

od0613

[华为OD 机考 - 简易内存池 \(2025 B卷\)-CSDN博客](#)

[华为OD 机试 - 中文分词模拟器 \(2025 B卷 200分\)-CSDN博客](#)

[华为OD机试 - 模拟工作队列 \(2025 B卷 200分\)_2025 华为od机考 队列-CSDN博客](#)

[华为od 机试 - 最小矩阵宽度 \(2025 B卷 200分\)-CSDN博客](#)

[华为OD机试 - 二叉树的广度优先遍历 \(2025 B卷 200分\)_华为od机试 二叉树的广度优先遍历-CSDN博客](#)

[华为OD 机试 - 池化资源共享 \(2025 B卷 100分\)-CSDN博客](#)

[华为OD 2025 B卷 - 选修课 \(机试 100分\)-CSDN博客](#)

[2025 华为OD机试 - 考勤信息 \(2025 B卷 100分\)-CSDN博客](#)

[华为OD 机考 - 找等值元素 / 找数字 \(2025 B卷\) 100分_华为od机考题目-CSDN博客](#)

[华为OD 机考 - 字符串计数匹配 \(机试 2025 B卷 100分\)_华为od 计数器算法题-CSDN博客](#)

华为OD机考 - 简易内存池(2025 B卷)-CSDN博客

简易内存池

华为OD机试真题目录: [点击查看](#)

2025B卷 200分题型

题目描述

请实现一个简易内存池,根据请求命令完成[内存分配和释放](#)。

- 内存池支持两种操作命令, REQUEST和RELEASE, 其格式为:
- REQUEST=请求的内存大小 表示请求分配指定大小内存, 如果分配成功, 返回分配到的内存首地址; 如果内存不足, 或指定的大小为0, 则输出error。
- RELEASE=释放的内存首地址 表示释放掉之前分配的内存, 释放成功无需输出, 如果释放不存在的首地址则输出error。

注意:

1. 内存池总大小为100字节。
2. 内存池地址分配必须是连续内存, 并优先从低地址分配。
3. 内存释放后可被再次分配, 已释放的内存空闲时不能被二次释放。
4. 不会释放已申请的内存块的中间地址。
5. 释放操作只是针对首地址所对应的单个内存块进行操作, 不会影响其它内存块。

输入描述

首行为整数 N , 表示操作命令的个数, 取值范围: $0 < N \leq 100$ 。

接下来的N行, 每行将给出一个操作命令, 操作命令和参数之间用“=”分割。

输出描述

请求分配指定大小内存时, 如果分配成功, 返回分配到的内存首地址; 如果内存不足, 或指定的大小为0, 则输出error

释放掉之前分配的内存时, 释放成功无需输出, 如果释放不存在的首地址则输出error。

用例1

输入

```
1 2  
2 REQUEST=10  
3 REQUEST=20
```

Plain Text |

输出

```
1 0  
2 10
```

Plain Text |

说明

用例2

输入

```
1 5  
2 REQUEST=10  
3 REQUEST=20  
4 RELEASE=0  
5 REQUEST=20  
6 REQUEST=10
```

Plain Text |

输出

```
1 0  
2 10  
3 30  
4 0
```

Plain Text |

说明

第一条指令，申请地址0~9的10个字节内存，返回首地址0

第二条指令，申请地址10~29的20字节内存，返回首地址10

第三条指令，释放首地址为0的内存申请，0~9地址内存被释放，变为空闲，释放成功，无需输出

第四条指令，申请20字节内存，09地址内存连续空间不足20字节，往后查找到30-49地址，返回首地址30

第五条指令，申请10字节，0~9地址内存空间足够，返回首地址0

题解

思路： 模拟

- 定义used数组(按照起始位置升序排序) 存储被占用内存空间区间,存储格式为:[起始位置, 结束位置]
- 当request申请size大小内存时, 默认start =0开始, [start, start+size-1],判断是否和已分配的区间存在交叉区域, 如果存在交叉区域。start更新为used[i] 1+1.直到找到不交叉区间。如果一直找不到这样的区域则打印error。 注意边界问题
- release时, 遍历used, 寻找是否存在这样的区域。存在则移除这个区间, 不存在输出error。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 using namespace std;
8
9 // 通用 split 函数
10 vector<string> split(const string& str, const string& delimiter) {
11     vector<string> result;
12     size_t start = 0;
13     size_t end = str.find(delimiter);
14     while (end != string::npos) {
15         result.push_back(str.substr(start, end - start));
16         start = end + delimiter.length();
17         end = str.find(delimiter, start);
18     }
19     // 添加最后一个部分
20     result.push_back(str.substr(start));
21     return result;
22 }
23
24 bool isIntersection(int start1, int end1, int start2, int end2) {
25     if (start1 == start2) {
26         return true;
27     } else if (start1 < start2) {
28         return end1 >= start2;
29     } else {
30         return end2 >= start1;
31     }
32 }
33 }
34
35 void handle(vector<vector<string>> cmdss) {
36     vector<pair<int,int>> used;
37     //处理边界
38     used.push_back({100,101});
39
40     for (int i = 0; i < cmdss.size();i++) {
41         string op = cmdss[i][0];
42         int value = stoi(cmdss[i][1]);
43         if (op == "REQUEST") {
44             if (value == 0) {
```

```
45             cout << "error" << endl;
46             continue;
47         }
48         int start = 0;
49         bool flag = true;
50         for (int j = 0; j < used.size(); j++) {
51             int end = start + value - 1;
52             // 超过可申请的最大内存空间
53             if (end >= 100) {
54                 break;
55             }
56             if (!isIntersection(start, end, used[j].first, used[j].se
cond)) {
57
58                 used.insert(used.begin() + j, {start, end});
59                 flag = false;
60                 cout << start << endl;
61
62                 break;
63             }else {
64                 start = used[j].second+1;
65             }
66         }
67
68         if (flag) {
69             cout << "error" << endl;
70         }
71     } else {
72         if (value >= 100) {
73             cout << "error" << endl;
74         }
75         bool flag = true;
76         for (int j = 0; j < used.size(); j++) {
77             if (used[j].first == value) {
78                 used.erase(used.begin() + j);
79                 flag = false;
80                 break;
81             }
82         }
83     }
84     if (flag) {
85         cout << "error" << endl;
86     }
87 }
88 }
89 }
90 }
```

