【华为OD机考 统一考试机试C卷】测试用例执行计划（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

某个产品当前迭代周期内有 N 个特性（F1,F2,…FN）需要进行覆盖测试，每个特性都被评估了对应的优先级，特性使用其 ID 作为下标进行标识。

设计了 M 个测试用例（T1,T2,…,TM），每个测试用例对应一个覆盖特性的集合，测试用例使用其 ID 作为下标进行标识，测试用例的优先级定义为其覆盖的特性的优先级之和。

在开展测试之前，需要制定测试用例的执行顺序，规则为：优先级大的用例先执行，如果存在优先级相同的用例，用例 ID 小的先执行。

输入描述

第一行输入为 N 和 M，

- N 表示特性的数量，$0 < N \leq 100$
- M 表示测试用例的数量，$0 < M \leq 100$

之后 N 行表示特性 ID=1 到特性 ID=N 的优先级，

再接下来 M 行表示测试用例 ID=1 到测试用例 ID=M 关联的特性的 ID 的列表。

输出描述

按照执行顺序（优先级从大到小）输出测试用例的 ID，每行一个ID。

测试用例覆盖的 ID 不重复。

用例1

输入

```
5 4
1
1
2
3
5
1 2 3
1 4
3 4 5
2 3 4
```

输出

```
3
4
1
2
```

说明

> 测试用例的优先级计算如下：
> T1 = Pf1 + Pf2 + Pf3 = 1 + 1 + 2 = 4
> T2 = Pf1 + Pf4 = 1 + 3 = 4
> T3 = Pf3 + Pf4 + Pf5 = 2 + 3 + 5 = 10
> T4 = Pf2 + Pf3 + Pf4 = 1 + 2 + 3 = 6
> 按照优先级从小到大，以及相同优先级，ID小的先执行的规则，执行顺序为T3,T4,T1,T2

用例2

输入

```
3 3
3
1
5
1 2 3
1 2 3
1 2 3
```

输出

```
1
2
3
```

说明

> 测试用例的优先级计算如下：
> T1 = Pf1 + Pf2 + Pf3 = 3 + 1 + 5 = 9
> T2 = Pf1 + Pf2 + Pf3 = 3 + 1 + 5 = 9
> T3 = Pf1 + Pf2 + Pf3 = 3 + 1 + 5 = 9
> 每个优先级一样，按照 ID 从小到大执行，执行顺序为T1,T2,T3

解题思路

这道题看懂题目就会做了！！！

用例1包含了5个特性和4个测试用例，具体如下：

1. 特性优先级列表：
    - 特性1的优先级为1
    - 特性2的优先级为1

- 特性3的优先级为2
- 特性4的优先级为3
- 特性5的优先级为5

2. 测试用例及其涉及的特性：

- 测试用例1涉及的特性为1, 2, 3
- 测试用例2涉及的特性为1, 4
- 测试用例3涉及的特性为3, 4, 5
- 测试用例4涉及的特性为2, 3, 4

接下来解释每个测试用例的优先级计算和排序：

- 测试用例1的优先级计算：特性1 + 特性2 + 特性3 = 1 + 1 + 2 = 4
- 测试用例2的优先级计算：特性1 + 特性4 = 1 + 3 = 4
- 测试用例3的优先级计算：特性3 + 特性4 + 特性5 = 2 + 3 + 5 = 10
- 测试用例4的优先级计算：特性2 + 特性3 + 特性4 = 1 + 2 + 3 = 6

根据这些计算，测试用例按照优先级从高到低排列为：测试用例3, 测试用例4, 测试用例1, 测试用例2。如果有优先级相同的情况，则按照测试用例的ID升序排列。在这个例子中，测试用例1和测试用例2的优先级相同，因此按照ID顺序排列。最终的排序结果是：

1. 测试用例3
2. 测试用例4
3. 测试用例1
4. 测试用例2

C++

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
int main() {
```

```
    int N, M;
    cin >> N >> M;

    // 存储每个特性的优先级
    vector<int> feature_priorities(N);
    for (int i = 0; i < N; ++i) {
        cin >> feature_priorities[i];
    }

    // 处理每个测试用例，存储ID和优先级
    vector<pair<int, int>> test_cases;
    for (int i = 0; i < M; ++i) {
        int priority_sum = 0;
        for (int j = 0; j < N; ++j) {
            int feature;
            cin >> feature;
            priority_sum += feature_priorities[feature - 1];
        }
        test_cases.emplace_back(i + 1, priority_sum);
    }

    // 按优先级降序排序，相同优先级时按ID升序排序
    sort(test_cases.begin(), test_cases.end(), [](const pair<int, int>& a, const pair<int, int>& b) {
        if (a.second != b.second) return a.second > b.second;
        return a.first < b.first;
    });

    // 输出按优先级排序后的测试用例ID
    for (const auto& case_info : test_cases) {
        cout << case_info.first << endl;
    }

    return 0;
}
```

Java

```
import java.util.*;

public class TestPriority {
```

```
    // 内部类：表示测试用例，包含ID和优先级
    static class TestCase {
        int id; // 测试用例的ID
        int priority; // 测试用例的优先级

