int[][] roomTimes) {

        // 使用Arrays.sort方法和Comparator来按照会议的开始时间对会议进行排序
        Arrays.sort(roomTimes, Comparator.comparingInt(a -> a[0]));

        // 创建一个ArrayList来存储合并后的会议时间段
        List<int[]> merged = new ArrayList<>();
        // 初始化当前会议时间段为排序后的第一个会议时间段
        int[] currentMeeting = roomTimes[0];
        // 将当前会议时间段添加到合并后的列表中
        merged.add(currentMeeting);

        // 遍历剩余的会议时间段
        for (int i = 1; i < roomTimes.length; i++) {
            // 获取下一个会议时间段
            int[] nextMeeting = roomTimes[i];

            // 如果当前会议的结束时间大于等于下一个会议的开始时间，说明两个会议时间段有重叠
            if (currentMeeting[1] >= nextMeeting[0]) {
                // 将当前会议的结束时间更新为两个会议结束时间的较大者，实现合并
```

```
                currentMeeting[1] = Math.max(currentMeeting[1], nextMeeting[1]);
            } else {
                // 如果没有重叠，则将下一个会议时间段设置为当前会议时间段，并添加到合并后的列表中
                currentMeeting = nextMeeting;
                merged.add(currentMeeting);
            }
        }

        // 将合并后的会议时间段列表转换为二维数组并返回
        return merged.toArray(new int[merged.size()][]);
    }
}
```

javaScript

```
// 引入readline模块，用于从命令行读取输入
const readline = require('readline');

// 创建readline接口实例
const rl = readline.createInterface({
  input: process.stdin, // 标准输入流
  output: process.stdout // 标准输出流
});

// 当从命令行接收到一行输入时触发回调函数
rl.on('line', (input) => {
  // 使用正则表达式移除输入字符串中的所有方括号
  input = input.replace(/\[/g, '').replace(/\]/g, '');
  // 根据逗号分割字符串，并将分割后的字符串数组转换为数字数组
  const inputArray = input.split(',').map(Number);
  // 初始化会议时间数组
  const meetings = [];

  // 遍历输入的数字数组，每两个数字组成一个会议时间，添加到会议时间数组中
  for (let i = 0; i < inputArray.length; i += 2) {
    const start = inputArray[i]; // 会议开始时间
    const end = inputArray[i + 1]; // 会议结束时间
    meetings.push([start, end]); // 将会议时间添加到数组中
  }
```

```
  // 调用merge函数合并会议时间
  const mergedMeetings = merge(meetings);
  // 输出合并后的会议时间数组，转换为JSON字符串格式
  console.log(JSON.stringify(mergedMeetings));
  // 关闭readline接口实例
  rl.close();
});

// 定义merge函数，用于合并会议时间
function merge(roomTimes) {

  // 对会议时间数组进行排序，按照会议的开始时间从小到大排序
  roomTimes.sort((a, b) => a[0] - b[0]);

  // 初始化合并后的会议时间数组，起始为排序后的第一个会议时间
  const merged = [roomTimes[0]];

  // 遍历剩余的会议时间
  for (let i = 1; i < roomTimes.length; i++) {
    // 获取合并数组中的最后一个会议时间
    const currentMeeting = merged[merged.length - 1];
    // 获取当前遍历到的会议时间
    const nextMeeting = roomTimes[i];

    // 如果当前会议的结束时间大于等于下一个会议的开始时间，则合并这两个会议
    if (currentMeeting[1] >= nextMeeting[0]) {
      // 更新当前会议的结束时间为两个会议结束时间的较大值
      currentMeeting[1] = Math.max(currentMeeting[1], nextMeeting[1]);
    } else {
      // 如果没有重叠，则将下一个会议添加到合并数组中
      merged.push(nextMeeting);
    }
  }

  // 返回合并后的会议时间数组
  return merged;
}
```

Python

```python
# 导入必要的库
import re

def merge_meetings(meetings):
    """
    合并会议时间段

    :param meetings: 二维列表，表示会议的开始和结束时间
    :return: 合并后的会议时间段列表
    """
    # 按照会议的开始时间对会议进行排序
    meetings.sort(key=lambda x: x[0])

    # 创建一个列表来存储合并后的会议时间段
    merged = []
    # 初始化当前会议时间段为排序后的第一个会议时间段
    current_meeting = meetings[0]
    # 将当前会议时间段添加到合并后的列表中
    merged.append(current_meeting)