```
91
92 int main() {
93     int n ;
94     cin >> n;
95     vector<vector<string>> cmd;
96     while (n--){
97         string s;
98         cin >> s;
99         cmd.push_back(split(s, "="));
100    }
101    handle(cmd);
102    return 0;
103 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 通用 split 函数
5     private static String[] split(String str, String delimiter) {
6         return str.split(delimiter);
7     }
8
9     // 判断两个区间是否有交集
10    private static boolean isIntersection(int start1, int end1, int start
11    2, int end2) {
12        if (start1 == start2) {
13            return true;
14        } else if (start1 < start2) {
15            return end1 >= start2;
16        } else {
17            return end2 >= start1;
18        }
19    }
20
21    private static void handle(List<String[]> cmds) {
22        List<int[]> used = new ArrayList<>();
23        // 处理边界
24        used.add(new int[]{100, 101});
25
26        for (String[] cmd : cmds) {
27            String op = cmd[0];
28            int value = Integer.parseInt(cmd[1]);
29
30            if (op.equals("REQUEST")) {
31                int start = 0;
32                boolean flag = true;
33                if (value == 0) {
34                    System.out.println("error");
35                    continue;
36                }
37
38                for (int j = 0; j < used.size(); j++) {
39                    int end = start + value - 1;
40                    // 超过可申请的最大内存空间
41                    if (end >= 100) {
42                        break;
43                    }
44                }
45            }
46        }
47    }
48}
```

```

44                     if (!isIntersection(start, end, used.get(j)[0], used.g
45     et(j)[1])) {
46                     used.add(j, new int[]{start, end});
47                     flag = false;
48                     System.out.println(start);
49                     break;
50                 } else {
51                     start = used.get(j)[1] + 1;
52                 }
53             }
54             if (flag) {
55                 System.out.println("error");
56             }
57         } else {
58             if (value >= 100) {
59                 System.out.println("error");
60             }
61             boolean flag = true;
62             for (int j = 0; j < used.size(); j++) {
63                 if (used.get(j)[0] == value) {
64                     used.remove(j);
65                     flag = false;
66                     break;
67                 }
68             }
69             if (flag) {
70                 System.out.println("error");
71             }
72         }
73     }
74 }
75
76 public static void main(String[] args) {
77     Scanner scanner = new Scanner(System.in);
78     int n = Integer.parseInt(scanner.nextLine());
79     List<String[]> cmd = new ArrayList<>();
80
81     while (n-- > 0) {
82         String s = scanner.nextLine();
83         cmd.add(split(s, "="));
84     }
85
86     handle(cmd);
87     scanner.close();
88 }
```

Python

```
1 import sys
2
3
4
5 # 判断两个区间是否有交集
6 def is_intersection(start1, end1, start2, end2):
7     if start1 == start2:
8         return True
9     elif start1 < start2:
10        return end1 >= start2
11    else:
12        return end2 >= start1
13
14 def handle(cmds):
15     used = [[100, 101]] # 处理边界
16
17     for op, value in cmds:
18         value = int(value)
19
20         if op == "REQUEST":
21             if value == 0:
22                 print("error")
23                 continue
24             start = 0
25             flag = True
26
27             for j in range(len(used)):
28                 end = start + value - 1
29                 # 超过可申请的最大内存空间
30                 if end >= 100:
31                     break
32                 if not is_intersection(start, end, used[j][0], used[j]
33 [1]):
34                     used.insert(j, [start, end])
35                     flag = False
36                     print(start)
37                     break
38                 else:
39                     start = used[j][1] + 1
40
41             if flag:
42                 print("error")
43
44         else: # RELEASE 操作
```

```
44         if value >= 100:
45             print("error")
46             continue
47         flag = True
48         for j in range(len(used)):
49             if used[j][0] == value:
50                 used.pop(j)
51                 flag = False
52                 break
53         if flag:
54             print("error")
55
56 def main():
57     n = int(sys.stdin.readline().strip())
58     cmd = [sys.stdin.readline().strip().split("=") for _ in range(n)]
59     handle(cmd)
60
61 if __name__ == "__main__":
62     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9 rl.on('line', (line) => {
10     inputLines.push(line);
11 }).on('close', () => {
12     let n = parseInt(inputLines[0].trim());
13     let cmds = inputLines.slice(1, n + 1).map(line => line.split("="));
14
15     function isIntersection(start1, end1, start2, end2) {
16         if (start1 === start2) return true;
17         return start1 < start2 ? end1 >= start2 : end2 >= start1;
18     }
19
20     function handle(cmds) {
21         let used = [[100, 101]]; // 处理边界
22
23         for (let i = 0; i < cmds.length; i++) {
24             let [op, value] = cmds[i];
25             value = parseInt(value);
26
27             if (op === "REQUEST") {
28                 if (value == 0) {
29                     console.log("error");
30                     continue
31                 }
32                 let start = 0;
33                 let flag = true;
34
35                 for (let j = 0; j < used.length; j++) {
36                     let end = start + value - 1;
37                     if (end >= 100) break;
38                     if (!isIntersection(start, end, used[j][0], used[j]
39 [1])) {
40                         used.splice(j, 0, [start, end]);
41                         flag = false;
42                         console.log(start);
43                         break;
44                     } else {
```

