【华为OD机考 统一考试机试C卷】内存冷热标记（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

现代计算机系统中通常存在多级的存储设备，针对海量 workload 的优化的一种思路是将热点内存页优先放到快速存储层级，这就需要对内存页进行冷热标记。

一种典型的方案是基于内存页的访问频次进行标记，如果统计窗口内访问次数大于等于设定阈值，则认为是热内存页，否则是冷内存页。

对于统计窗口内跟踪到的访存序列和阈值，现在需要实现基于频次的冷热标记。内存页使用页框号作为标识。

输入描述

第一行输入为 `N`，表示访存序列的记录条数，$0 < N \leq 10000$

第二行为访存序列，空格分隔的 `N` 个内存页框号

第三行为阈值

输出描述

第一行输出标记为热内存的内存页个数，如果没有被标记的热内存页，则输出 0 。

如果第一行 > 0，则接下来按照访问频次降序输出内存页框号，一行一个，频次一样的页框号，页框号小的排前面。

用例1

输入	**10** **1 2 1 2 1 2 1 2 1 2** **5**
输出	2 1 2
说明	在这个例子中，内存页框号 1 和 2 都被访问了 5 次，达到了阈值，因此它们被标记为热内存页。输出首先是热内存页的数量 2，然后是按照访问频次降序排列的页框号 1 和 2(频次一样的页框号，页框号小的排前面)。

用例2

输入	**5** **1 2 3 4 5** **3**
输出	0
说明	在这个例子中，没有任何内存页的访问次数达到阈值 3，因此没有热内存页，输出为 0。

C++

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

using namespace std;
int main() {
    // 用于存储输入行的字符串
    string line;
    // 用于存储页面访问序列
    vector<int> pageAccessSequence;
    // 页面访问次数
```

```
    int pageAccessCount;
    // 热门页面的阈值
    int hotThreshold;

    // 读取页面访问次数
    getline(cin, line);
    pageAccessCount = stoi(line);

    // 读取页面访问序列
    getline(cin, line);
    istringstream iss(line);
    int page;
    while (iss >> page) {
        pageAccessSequence.push_back(page);
    }

    // 读取热门页面的阈值
    getline(cin, line);
    hotThreshold = stoi(line);

    // 统计每个页面的访问频率
    unordered_map<int, int> pageFrequency;
    for (int page : pageAccessSequence) {
        pageFrequency[page]++;
    }

    // 过滤出热门页面
    vector<int> hotPages;
    for (const auto& kv : pageFrequency) {
        if (kv.second >= hotThreshold) {
            hotPages.push_back(kv.first);
        }
    }

    // 输出热门页面的数量
    cout << hotPages.size() << endl;

    // 如果存在热门页面
    if (!hotPages.empty()) {
        // 对热门页面进行排序

```

```
        sort(hotPages.begin(), hotPages.end(), [&pageFrequency](int a, int b) {
            if (pageFrequency[a] == pageFrequency[b]) return a < b;
            return pageFrequency[a] > pageFrequency[b];
        });

        // 输出排序后的热门页面
        for (int page : hotPages) {
            cout << page << endl;
        }
    }

    return 0;
}
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int N = Integer.parseInt(scanner.nextLine());
        String[] accesses = scanner.nextLine().split(" ");
        int threshold = Integer.parseInt(scanner.nextLine());
        scanner.close();

        // 使用 TreeMap 来存储内存页框号和对应的访问次数
        // TreeMap 默认按照 key 升序排列，这里我们需要按照访问次数降序，页框号升序排列
        Map<Integer, Integer> frequencyMap = new TreeMap<>();
        for (String access : accesses) {
            int pageFrame = Integer.parseInt(access);
            frequencyMap.put(pageFrame, frequencyMap.getOrDefault(pageFrame, 0) + 1);
        }

        // 使用 PriorityQueue 来对内存页框号进行排序
        PriorityQueue<Integer> hotPages = new PriorityQueue<>((a, b) -> {
            int freqCompare = frequencyMap.get(b).compareTo(frequencyMap.get(a));
            if (freqCompare == 0) {
                return a.compareTo(b);
            }
```

```
            return freqCompare;
        });

        // 将达到阈值的热内存页加入到优先队列中
        for (Map.Entry<Integer, Integer> entry : frequencyMap.entrySet()) {
            if (entry.getValue() >= threshold) {
                hotPages.offer(entry.getKey());
            }
        }

        // 输出结果
        int hotCount = hotPages.size();
        System.out.println(hotCount);
        while (!hotPages.isEmpty()) {
            System.out.println(hotPages.poll());
        }
    }
}
```

javaScript

```
// 引入 readline 模块用于读取命令行输入
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
});

// 用于存储输入行的数组
let inputLines = [];

// 监听 'line' 事件，每次输入后触发
rl.on('line', (line) => {
  // 将输入的每一行添加到 inputLines 数组
  inputLines.push(line);
  // 当输入行数达到 3 行时，关闭 readline 接口
  if (inputLines.length === 3) {
    rl.close();
```