        TestCase(int id, int priority) {
            this.id = id; // 设置测试用例ID
            this.priority = priority; // 设置测试用例优先级
        }
    }
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 读取输入的N和M，分别代表特性数量和测试用例数量
        int[] tmp = Arrays.stream(scanner.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        int N = tmp[0]; // 特性数量
        int M = tmp[1]; // 测试用例数量

        // 存储每个特性的优先级
        int[] featurePriorities = new int[N];
        for (int i = 0; i < N; i++) {
            featurePriorities[i] = Integer.parseInt(scanner.nextLine()); // 读取并存储每个特性的优先级
        }

        // 处理每个测试用例
        TestCase[] testCases = new TestCase[M];
        for (int i = 0; i < M; i++) {
            // 读取并解析每个测试用例涉及的特性ID列表
            String[] features = scanner.nextLine().trim().split(" ");
            int prioritySum = 0; // 测试用例的总优先级
            for (String feature : features) {
                int featureId = Integer.parseInt(feature) - 1; // 调整下标，因为特性ID是从1开始的
                prioritySum += featurePriorities[featureId]; // 累加涉及特性的优先级到测试用例的总优先级
            }
            // 创建测试用例对象，并存储ID和计算得到的优先级
            testCases[i] = new TestCase(i + 1, prioritySum);
        }

        // 对测试用例根据优先级进行排序
        Arrays.sort(testCases, (a, b) -> {
```

```
            if (a.priority != b.priority) {
                return b.priority - a.priority; // 优先级不同时，按优先级降序排列
            }
            return a.id - b.id; // 优先级相同时，按ID升序排列
        });

        // 输出按优先级排序后的测试用例ID
        for (TestCase testCase : testCases) {
            System.out.println(testCase.id);
        }
    }


}
```

javaScript

```
const readline = require('readline');
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];

rl.on('line', (line) => {
    inputLines.push(line);
}).on('close', () => {
    let [N, M] = inputLines.shift().split(' ').map(Number);

    // 存储每个特性的优先级
    let featurePriorities = inputLines.slice(0, N).map(Number);

    // 处理每个测试用例，存储ID和优先级
    let testCases = inputLines.slice(N).map((line, index) => {
        let prioritySum = line.split(' ').reduce((sum, feature) => {
            return sum + featurePriorities[parseInt(feature) - 1];
        }, 0);
        return { id: index + 1, priority: prioritySum };
    });
```

```
    // 按优先级降序排序，相同优先级时按ID升序排序
    testCases.sort((a, b) => {
        return b.priority - a.priority || a.id - b.id;
    });

    // 输出按优先级排序后的测试用例ID
    testCases.forEach(testCase => {
        console.log(testCase.id);
    });
});
```

Python

```python
# 读取输入
N, M = map(int, input().split())

# 存储每个特性的优先级
feature_priorities = [int(input()) for _ in range(N)]

# 处理每个测试用例，存储ID和优先级
test_cases = []
for i in range(M):
    features = map(int, input().split())
    priority_sum = sum(feature_priorities[f - 1] for f in features)
    test_cases.append((i + 1, priority_sum))

# 按优先级降序排序，相同优先级时按ID升序排序
test_cases.sort(key=lambda x: (-x[1], x[0]))

# 输出按优先级排序后的测试用例ID
for case in test_cases:
    print(case[0])
```

C语言

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


```

```
// 测试用例结构体，包含用例ID和优先级
typedef struct {
    int id;
    int priority;
} TestCase;

// 比较函数，用于按优先级和ID排序
int compare(const void *a, const void *b) {
    TestCase *testCaseA = (TestCase *)a;
    TestCase *testCaseB = (TestCase *)b;
    // 如果优先级不同，则按优先级降序排序
    if (testCaseA->priority != testCaseB->priority) {
        return testCaseB->priority - testCaseA->priority;
    }
    // 如果优先级相同，则按ID升序排序
    return testCaseA->id - testCaseB->id;
}

int main() {
    int N, M;
    scanf("%d %d", &N, &M); // 读取特性数量和测试用例数量

    int feature_priorities[N]; // 存储每个特性的优先级
    for (int i = 0; i < N; ++i) {
        scanf("%d", &feature_priorities[i]); // 读取每个特性的优先级
    }

    TestCase test_cases[M]; // 存储测试用例
    for (int i = 0; i < M; ++i) {
        int priority_sum = 0; // 存储测试用例的优先级总和
        for (int j = 0; j < N; ++j) {
            int feature;
            scanf("%d", &feature); // 读取测试用例覆盖的特性ID
            priority_sum += feature_priorities[feature - 1]; // 计算优先级总和
        }
        test_cases[i].id = i + 1; // 设置测试用例ID
        test_cases[i].priority = priority_sum; // 设置测试用例优先级
    }

    // 对测试用例进行排序

```

```
    qsort(test_cases, M, sizeof(TestCase), compare);

    // 输出排序后的测试用例ID
    for (int i = 0; i < M; ++i) {
        printf("%d\n", test_cases[i].id);
    }

    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师