```
44                     start = used[j][1] + 1;
45                 }
46             }
47             if (flag) console.log("error");
48         } else {
49             if (value >= 100) {
50                 console.log("error");
51                 continue;
52             }
53             let flag = true;
54             for (let j = 0; j < used.length; j++) {
55                 if (used[j][0] === value) {
56                     used.splice(j, 1);
57                     flag = false;
58                     break;
59                 }
60             }
61             if (flag) console.log("error");
62         }
63     }
64 }
65
66 handle(cmds);
67 }) .
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 判断两个区间是否相交
12 func isIntersection(start1, end1, start2, end2 int) bool {
13     if start1 == start2 {
14         return true
15     } else if start1 < start2 {
16         return end1 >= start2
17     } else {
18         return end2 >= start1
19     }
20 }
21
22 // 处理 REQUEST 和 RELEASE 命令
23 func handle(cmds [][]string) {
24     used := []struct {
25         start, end int
26     }{{100, 101}} // 预设一个边界, 防止越界
27
28     for _, cmd := range cmds {
29         op := cmd[0]
30         value, _ := strconv.Atoi(cmd[1])
31
32         if op == "REQUEST" {
33             start := 0
34             flag := true
35             if value == 0 {
36                 fmt.Println("error")
37                 continue
38             }
39
40             for i := 0; i < len(used); i++ {
41                 end := start + value - 1
42                 // 超过可申请的最大内存空间
43                 if end >= 100 {
44                     break
```

```
45     }
46     if !isIntersection(start, end, used[i].start, used[i].end) {
47         // 插入新的内存块
48         used = append(used[:i], append([]struct{ start, end int }{{star
t, end}}), used[i:]...))
49         flag = false
50         fmt.Println(start)
51         break
52     } else {
53         start = used[i].end + 1
54     }
55 }
56 if flag {
57     fmt.Println("error")
58 }
59 } else { // RELEASE
60     if value >= 100 {
61         fmt.Println("error")
62         continue
63     }
64     flag := true
65     for i := 0; i < len(used); i++ {
66         if used[i].start == value {
67             // 删除该内存块
68             used = append(used[:i], used[i+1:]...)
69             flag = false
70             break
71         }
72     }
73     if flag {
74         fmt.Println("error")
75     }
76 }
77 }
78 }
79
80 func main() {
81     scanner := bufio.NewScanner(os.Stdin)
82
83     // 读取命令个数
84     scanner.Scan()
85     n, _ := strconv.Atoi(scanner.Text())
86
87     cmd := make([][]string, 0, n)
88     for i := 0; i < n; i++ {
89         scanner.Scan()
90         cmd = append(cmd, strings.Split(scanner.Text(), "="))
```

```
91     }
92
93     handle(cmds)
```

| 来自: 华为OD 机考 – 简易内存池 (2025 B卷)-CSDN博客

华为OD机试 - 中文分词模拟器 (2025 B卷 200分)-CSDN博客

中文分词模拟器

华为OD机试真题目录: [点击查看](#)

2025 B卷 200分题型

题目描述

给定一个连续不包含空格的字符串，该字符串仅包含英文小写字母及英文标点符号（逗号、分号、句号），同时给定词库，对该字符串进行精确分词。

说明：

1. 精确分词：字符串分词后，不会出现重叠。即"ilovechina"，不同词库可分割为"i,love,china"，"ilove,china"，不能分割出现重叠的"i,ilove,china"，i 出现重叠
2. 标点符号不成词，仅用于断句
3. 词库：根据外部知识库统计出来的常用词汇例：dictionary = [“i”，“love”，“china”，“ilovechina”，“ilove”]
4. 分词原则：采用分词顺序优先且最长匹配原则
 - “ilovechina”，假设分词结果 [i,ilove,lo,love,ch,china,ilovechina]，则输出 [ilove,chna]
 - 错误输出：[i,ilovechina]，原因：“ilove” > 优先于 “ilovechina” 成词
 - 错误输出：[i,love,chna]，原因：“ilove” > "i"遵循最长匹配原则

输入描述

第一行输入待分词语句 “ilovechina”[字符串长度](#)限制：0 < length < 256

第二行输入中文词库 “i,love,china,ch,na,ve,lo,this,is,this,word” 词库长度限制：1 < length < 100000

输出描述

按顺序输出分词结果 “i,love,chna”

示例1

输入

```
Plain Text |  
1 ilovechina  
2 i,love,china,ch,na,ve,lo,this,is,the,word
```

输出

```
Plain Text |  
1 i,love,china
```

示例2

输入

```
Plain Text |  
1 iat  
2 i,love,china,ch,na,ve,lo,this,is,the,word,beauti,tiful,ful
```

输出

```
Plain Text |  
1 i,a,t
```

说明

| 单个字母，不在词库中且不成词则输出单个字母

示例3

输入

```
Plain Text |  
1 ilovechina,thewordisbeautiful  
2 i,love,china,ch,na,ve,lo,this,is,the,word,beauti,tiful,ful
```

输出

