【华为OD机考 统一考试机试C卷】API集群负载统计（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

某个产品的RESTful API集合部署在服务器集群的多个节点上，近期对客户端访问日志进行了采集，需要统计各个API的访问频次，根据热点信息在服务器节点之间做负载均衡，现在需要实现热点信息统计查询功能。

RESTful API是由多个层级构成，层级之间使用 / 连接，如 /A/B/C/D 这个地址，A属于第一级，B属于第二级，C属于第三级，D属于第四级。

现在负载均衡模块需要知道给定层级上某个名字出现的频次，未出现过用0表示，实现这个功能。

输入描述

第一行为N，表示访问历史日志的条数，$0 < N \leq 100$。

接下来N行，每一行为一个RESTful API的URL地址，约束地址中仅包含英文字母和连接符 / ，最大层级为10，每层级字符串最大长度为10。

最后一行为层级L和要查询的关键字

输出描述

输出给定层级上，关键字出现的频次，使用完全匹配方式（大小写敏感）。

用例

输入

```
5
/huawei/computing/no/one
/huawei/computing
/huawei
/huawei/cloud/no/one
/huawei/wireless/no/one
2 computing
```

输出

```
2
```

说明

在第二层级上，computing出现了2次，因此输出2

用例2

输入

```
5
/huawei/computing/no/one
/huawei/computing
/huawei
/huawei/cloud/no/one
/huawei/wireless/no/one
4 two
```

输出

```
0
```

说明

> 存在第四层级的URL上，没有出现two，因此频次是0

解题思路

1. 创建一个映射表，键为层级和关键字，值为频次。这个映射表将用于存储每个层级和关键字的出现频次。
2. 遍历每一条访问历史日志。对于每一条日志，将URL地址按照"/"分割成多个部分。
3. 检查每个层级的字符串。对于每个层级，将层级和关键字作为键。如果键在映射表中存在，就将频次加1，否则将频次设为1。

C++

```cpp
#include <iostream>
#include <sstream>
#include <map>
#include <vector>
#include <string>
using namespace std;
int main() {
    // 创建一个map，键为层级和关键字，值为频次
    map<string, int> map;

    // 创建一个vector来存储输入的数据
    vector<string> lines;

    // 读取输入的数据
    string line;
    while (getline(cin, line)) {
        if (line.empty()) {
            break;
        }
        lines.push_back(line);
    }

    // 读取访问历史日志的条数
    int N = stoi(lines[0]);

    // 遍历每一条访问历史日志
    for (int i = 1; i <= N; i++) {
        // 将URL地址按照"/"分割成多个部分
```

```
        stringstream ss(lines[i]);
        string part;
        vector<string> parts;
        while (getline(ss, part, '/')) {
            parts.push_back(part);
        }

        // 检查每个层级的字符串
        for (int j = 1; j < parts.size(); j++) {
            // 将层级和关键字作为键
            string key = to_string(j) + '-' + parts[j];
            // 如果键在map中存在，就将频次加1，否则将频次设为1
            map[key]++;
        }
    }

    // 读取要查询的层级和关键字
    stringstream ss(lines[N + 1]);
    string L, keyword;
    ss >> L >> keyword;

    // 输出给定层级上，关键字出现的频次
    string key = L + '-' + keyword;
    cout << (map.count(key) ? map[key] : 0) << endl;

    return 0;
}
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // 读取访问历史日志的条数
        int N = scanner.nextInt();
        scanner.nextLine();

        // 创建一个HashMap，键为层级和关键字，值为频次
```

```
        Map<String, Integer> map = new HashMap<>();

        // 遍历每一条访问历史日志
        for (int i = 0; i < N; i++) {
            // 将URL地址按照"/"分割成多个部分
            String[] parts = scanner.nextLine().split("/");
            // 检查每个层级的字符串
            for (int j = 1; j < parts.length; j++) {
                // 将层级和关键字作为键
                String key = j + "-" + parts[j];
                // 如果键在HashMap中存在，就将频次加1，否则将频次设为1
                map.put(key, map.getOrDefault(key, 0) + 1);
            }
        }

        // 读取要查询的层级和关键字
        int L = scanner.nextInt();
        String keyword = scanner.next();

        // 输出给定层级上，关键字出现的频次
        System.out.println(map.getOrDefault(L + "-" + keyword, 0));
    }
}
```

javaScript

```
// 创建一个对象，键为层级和关键字，值为频次
let map = {};

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

// 创建一个数组来存储输入的数据
let lines = [];

// 当接收到用户输入的数据时，将数据存入数组
```

```javascript
rl.on('line', function(line){
    lines.push(line.trim());
});

// 当接收完所有数据后，开始处理
rl.on('close', function(){
    // 读取访问历史日志的条数
    let N = parseInt(lines[0]);

    // 遍历每一条访问历史日志
    for (let i = 1; i <= N; i++) {
        // 将URL地址按照"/"分割成多个部分
        let parts = lines[i].split('/');
        // 检查每个层级的字符串
        for (let j = 1; j < parts.length; j++) {
            // 将层级和关键字作为键
            let key = j.toString() + '-' + parts[j];
            // 如果键在对象中存在，就将频次加1，否则将频次设为1
            map[key] = (map[key] || 0) + 1;
        }
    }

    // 读取要查询的层级和关键字
    let [L, keyword] = lines[N + 1].split(' ');

    // 输出给定层级上，关键字出现的频次
    console.log(map[L + '-' + keyword] || 0);
});
```

