【华为OD机考 统一考试机试C卷】石头剪刀布游戏（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

石头剪刀布游戏有 3 种出拳形状：石头、剪刀、布。分别用字母 A , B , C 表示。

游戏规则:

出拳形状之间的胜负规则如下： A > B；B > C；C > A；">"左边一个字母，表示相对优势形状。右边一个字母，表示相对劣势形状。
当本场次中有且仅有一种出拳形状优于其它出拳形状，则该形状的玩家是胜利者。否则认为是平局。
当发生平局，没有赢家。有多个胜利者时，同为赢家。

- 例如 1： 三个玩家出拳分别是A, B, C ， 由于出现三方优势循环(即没有任何一方优于其它出拳者)，判断为平局。
- 例如 2： 两个玩家，出拳分别是 A, B ， 出拳 A 的获胜。
- 例如 3： 三个玩家，出拳全部是 A ，判为平局。

输入描述

在一场游戏中，每个玩家的信息为一行。玩家数量不超过 `1000` 。每个玩家信息有 `2` 个字段，用空格隔开：

1. 玩家 ID：一个仅由英文字母和数字组成的字符串
2. 出拳形状：以英文大写字母表示，`A` 、`B` 、`C` 形状。 例：

```
abc1 A
xyz B
```

输出描述

输出为赢家的玩家 ID 列表(一个或多个)，每个 ID 一行，按字符串升序排列。如果没有赢家，输出为 `"NULL"` 字符串。例如：

```
abc1
```

用例1

输入

```
abc1 A
xyz B
```

输出

```
abc1
```

说明

> A比B有优势，abc1 胜出

用例2

输入

```
abc1 A
xyz A
```

输出

```
NULL
```

说明

> 没有优胜的出拳形状，平局

用例3

输入

```
abc1 A
def A
alic A
xyz B
```

输出

```
abc1
alic
def
```

说明

> A为优胜方，有三个赢家

解题思路

1. 创建一个映射，用来存储每种出拳形状（A、B、C）对应的玩家ID列表。
2. 读取输入，将每个玩家的ID根据其出拳形状添加到映射中。

   根据游戏规则，判断出拳形状的种类：
   - 如果不是两种形状，即为0种（无输入）、1种（所有玩家出同一形状）或3种（每种形状至少一个玩家），则判定为平局。
   - 如果是两种形状，根据出拳规则（A > B, B > C, C > A）确定胜出的形状，其对应的玩家ID即为胜利者。
4. 如果有胜利者，对胜利者的ID进行排序并输出；如果没有胜利者，输出"NULL"。

为什么不需要考虑每种出拳形状的人数?

根据题目的游戏规则，只有当有且仅有一种出拳形状优于其他出拳形状时，才有赢家。这意味着只有两种出拳形状存在时，才能根据规则确定胜者。如果有三种形状或者只有一种形状，无论每种形状的人数是多少，都会判定为平局。因此，在判断胜负时，只需要关注出拳形状的种类，而不是每种形状的人数。

C++

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

using namespace std;
int main() {
    // 存储每种出拳形状对应的玩家ID列表
    map<string, vector<string>> shapeToPlayerIds;
    string playerId, shape;

    // 循环读取玩家ID和出拳形状
    while (cin >> playerId >> shape) {
        // 如果该出拳形状还未记录，则初始化玩家ID列表
        if (shapeToPlayerIds.find(shape) == shapeToPlayerIds.end()) {
            shapeToPlayerIds[shape] = vector<string>();
        }
        // 将玩家ID添加到对应出拳形状的列表中
        shapeToPlayerIds[shape].push_back(playerId);
    }

    // 如果每种出拳形状都只有一种，或者三种都有，则判定为平局
    if (shapeToPlayerIds.size() != 2) {
        cout << "NULL" << endl;
        return 0;
    }

    // 存储胜利玩家ID的列表
    vector<string> winningPlayerIds;