```
Plain Text |  
1 i,love,china the,word,is,beauti,ful
```

说明

题解

思路：本题解决需要用到 前缀树/字典树 的数据结构来处理词典。

1. 接收词库中的单词，使用 词库 单词预构建 字典树，用于处理字符串分词。 字典树 也是数据结构中一个比较常见的数据结构，推荐自己去学习学习。
2. 后续利用 前缀树 模拟分词，遍历待分词字符串，确定起点之后 贪心 尽可能在词典中向后寻找最长匹配的单词进行分词。具体匹配的逻辑可以参照下面的代码逻辑，本质上也是 字典树 数据结构的运用。匹配不上的情况则原样输出当前字符即可。
3. 遍历完待分词字符串之后按题目要求输出对应答案即可。

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 #include <algorithm>
6
7 using namespace std;
8
9 // Trie 结构定义
10 struct TrieNode {
11     bool isWord;
12     TrieNode* children[26];
13
14     TrieNode() : isWord(false) {
15         fill(begin(children), end(children), nullptr);
16     }
17 };
18
19 // 根节点
20 TrieNode* root = new TrieNode();
21
22 // 插入单词
23 void insertWord(const string& word) {
24     TrieNode* currentNode = root;
25     for (char c : word) {
26         int index = c - 'a';
27         if (!currentNode->children[index]) {
28             currentNode->children[index] = new TrieNode();
29         }
30         currentNode = currentNode->children[index];
31     }
32     currentNode->isWord = true;
33 }
34
35 int main() {
36     // 读取输入并转换为小写
37     string sentence, dictionary;
38     getline(cin, sentence);
39     transform(sentence.begin(), sentence.end(), sentence.begin(), ::tolower);
40     getline(cin, dictionary);
41
42     // 插入字典中的单词
43     stringstream ss(dictionary);
```

```

44     string word;
45     while (getline(ss, word, ',')) {
46         insertWord(word);
47     }
48
49     vector<string> result;
50     int startIndex = 0;
51     int sentenceSize = sentence.size();
52
53     while (startIndex < sentenceSize) {
54         // 如果当前字符不是字母，直接加入结果
55         if (!isalpha(sentence[startIndex])) {
56             result.push_back(string(1, sentence[startIndex++]));
57             continue;
58         }
59
60         int endIndex = sentenceSize;
61         TrieNode* currentNode = root;
62         int lastValidIndex = -1;
63
64         // 从 startIndex 开始，寻找最长匹配的单词
65         for (int i = startIndex; i < sentenceSize; i++) {
66             char c = sentence[i];
67             if (!isalpha(c) || !currentNode->children[c - 'a']) {
68                 break;
69             }
70             currentNode = currentNode->children[c - 'a'];
71             if (currentNode->isWord) {
72                 lastValidIndex = i;
73             }
74         }
75
76         // 如果找到了匹配的单词，则使用最长匹配的单词，否则取当前字符
77         if (lastValidIndex != -1) {
78             result.push_back(sentence.substr(startIndex, lastValidIndex -
79                               startIndex + 1));
80             startIndex = lastValidIndex + 1;
81         } else {
82             result.push_back(string(1, sentence[startIndex++]));
83         }
84
85         // 结果输出
86         cout << result[0];
87         for (size_t i = 1; i < result.size(); i++) {
88             cout << "," << result[i];
89         }

```

```
90     cout << endl;  
91  
92     return 0;  
93 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     static class TrieNode {
5         boolean isWord;
6         TrieNode[] children = new TrieNode[26];
7
8         TrieNode() {
9             Arrays.fill(children, null);
10        }
11    }
12
13    static TrieNode root = new TrieNode();
14
15    // 插入单词
16    static void insertWord(String word) {
17        TrieNode currentNode = root;
18        for (char c : word.toCharArray()) {
19            int index = c - 'a';
20            if (currentNode.children[index] == null) {
21                currentNode.children[index] = new TrieNode();
22            }
23            currentNode = currentNode.children[index];
24        }
25        currentNode.isWord = true;
26    }
27
28    public static void main(String[] args) {
29        Scanner scanner = new Scanner(System.in);
30
31        // 读取输入，并转换为小写
32        String sentence = scanner.nextLine().toLowerCase();
33        String dictionary = scanner.nextLine();
34        scanner.close();
35
36        // 插入字典中的单词
37        for (String word : dictionary.split(",")) {
38            insertWord(word);
39        }
40
41        List<String> result = new ArrayList<>();
42        int startIndex = 0, sentenceSize = sentence.length();
43
44        while (startIndex < sentenceSize) {
```

```
45 // 如果当前字符不是字母，直接加入结果
46 if (!Character.isLetter(sentence.charAt(startIndex))) {
47     result.add(String.valueOf(sentence.charAt(startIndex++)));
48     continue;
49 }
50
51 int lastValidIndex = -1;
52 TrieNode currentNode = root;
53
54 // 从 startIndex 开始，寻找最长匹配的单词
55 for (int i = startIndex; i < sentenceSize; i++) {
56     char c = sentence.charAt(i);
57     if (!Character.isLetter(c) || currentNode.children[c - 'a'] == null) {
58         break;
59     }
60     currentNode = currentNode.children[c - 'a'];
61     if (currentNode.isWord) {
62         lastValidIndex = i;
63     }
64 }
65
66 if (lastValidIndex != -1) {
67     result.add(sentence.substring(startIndex, lastValidIndex
+ 1));
68     startIndex = lastValidIndex + 1;
69 } else {
70     result.add(String.valueOf(sentence.charAt(startIndex++)));
71 }
72 }
73
74 // 结果输出
75 System.out.println(String.join(",", result));
76 }
77 }
```

Python

```
1 import sys
2
3 # Trie 结构定义
4 class TrieNode:
5     def __init__(self):
6         self.is_word = False
7         self.children = {}
8
9 # 根节点
10 root = TrieNode()
11
12 # 插入单词
13 def insert_word(word):
14     current_node = root
15     for c in word:
16         if c not in current_node.children:
17             current_node.children[c] = TrieNode()
18             current_node = current_node.children[c]
19         current_node.is_word = True
20
21 # 读取输入并转换为小写
22 sentence = sys.stdin.readline().strip().lower()
23 dictionary = sys.stdin.readline().strip()
24
25 # 插入字典中的单词
26 for word in dictionary.split(','):
27     insert_word(word)
28
29 result = []
30 start_index = 0
31 sentence_size = len(sentence)
32
33 while start_index < sentence_size:
34     # 如果当前字符不是字母，直接加入结果
35     if not sentence[start_index].isalpha():
36         result.append(sentence[start_index])
37         start_index += 1
38         continue
39
40     last_valid_index = -1
41     current_node = root
42
43     # 从 start_index 开始，寻找最长匹配的单词
44     for i in range(start_index, sentence_size):
```

```
45     c = sentence[i]
46     if c not in current_node.children:
47         break
48     current_node = current_node.children[c]
49     if current_node.is_word:
50         last_valid_index = i
51
52     if last_valid_index != -1:
53         result.append(sentence[start_index:last_valid_index + 1])
54         start_index = last_valid_index + 1
55     else:
56         result.append(sentence[start_index])
57         start_index += 1
58
59 # 结果输出
60 print("".join(result))
```