```
  }
}).on('close', () => {
  // 解析输入的第一行为页面访问次数
  const pageAccessCount = parseInt(inputLines[0].trim(), 10);
  // 解析输入的第二行为页面访问序列，转换为数字数组
  const pageAccessSequence = inputLines[1].trim().split(' ').map(Number);
  // 解析输入的第三行为热门页面的阈值
  const hotThreshold = parseInt(inputLines[2].trim(), 10);

  // 使用 reduce 方法统计每个页面的访问频率
  const pageFrequency = pageAccessSequence.reduce((acc, page) => {
    acc[page] = (acc[page] || 0) + 1; // 如果页面已存在则增加计数，否则初始化为 1
    return acc;
  }, {});

  // 根据阈值过滤出热门页面，并转换为数字数组
  const hotPages = Object.entries(pageFrequency)
    .filter(([page, freq]) => freq >= hotThreshold)
    .map(([page]) => parseInt(page, 10));

  // 输出热门页面的数量
  console.log(hotPages.length);

  // 如果存在热门页面
  if (hotPages.length > 0) {
    // 对热门页面进行排序，先按访问频率降序，频率相同则按页面号升序
    hotPages.sort((a, b) => {
      return pageFrequency[b] - pageFrequency[a] || a - b;
    });

    // 输出排序后的热门页面
    hotPages.forEach((page) => {
      console.log(page);
    });
  }


});
```

Python

```python
# 获取输入
page_access_count = int(input().strip())
page_access_sequence = map(int, input().strip().split())
hot_threshold = int(input().strip())

# 统计内存页框号出现的次数
from collections import Counter
page_frequency = Counter(page_access_sequence)

# 确定热内存页
hot_pages = [page for page, freq in page_frequency.items() if freq >= hot_threshold]

# 输出热内存页数量
print(len(hot_pages))

# 如果存在热内存页，按照要求排序并输出
if hot_pages:
    hot_pages.sort(key=lambda page: (-page_frequency[page], page))
    for page in hot_pages:
        print(page)
```

C语言

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

#define MAX_NUM 10000

// 定义一个结构体，用于存储页面和对应的访问频率
typedef struct {
    int page;
    int frequency;
} PageFrequency;

// 定义一个比较函数，用于 qsort 函数
int cmp(const void *a, const void *b) {
    PageFrequency *pf1 = (PageFrequency *)a;
    PageFrequency *pf2 = (PageFrequency *)b;
    if (pf1->frequency == pf2->frequency)
        return pf1->page - pf2->page;
```

```
    return pf2->frequency - pf1->frequency;
}

int main() {
    // 页面访问次数
    int pageAccessCount;
    // 热门页面的阈值
    int hotThreshold;

    // 读取页面访问次数
    scanf("%d", &pageAccessCount);

    // 创建一个数组存储页面访问序列
    int pageAccessSequence[MAX_NUM];
    // 循环读取每个页面的访问序列
    for (int i = 0; i < pageAccessCount; ++i) {
        scanf("%d", &pageAccessSequence[i]);
    }

    // 读取热门页面的阈值
    scanf("%d", &hotThreshold);

    // 创建一个结构体数组，用于存储每个页面的访问频率
    PageFrequency pageFrequency[MAX_NUM] = {0};
    for (int i = 0; i < pageAccessCount; ++i) {
        pageFrequency[pageAccessSequence[i]].page = pageAccessSequence[i];
        pageFrequency[pageAccessSequence[i]].frequency++;
    }

    // 过滤出热门页面
    PageFrequency hotPages[MAX_NUM];
    int hotPagesCount = 0;
    for (int i = 0; i < MAX_NUM; ++i) {
        if (pageFrequency[i].frequency >= hotThreshold) {
            hotPages[hotPagesCount++] = pageFrequency[i];
        }
    }

    // 输出热门页面的数量
    printf("%d\n", hotPagesCount);

```

```
59
60      // 如果存在热门页面
61      if (hotPagesCount > 0) {
62          // 对热门页面进行排序
63          qsort(hotPages, hotPagesCount, sizeof(PageFrequency), cmp);
64
65          // 输出排序后的热门页面
66          for (int i = 0; i < hotPagesCount; ++i) {
67              printf("%d\n", hotPages[i].page);
68          }
69      }
70
71      return 0;
    }
```

完整用例

用例1

```
1  10
2  1 1 2 2 3 3 4 4 5 5
3  2
```

用例2

```
1  10
2  1 1 1 1 1 1 1 1 1 2
3  3
```

用例3

```
1  10
2  1 2 3 4 5 6 7 8 9 10
3  11
```

用例4

```
10
1 2 2 3 3 3 4 4 4 4
3
```

用例5

```
10
1 1 1 1 2 2 2 3 3 4
3
```

用例6

```
10
100 200 300 100 200 100 400 500 100 200
3
```

用例7

```
10
1 2 3 4 5 6 7 8 9 10
1
```

用例8

```
10
1 2 3 4 5 6 7 8 9 10
2
```

用例9

```
10
65535 65535 65535 65535 65535 1 1 1 1 1
5
```

用例10

```
10
1 2 1 2 1 2 1 2 1 2
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师