    // 根据出拳规则，确定胜利玩家ID列表
    if (shapeToPlayerIds.count("A") && shapeToPlayerIds.count("B")) {
```

```
        winningPlayerIds = shapeToPlayerIds["A"]; // A胜B
    } else if (shapeToPlayerIds.count("B") && shapeToPlayerIds.count("C")) {
        winningPlayerIds = shapeToPlayerIds["B"]; // B胜C
    } else if (shapeToPlayerIds.count("A") && shapeToPlayerIds.count("C")) {
        winningPlayerIds = shapeToPlayerIds["C"]; // C胜A
    } else {
        cout << "NULL" << endl;
        return 0;
    }

    // 对胜利玩家ID进行排序
    sort(winningPlayerIds.begin(), winningPlayerIds.end());

    // 输出胜利玩家ID
    for (const string& id : winningPlayerIds) {
        cout << id << endl;
    }

    return 0;
}
```

Java

```

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

public class RockPaperScissors {
    public static void main(String[] args) {
        // 存储每种出拳形状对应的玩家ID列表
        Map<String, List<String>> shapeToPlayerIds = new HashMap<>();
        // 用于读取输入数据
        Scanner scanner = new Scanner(System.in);
        // 循环读取玩家ID和出拳形状
        while (scanner.hasNext()) {
```

```
            String playerId = scanner.next(); // 玩家ID
            String shape = scanner.next();    // 出拳形状
            // 如果该出拳形状还未记录，则初始化玩家ID列表
            shapeToPlayerIds.putIfAbsent(shape, new ArrayList<>());
            // 将玩家ID添加到对应出拳形状的列表中
            shapeToPlayerIds.get(shape).add(playerId);
        }

        // 如果每种出拳形状都只有一种，或者三种都有，则判定为平局
        if (shapeToPlayerIds.size() != 2) {
            System.out.println("NULL");
            return;
        }

        // 存储胜利玩家ID的列表
        List<String> winningPlayerIds = new ArrayList<>();
        // 根据出拳规则，确定胜利玩家ID列表
        if (shapeToPlayerIds.containsKey("A") && shapeToPlayerIds.containsKey("B")) {
            winningPlayerIds = shapeToPlayerIds.get("A"); // A胜B
        } else if (shapeToPlayerIds.containsKey("B") && shapeToPlayerIds.containsKey("C")) {
            winningPlayerIds = shapeToPlayerIds.get("B"); // B胜C
        } else if (shapeToPlayerIds.containsKey("A") && shapeToPlayerIds.containsKey("C")) {
            winningPlayerIds = shapeToPlayerIds.get("C"); // C胜A
        } else { // 如果没有满足以上任何条件，则没有胜者
            System.out.println("NULL");
            return;
        }

        // 对胜利玩家ID进行排序
        Collections.sort(winningPlayerIds);
        // 输出胜利玩家ID
        for (String playerId : winningPlayerIds) {
            System.out.println(playerId);
        }
    }
}
```

javaScript

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

// 存储每种出拳形状对应的玩家ID列表
let shapeToPlayerIds = {};

rl.on('line', (line) => {
  const [playerId, shape] = line.split(' ');
  // 如果该出拳形状还未记录，则初始化玩家ID列表
  shapeToPlayerIds[shape] = shapeToPlayerIds[shape] || [];
  // 将玩家ID添加到对应出拳形状的列表中
  shapeToPlayerIds[shape].push(playerId);
}).on('close', () => {
  // 如果每种出拳形状都只有一种，或者三种都有，则判定为平局
  const shapes = Object.keys(shapeToPlayerIds);
  if (shapes.length !== 2) {
    console.log('NULL');
    process.exit(0);
  }

  // 存储胜利玩家ID的列表
  let winningPlayerIds;

  // 根据出拳规则，确定胜利玩家ID列表
  if (shapeToPlayerIds['A'] && shapeToPlayerIds['B']) {
    winningPlayerIds = shapeToPlayerIds['A']; // A胜B
  } else if (shapeToPlayerIds['B'] && shapeToPlayerIds['C']) {
    winningPlayerIds = shapeToPlayerIds['B']; // B胜C
  } else if (shapeToPlayerIds['A'] && shapeToPlayerIds['C']) {
    winningPlayerIds = shapeToPlayerIds['C']; // C胜A
  } else {
    console.log('NULL');
    process.exit(0);
  }

  // 对胜利玩家ID进行排序