JavaScript

```
1 const readline = require('readline');
2
3 // Trie 结构定义
4 class TrieNode {
5     constructor() {
6         this.isWord = false;
7         this.children = {};
8     }
9 }
10
11 // 根节点
12 const root = new TrieNode();
13
14 // 插入单词
15 function insertWord(word) {
16     let currentNode = root;
17     for (const c of word) {
18         if (!(c in currentNode.children)) {
19             currentNode.children[c] = new TrieNode();
20         }
21         currentNode = currentNode.children[c];
22     }
23     currentNode.isWord = true;
24 }
25
26 // 读取输入
27 const rl = readline.createInterface({
28     input: process.stdin,
29     output: process.stdout
30 });
31
32 let inputLines = [];
33 rl.on('line', (line) => {
34     inputLines.push(line);
35     if (inputLines.length === 2) {
36         rl.close();
37     }
38 });
39
40 rl.on('close', () => {
41     let sentence = inputLines[0].toLowerCase();
42     let dictionary = inputLines[1];
43
44     // 插入字典中的单词
```

```

45     dictionary.split(',').forEach(insertWord);
46
47     let result = [];
48     let startIndex = 0;
49     let sentenceSize = sentence.length;
50
51     while (startIndex < sentenceSize) {
52         if (!/[a-z]/.test(sentence[startIndex])) {
53             result.push(sentence[startIndex]);
54             startIndex++;
55             continue;
56         }
57
58         let lastValidIndex = -1;
59         let currentNode = root;
60
61         for (let i = startIndex; i < sentenceSize; i++) {
62             let c = sentence[i];
63             if (!(c in currentNode.children)) break;
64             currentNode = currentNode.children[c];
65             if (currentNode.isWord) lastValidIndex = i;
66         }
67
68         if (lastValidIndex !== -1) {
69             result.push(sentence.slice(startIndex, lastValidIndex + 1));
70             startIndex = lastValidIndex + 1;
71         } else {
72             result.push(sentence[startIndex]);
73             startIndex++;
74         }
75     }
76
77     console.log(result.join(","));
78 }

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // Trie 结构定义
11 type TrieNode struct {
12     isWord    bool
13     children  map[rune]*TrieNode
14 }
15
16 // 初始化根节点
17 var root = &TrieNode{children: make(map[rune]*TrieNode)}
18
19 // 插入单词
20 func insertWord(word string) {
21     currentNode := root
22     for _, c := range word {
23         if _, exists := currentNode.children[c]; !exists {
24             currentNode.children[c] = &TrieNode{children: make(map[rune]*TrieNode)}
25         }
26         currentNode = currentNode.children[c]
27     }
28     currentNode.isWord = true
29 }
30
31 func main() {
32     reader := bufio.NewReader(os.Stdin)
33     sentence, _ := reader.ReadString('\n')
34     sentence = strings.ToLower(strings.TrimSpace(sentence))
35
36     dictionary, _ := reader.ReadString('\n')
37     dictionary = strings.TrimSpace(dictionary)
38
39     // 插入字典中的单词
40     for _, word := range strings.Split(dictionary, ",") {
41         insertWord(word)
42     }
43 }
```

```
44     var result []string
45     startIndex := 0
46     sentenceSize := len(sentence)
47
48     for startIndex < sentenceSize {
49         if sentence[startIndex] < 'a' || sentence[startIndex] > 'z' {
50             result = append(result, string(sentence[startIndex]))
51             startIndex++
52             continue
53         }
54
55         lastValidIndex := -1
56         currentNode := root
57
58         for i := startIndex; i < sentenceSize; i++ {
59             c := rune(sentence[i])
60             if _, exists := currentNode.children[c]; !exists {
61                 break
62             }
63             currentNode = currentNode.children[c]
64             if currentNode.isWord {
65                 lastValidIndex = i
66             }
67         }
68
69         if lastValidIndex != -1 {
70             result = append(result, sentence[startIndex:lastValidIndex+1])
71             startIndex = lastValidIndex + 1
72         } else {
73             result = append(result, string(sentence[startIndex]))
74             startIndex++
75         }
76     }
77
78     fmt.Println(strings.Join(result, ","))
79 }
```

来自: 华为OD 机试 – 中文分词模拟器 (2025 B卷 200分)-CSDN博客

华为OD机试 - 模拟工作队列 (2025 B卷 200分)_2025 华为od机考 队列-CSDN博客

模拟工作队列

华为OD机试真题目录: [点击查看](#)

2025 B卷 200分题型

题目描述

让我们来模拟一个工作队列的运作，有一个任务提交者和若干任务执行者，执行者从1开始编号。提交者会在给定的时刻向工作队列提交任务，任务有执行所需的时间，执行者取出任务的时刻加上执行时间即为任务完成的时刻，执行者完成任务变为空闲的时刻会从工作队列中取最老的任务执行，若这一时刻有多个空闲的执行者，其中优先级最高的会执行这个任务。编号小的执行者优先级高，初始状态下所有执行者都空闲。

工作队列有最大长度限制，当工作队列满有新的任务加入队列时，队列中最老的任务会被丢弃。

在工作队列满的情况下，当执行者变为空闲的时刻和新的任务提交的时刻相同时，队列中最老的任务被取出执行，新的任务加入队列。

输入描述

输入为两行。

- 第一行为 $2N$ 个正整数，代表提交者提交的 N 个任务的时刻和执行时间。第一个数字是第一个任务的提交时刻，第二个数字是第一个任务的执行时间，以此类推。用例保证提交时刻不会重复，任务按提交时刻升序排列。
- 第二行为两个数字，分别为工作队列的最大长度和执行者的数量。