Python

```python
# 创建一个字典，键为层级和关键字，值为频次
map = {}

# 读取访问历史日志的条数
N = int(input())

# 遍历每一条访问历史日志
for _ in range(N):
    # 将URL地址按照"/"分割成多个部分
```

```
    parts = input().split('/')
    # 检查每个层级的字符串
    for j in range(1, len(parts)):
        # 将层级和关键字作为键
        key = str(j) + '-' + parts[j]
        # 如果键在字典中存在，就将频次加1，否则将频次设为1
        map[key] = map.get(key, 0) + 1

# 读取要查询的层级和关键字
L, keyword = input().split()

# 输出给定层级上，关键字出现的频次
print(map.get(L + '-' + keyword, 0))
```

C语言

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

#define MAX_LOGS 100
#define MAX_LEVELS 10
#define MAX_LENGTH 11 // 每层级字符串最大长度为10，再加上一个结束符

// 定义一个结构体来存储层级和关键字的组合以及对应的频次
typedef struct {
    char key[MAX_LEVELS * MAX_LENGTH]; // 层级和关键字的组合
    int count; // 频次
} ApiFrequency;

int main() {
    int N, L;
    char keyword[MAX_LENGTH];
    ApiFrequency frequencies[MAX_LOGS * MAX_LEVELS] = {0}; // 存储所有可能的层级和关键字组合
    int frequencyCount = 0;

    scanf("%d", &N); // 读取访问历史日志的条数
    getchar(); // 消除换行符

    // 读取每一条访问历史日志
```

```
    for (int i = 0; i < N; i++) {
        char url[MAX_LEVELS * MAX_LENGTH];
        fgets(url, sizeof(url), stdin); // 读取一行URL地址
        url[strcspn(url, "\n")] = 0; // 去除换行符

        // 分割URL地址
        char *part = strtok(url, "/");
        int level = 1;
        while (part != NULL) {
            // 构造层级和关键字的组合
            char key[MAX_LEVELS * MAX_LENGTH];
            sprintf(key, "%d-%s", level, part);

            // 检查是否已经存在于frequencies中
            int found = 0;
            for (int j = 0; j < frequencyCount; j++) {
                if (strcmp(frequencies[j].key, key) == 0) {
                    frequencies[j].count++; // 如果存在，频次加1
                    found = 1;
                    break;
                }
            }
            if (!found) {
                // 如果不存在，添加新的组合到frequencies
                strcpy(frequencies[frequencyCount].key, key);
                frequencies[frequencyCount].count = 1;
                frequencyCount++;
            }

            part = strtok(NULL, "/");
            level++;
        }
    }

    // 读取要查询的层级和关键字
    scanf("%d %s", &L, keyword);

    // 构造查询的key
    char queryKey[MAX_LEVELS * MAX_LENGTH];
    sprintf(queryKey, "%d-%s", L, keyword);
```

```
    // 查询频次
    int frequency = 0;
    for (int i = 0; i < frequencyCount; i++) {
        if (strcmp(frequencies[i].key, queryKey) == 0) {
            frequency = frequencies[i].count;
            break;
        }
    }

    // 输出给定层级上，关键字出现的频次
    printf("%d\n", frequency);

    return 0;
}
```

完整用例

用例1

```
3
/x/y/z
/x/y
/x
1 y
```

用例2

```
1
/a/b/c
3 c
```

用例3

```
4
/apple
/apple/orange
/apple/banana

```

```
6 /apple/orange/kiwi
  2 orange
```

用例4

```
5
/a
/a/b
/a/b/c
/a/b/c/d
/a/b/c/d/e
4 d
```

用例5

```
2
/x/y/z
/x/y/z/a
3 a
```

用例6

```
3
/foo/bar
/foo/bar/baz
/foo
1 bar
```

用例7

```
4
/x
/x/y
/y/x
/x/y/x
3 x
```

用例8

```
6
/a/b/c/d
/a/b/c
/a/b
/a
/b/a
/c/b/a
2 b
```

用例9

```
7
/p/q/r
/p/q/r/s
/p/q/r/s/t
/p/q/r/s/t/u
/v/w/x/y/z
/a/b/c/d/e
/f/g/h/i/j
5 e
```

用例10

```
8
/one/two/three/four/five/six/seven/eight
/one/two/three/four/five/six/seven
/one/two/three/four/five/six
/one/two/three/four/five
/one/two/three/four
/one/two/three
/one/two
/one
6 six
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师