    # 遍历剩余的会议时间段
    for next_meeting in meetings[1:]:
        # 如果当前会议的结束时间大于等于下一个会议的开始时间，说明两个会议时间段有重叠
        if current_meeting[1] >= next_meeting[0]:
            # 将当前会议的结束时间更新为两个会议结束时间的较大者，实现合并
            current_meeting[1] = max(current_meeting[1], next_meeting[1])
        else:
            # 如果没有重叠，则将下一个会议时间段设置为当前会议时间段，并添加到合并后的列表中
            current_meeting = next_meeting
            merged.append(current_meeting)

    # 返回合并后的会议时间段列表
    return merged

# 读取输入字符串
input_str = input()
# 使用正则表达式去除字符串中的中括号
input_str = re.sub(r"[\[\]]", "", input_str)
# 按照逗号分割字符串，得到会议时间的字符串列表
input_array = input_str.split(",")
```

```python
# 将字符串列表转换为会议时间的二维列表
meetings = [[int(input_array[i]), int(input_array[i + 1])] for i in range(0, len(input_array), 2)]

# 调用函数合并会议时间段
merged_meetings = merge_meetings(meetings)
# 打印合并后的会议时间段
print("[", end="")
for i, time in enumerate(merged_meetings):
    print(f"[{time[0]},{time[1]}]", end="")
    if i < len(merged_meetings) - 1:
        print(",", end="")
print("]")
```

C语言

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// 定义用于存储会议时间段的结构体
typedef struct {
    int start;
    int end;
} Meeting;

// 比较函数，用于按会议开始时间排序
int compareMeetings(const void *a, const void *b) {
    Meeting *meetingA = (Meeting *)a;
    Meeting *meetingB = (Meeting *)b;
    return meetingA->start - meetingB->start;
}

// 合并会议时间段的函数
Meeting *merge(Meeting *meetings, int size, int *returnSize) {
    if (size <= 1) {
        *returnSize = size;
        return meetings;
    }

    // 使用qsort函数，根据会议开始时间对会议进行排序
```

```
    qsort(meetings, size, sizeof(Meeting), compareMeetings);

    // 创建一个新的数组存储合并后的会议时间段
    Meeting *merged = (Meeting *)malloc(size * sizeof(Meeting));
    int mergedSize = 0;
    merged[mergedSize++] = meetings[0]; // 将第一个会议添加到合并后的数组中

    // 遍历所有会议
    for (int i = 1; i < size; ++i) {
        // 如果当前会议的开始时间小于等于最后一个合并会议的结束时间，则合并
        if (meetings[i].start <= merged[mergedSize - 1].end) {
            // 更新最后一个合并会议的结束时间为当前会议和最后一个合并会议中较晚的结束时间
            merged[mergedSize - 1].end = (meetings[i].end > merged[mergedSize - 1].end) ? meetings[i].end : merged[mergedSize - 1].end;
        } else {
            // 如果没有重叠，则将当前会议添加到合并数组中
            merged[mergedSize++] = meetings[i];
        }
    }

    *returnSize = mergedSize;
    return merged;
}

int main() {
    char input[1024];
    fgets(input, sizeof(input), stdin);

    // 移除输入字符串中的所有方括号
    char *p = input;
    while (*p) {
        if (*p == '[' || *p == ']') {
            *p = ' ';
        }
        p++;
    }

    // 分割输入字符串，获取会议时间
    Meeting meetings[512];
    int size = 0;
    char *token = strtok(input, ", ");
```

```
    while (token != NULL) {
        meetings[size].start = atoi(token);
        token = strtok(NULL, ", ");
        meetings[size].end = atoi(token);
        token = strtok(NULL, ", ");
        size++;
    }

    // 合并会议时间段
    int mergedSize;
    Meeting *mergedMeetings = merge(meetings, size, &mergedSize);

    // 输出合并后的会议时间段
    printf("[");
    for (int i = 0; i < mergedSize; ++i) {
        printf("[%d,%d]", mergedMeetings[i].start, mergedMeetings[i].end);
        if (i < mergedSize - 1) {
            printf(",");
        }
    }
    printf("]\n");

    free(mergedMeetings); // 释放动态分配的内存
    return 0;
}
```

文章目录

Python

C语言