```

```
  winningPlayerIds.sort();

  // 输出胜利玩家ID
  winningPlayerIds.forEach((id) => {
    console.log(id);
  });
});
```

Python

```python
# Python代码
import sys

# 存储每种出拳形状对应的玩家ID列表
shape_to_player_ids = {}

# 循环读取玩家ID和出拳形状
for line in sys.stdin:
    player_id, shape = line.strip().split()
    # 如果该出拳形状还未记录，则初始化玩家ID列表
    if shape not in shape_to_player_ids:
        shape_to_player_ids[shape] = []
    # 将玩家ID添加到对应出拳形状的列表中
    shape_to_player_ids[shape].append(player_id)

# 如果每种出拳形状都只有一种，或者三种都有，则判定为平局
if len(shape_to_player_ids) != 2:
    print("NULL")
else:
    # 存储胜利玩家ID的列表
    winning_player_ids = []

    # 根据出拳规则，确定胜利玩家ID列表
    if 'A' in shape_to_player_ids and 'B' in shape_to_player_ids:
        winning_player_ids = shape_to_player_ids['A']  # A胜B
    elif 'B' in shape_to_player_ids and 'C' in shape_to_player_ids:
        winning_player_ids = shape_to_player_ids['B']  # B胜C
    elif 'A' in shape_to_player_ids and 'C' in shape_to_player_ids:
        winning_player_ids = shape_to_player_ids['C']  # C胜A
    else:
```

```
        print("NULL")

    # 对胜利玩家ID进行排序
    winning_player_ids.sort()

    # 输出胜利玩家ID
    for player_id in winning_player_ids:
        print(player_id)
```

C语言

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

// 定义玩家结构体
typedef struct {
    char id[100]; // 玩家ID
    char shape;   // 出拳形状
} Player;

// 比较函数，用于qsort排序
int compare(const void *a, const void *b) {
    Player *playerA = (Player *)a;
    Player *playerB = (Player *)b;
    return strcmp(playerA->id, playerB->id);
}

int main() {
    Player players[1000]; // 存储所有玩家信息
    int countA = 0, countB = 0, countC = 0; // 记录每种出拳形状的玩家数量
    int n = 0; // 玩家总数
    char id[100], shape;

    // 循环读取玩家ID和出拳形状
    while (scanf("%s %c", id, &shape) != EOF) {
        strcpy(players[n].id, id);
        players[n].shape = shape;
        // 根据出拳形状增加计数
        if (shape == 'A') countA++;
```

```
        if (shape == 'B') countB++;
        if (shape == 'C') countC++;
        n++;
    }

    // 如果每种出拳形状都只有一种，或者三种都有，则判定为平局
    if (countA > 0 && countB > 0 && countC > 0 || countA == n || countB == n || countC == n) {
        printf("NULL\n");
    } else {
        // 根据出拳规则，确定胜利玩家ID列表
        char winShape = (countA > 0 && countC == 0) ? 'A' : (countB > 0 && countA == 0) ? 'B' : 'C';
        Player winners[1000]; // 存储胜利玩家信息
        int winCount = 0; // 胜利玩家数量
        for (int i = 0; i < n; i++) {
            if (players[i].shape == winShape) {
                winners[winCount++] = players[i];
            }
        }
        // 对胜利玩家ID进行排序
        qsort(winners, winCount, sizeof(Player), compare);
        // 输出胜利玩家ID
        for (int i = 0; i < winCount; i++) {
            printf("%s\n", winners[i].id);
        }
    }

    return 0;
}
```

文章目录

解题思路

C++

Java

javaScript

Python

C语言