两行的数字都由空格分割。 N 不超过 20，数字为不超过 1000 的正整数。

输出描述

输出两个数字，分别为最后一个任务执行完成的时刻和被丢弃的任务的数量，数字由空格分隔。

用例1

输入

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

Plain Text

输出

```
Plain Text |  
1 7 0
```

说明

两个执行者，队列最大长度3，所有任务均能正常加入并执行，无任务丢弃。

用例2

输入

```
Plain Text |  
1 1 6 2 4 4 3 6 3  
2 1 2
```

输出

```
Plain Text |  
1 10 0
```

说明

队列最大长度1较小，但任务提交和执行顺序使得没有任务丢弃。

题解

思路：进行 模拟求解 按照题目要求进行模拟求解即可。注意以下几个优化点或注意点。其它细节逻辑可参照下面代码

1. 使用 优先队列 来快速把完成的任务执行者放入空闲列表中。优先队列按照任务完成时间进行排序。完成时间相同的按照id进行排序。
2. 当执行者变为空闲的时刻和新的任务提交的时刻相同时，队列中最老的任务被取出执行 题目要求决定要先将忙碌执行者解放出来去处理任务。在考虑将新添加的任务加入队列。

C++

```
1 #include <functional>
2 #include<iostream>
3 #include <iostream>
4 #include<vector>
5 #include<string>
6 #include <utility>
7 #include <sstream>
8 #include<algorithm>
9 #include<list>
10 #include<queue>
11 #include<map>
12 using namespace std;
13
14 // 通用 split 函数
15 vector<string> split(const string& str, const string& delimiter) {
16     vector<string> result;
17     size_t start = 0;
18     size_t end = str.find(delimiter);
19     while (end != string::npos) {
20         result.push_back(str.substr(start, end - start));
21         start = end + delimiter.length();
22         end = str.find(delimiter, start);
23     }
24     // 添加最后一个部分
25     result.push_back(str.substr(start));
26     return result;
27 }
28
29 struct Compare{
30     bool operator()(pair<int, int> a, pair<int, int> b) {
31         // 空闲时间相同，按照id升序排
32         if (a.second == b.second) {
33             return a.first > b.first;
34         }
35         // 空闲时间小的优先级高
36         return a.second > b.second;
37     }
38 };
39
40
41 int main() {
42     string line;
43     getline(cin, line);
44     vector<string> task = split(line, " ");
```

```

45     int n = task.size() / 2;
46     vector<pair<int, int>> taskList(n);
47     for (int i = 0; i < n ; i++) {
48         taskList[i] = {stoi(task[2 * i]), stoi(task[2 * i + 1])};
49     }
50
51     getline(cin , line);
52     vector<string> tmp = split(line, " ");
53
54     int capacity = stoi(tmp[0]);
55     int executorCount = stoi(tmp[1]);
56
57     // 用于队列中的任务
58     queue<pair<int, int>> taskQueue;
59     // 根据结束时间进行排序
60     priority_queue<pair<int, int>, vector<pair<int, int>>, Compare> busyE
xecutor;
61     // 存储当前空闲执行者id
62     priority_queue<int, vector<int>, greater<int>> freeExecutor;
63
64     for (int i = 1 ; i <= executorCount; i++) {
65         freeExecutor.push(i);
66     }
67
68     int currrentTime = 1;
69     int dropOutCount = 0;
70     int pos = 0;
71     int lastFinishTime = -1;
72     while (pos < n || !taskQueue.empty()) {
73         // 将结束工作的生产者放到空闲队列中
74         while (!busyExecutor.empty() && busyExecutor.top().second <= curr
rentTime) {
75             pair<int, int> tmp = busyExecutor.top();
76             busyExecutor.pop();
77             // 放入空闲队列中
78             freeExecutor.push(tmp.first);
79         }
80
81         // 空闲生产者去任务队列中找任务
82         if (!freeExecutor.empty()) {
83             while (!freeExecutor.empty() && !taskQueue.empty()) {
84                 int executorId = freeExecutor.top();
85                 freeExecutor.pop();
86
87                 pair<int, int> task = taskQueue.front();
88                 taskQueue.pop();
89

```

```

90     d});
91     lastFinishTime = max(lastFinishTime, currrentTime + task.
92     second);
93 }
94
95 // 尝试将当前时机投递的任务放入队列中
96 if (pos < n && currrentTime >= taskList[pos].first) {
97     pair<int, int> task = taskList[pos];
98     pos++;
99     taskQueue.push(task);
100
101
102 // 丢弃情况情况处理
103 while (taskQueue.size() > capacity) {
104     taskQueue.pop();
105     dropOutCount++;
106 }
107 }
108
109 // 空闲生产者去任务队列中找任务执行
110 if (!freeExecutor.empty()) {
111     while (!freeExecutor.empty() && !taskQueue.empty()) {
112         int executorId = freeExecutor.top();
113         freeExecutor.pop();
114         pair<int, int> task = taskQueue.front();
115         taskQueue.pop();
116         busyExecutor.push({executorId, currrentTime + task.secon
117     d});
118     lastFinishTime = max(lastFinishTime, currrentTime + task.
119     second);
120 }
121 currrentTime++;
122 }
123 cout << lastFinishTime << " " << dropOutCount;
124

```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取任务信息
8         String[] taskInput = scanner.nextLine().split(" ");
9         int n = taskInput.length / 2;
10        List<int[]> taskList = new ArrayList<>();
11        for (int i = 0; i < n; i++) {
12            taskList.add(new int[]{Integer.parseInt(taskInput[2 * i]), Integer.parseInt(taskInput[2 * i + 1])});
13        }
14
15        // 读取容量和执行者数量
16        String[] config = scanner.nextLine().split(" ");
17        int capacity = Integer.parseInt(config[0]);
18        int executorCount = Integer.parseInt(config[1]);
19
20        // 任务队列
21        Queue<int[]> taskQueue = new LinkedList<>();
22        // 忙碌执行者队列 (根据任务结束时间排序)
23        PriorityQueue<int[]> busyExecutor = new PriorityQueue<>(Comparator.comparingInt(a -> a[1]));
24        // 空闲执行者队列 (存储执行者 ID, 按升序排序)
25        PriorityQueue<Integer> freeExecutor = new PriorityQueue<>();
26        for (int i = 1; i <= executorCount; i++) {
27            freeExecutor.add(i);
28        }
29
30        int currentTime = 1, dropOutCount = 0, pos = 0, lastFinishTime = -1;
31
32        while (pos < n || !taskQueue.isEmpty()) {
33            // 释放当前时间结束的执行者
34            while (!busyExecutor.isEmpty() && busyExecutor.peek()[1] <= currentTime) {
35                freeExecutor.add(busyExecutor.poll()[0]);
36            }
37
38            // 空闲执行者取任务执行
39            while (!freeExecutor.isEmpty() && !taskQueue.isEmpty()) {
40                int executorId = freeExecutor.poll();
```

```
41             int[] task = taskQueue.poll();
42             busyExecutor.add(new int[]{executorId, currentTime + task
43 [1]});                                lastFinishTime = Math.max(lastFinishTime, currentTime + ta
44         }                                         sk[1]);
45
46     // 处理新到达的任务
47     if (pos < n && currentTime >= taskList.get(pos)[0]) {
48         taskQueue.add(taskList.get(pos));
49         pos++;
50
51     // 任务队列超出容量则丢弃任务
52     while (taskQueue.size() > capacity) {
53         taskQueue.poll();
54         dropOutCount++;
55     }
56
57     // 任务投递后再次尝试分配给空闲执行者
58     while (!freeExecutor.isEmpty() && !taskQueue.isEmpty()) {
59         int executorId = freeExecutor.poll();
60         int[] task = taskQueue.poll();
61         busyExecutor.add(new int[]{executorId, currentTime + t
ask[1]});                                lastFinishTime = Math.max(lastFinishTime, currentTime
+ task[1]);
62         }
63     }
64
65     currentTime++;
66 }
67
68 System.out.println(lastFinishTime + " " + dropOutCount);
69 }
70 }
71 }
```

Python

```
1 import sys
2 import heapq
3
4 # 读取任务信息
5 task_input = list(map(int, sys.stdin.readline().strip().split()))
6 n = len(task_input) // 2
7 task_list = [(task_input[2 * i], task_input[2 * i + 1]) for i in range(n)]
8
9 # 读取队列容量和执行者数量
10 capacity, executor_count = map(int, sys.stdin.readline().strip().split())
11
12 # 任务队列
13 task_queue = []
14 # 忙碌执行者队列 (存储 (结束时间, 执行者ID))
15 busy_executor = []
16 # 空闲执行者队列 (存储执行者 ID, 按升序排序)
17 free_executor = list(range(1, executor_count + 1))
18 heapq.heapify(free_executor)
19
20 current_time = 1
21 drop_out_count = 0
22 pos = 0
23 last_finish_time = -1
24
25 while pos < n or task_queue:
26     # 释放已完成的执行者
27     while busy_executor and busy_executor[0][0] <= current_time:
28         _, executor_id = heapq.heappop(busy_executor)
29         heapq.heappush(free_executor, executor_id)
30
31     # 空闲执行者取任务执行
32     while free_executor and task_queue:
33         executor_id = heapq.heappop(free_executor)
34         start_time, duration = task_queue.pop(0)
35         heapq.heappush(busy_executor, (current_time + duration, executor_i
d))
36         last_finish_time = max(last_finish_time, current_time + duration)
37
38     # 处理新到达的任务
39     if pos < n and current_time >= task_list[pos][0]:
40         task_queue.append(task_list[pos])
41         pos += 1
42
43     # 任务队列超出容量则丢弃任务
```

```
44     while len(task_queue) > capacity:
45         task_queue.pop(0)
46         drop_out_count += 1
47
48     # 任务投递后再次尝试分配给空闲执行者
49     while free_executor and task_queue:
50         executor_id = heapq.heappop(free_executor)
51         start_time, duration = task_queue.pop(0)
52         heapq.heappush(busy_executor, (current_time + duration, execut
53         or_id))
54         last_finish_time = max(last_finish_time, current_time + durati
55         on)
56
57     print(last_finish_time, drop_out_count)
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 // 通用 split 函数
9 function split(str, delimiter) {
10     return str.trim().split(delimiter);
11 }
12
13 let inputLines = [];
14
15 rl.on("line", (line) => {
16     inputLines.push(line.trim());
17     if (inputLines.length === 2) {
18         main();
19     }
20 });
21
22 function main() {
23     let task = split(inputLines[0], " ");
24     let n = task.length / 2;
25     let taskList = [];
26     for (let i = 0; i < n; i++) {
27         taskList.push([parseInt(task[2 * i]), parseInt(task[2 * i + 1])]);
28     }
29
30     let tmp = split(inputLines[1], " ");
31     let capacity = parseInt(tmp[0]);
32     let executorCount = parseInt(tmp[1]);
33
34     // 任务队列
35     let taskQueue = [];
36     // 忙碌的执行者队列 (小根堆, 按空闲时间排序)
37     let busyExecutor = [];
38     // 空闲的执行者 (小根堆)
39     let freeExecutor = [];
40
41     for (let i = 1; i <= executorCount; i++) {
42         freeExecutor.push(i);
43     }
44 }
```

```

45     let currentTime = 1;
46     let dropOutCount = 0;
47     let pos = 0;
48     let lastFinishTime = -1;
49
50     while (pos < n || taskQueue.length > 0) {
51         // 将空闲的执行者放回空闲队列
52         busyExecutor.sort((a, b) => a[1] - b[1]); // 按空闲时间排序
53         while (busyExecutor.length > 0 && busyExecutor[0][1] <= currentTim
e) {
54             let tmp = busyExecutor.shift();
55             freeExecutor.push(tmp[0]);
56         }
57
58         // 空闲执行者去任务队列中找任务
59         while (freeExecutor.length > 0 && taskQueue.length > 0) {
60             let executorId = freeExecutor.shift();
61             let task = taskQueue.shift();
62             busyExecutor.push([executorId, currentTime + task[1]]);
63             lastFinishTime = Math.max(lastFinishTime, currentTime + task
[1]);
64         }
65
66         // 处理新任务投递
67         if (pos < n && currentTime >= taskList[pos][0]) {
68             taskQueue.push(taskList[pos]);
69             pos++;
70
71             // 丢弃超出容量的任务
72             while (taskQueue.length > capacity) {
73                 taskQueue.shift();
74                 dropOutCount++;
75             }
76         }
77
78         // 再次让空闲执行者取任务
79         while (freeExecutor.length > 0 && taskQueue.length > 0) {
80             let executorId = freeExecutor.shift();
81             let task = taskQueue.shift();
82             busyExecutor.push([executorId, currentTime + task[1]]);
83             lastFinishTime = Math.max(lastFinishTime, currentTime + task
[1]);
84         }
85
86         currentTime++;
87     }
88

```

```
89     console.log(lastFinishTime, dropOutCount);  
89 ,
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9     "container/heap"
10    )
11
12 // 任务队列
13 type TaskQueue struct {
14     tasks []Task
15 }
16
17 func (q *TaskQueue) Push(task Task) {
18     q.tasks = append(q.tasks, task)
19 }
20
21 func (q *TaskQueue) Pop() Task {
22     task := q.tasks[0]
23     q.tasks = q.tasks[1:]
24     return task
25 }
26
27 func (q *TaskQueue) Size() int {
28     return len(q.tasks)
29 }
30
31 // 任务结构体
32 type Task struct {
33     startTime int
34     duration  int
35 }
36
37 // 执行者优先队列 (最小堆)
38 type ExecutorHeap []Executor
39
40 func (h ExecutorHeap) Len() int           { return len(h) }
41 func (h ExecutorHeap) Less(i, j int) bool { return h[i].freeTime < h[j].f
reeTime || (h[i].freeTime == h[j].freeTime && h[i].id < h[j].id) }
42 func (h ExecutorHeap) Swap(i, j int)      { h[i], h[j] = h[j], h[i] }
43
```

```
44 func (h *ExecutorHeap) Push(x interface{}) {
45     *h = append(*h, x.(Executor))
46 }
47
48 func (h *ExecutorHeap) Pop() interface{} {
49     old := *h
50     n := len(old)
51     x := old[n-1]
52     *h = old[0 : n-1]
53     return x
54 }
55
56 // 执行者结构体
57 type Executor struct {
58     id      int
59     freeTime int
60 }
61
62 func main() {
63     scanner := bufio.NewScanner(os.Stdin)
64
65     // 读取任务
66     scanner.Scan()
67     taskStr := scanner.Text()
68     taskData := strings.Split(taskStr, " ")
69
70     n := len(taskData) / 2
71     taskList := make([]Task, n)
72
73     for i := 0; i < n; i++ {
74         startTime, _ := strconv.Atoi(taskData[2*i])
75         duration, _ := strconv.Atoi(taskData[2*i+1])
76         taskList[i] = Task{startTime, duration}
77     }
78
79     // 读取执行者和容量
80     scanner.Scan()
81     params := strings.Split(scanner.Text(), " ")
82     capacity, _ := strconv.Atoi(params[0])
83     executorCount, _ := strconv.Atoi(params[1])
84
85     // 任务队列
86     var taskQueue TaskQueue
87
88     // 忙碌执行者最小堆
89     busyExecutor := &ExecutorHeap{}
90     heap.Init(busyExecutor)
```

```
91 // 空闲执行者最小堆
92 freeExecutor := &ExecutorHeap{}
93 heap.Init(freeExecutor)
94
95 for i := 1; i <= executorCount; i++ {
96     heap.Push(freeExecutor, Executor{id: i, freeTime: 0})
97 }
98
99
100 currentTime := 1
101 dropOutCount := 0
102 pos := 0
103 lastFinishTime := -1
104
105 for pos < n || taskQueue.Size() > 0 {
106     // 将空闲执行者放回空闲队列
107     for busyExecutor.Len() > 0 && (*busyExecutor)[0].freeTime <= currentTime {
108         executor := heap.Pop(busyExecutor).(Executor)
109         heap.Push(freeExecutor, executor)
110     }
111
112     // 空闲执行者执行任务
113     for freeExecutor.Len() > 0 && taskQueue.Size() > 0 {
114         executor := heap.Pop(freeExecutor).(Executor)
115         task := taskQueue.Pop()
116         executor.freeTime = currentTime + task.duration
117         heap.Push(busyExecutor, executor)
118         if executor.freeTime > lastFinishTime {
119             lastFinishTime = executor.freeTime
120         }
121     }
122
123     // 添加新任务到队列
124     if pos < n && currentTime >= taskList[pos].startTime {
125         taskQueue.Push(taskList[pos])
126         pos++
127
128     // 丢弃超出容量的任务
129     for taskQueue.Size() > capacity {
130         taskQueue.Pop()
131         dropOutCount++
132     }
133 }
134
135     // 再次尝试分配任务
136     for freeExecutor.Len() > 0 && taskQueue.Size() > 0 {
```

```
137     executor := heap.Pop(freeExecutor).(Executor)
138     task := taskQueue.Pop()
139     executor.freeTime = currentTime + task.duration
140     heap.Push(busyExecutor, executor)
141     if executor.freeTime > lastFinishTime {
142         lastFinishTime = executor.freeTime
143     }
144 }
145
146     currentTime++
147 }
148
149 fmt.Println(lastFinishTime, dropOutCount)
```

| 来自: 华为OD机试 – 模拟工作队列 (2025 B卷 200分)_2025 华为od机考 队列-CSDN博客

华为OD机试 - 最小矩阵宽度 (2025 B卷 200分)- CSDN博客

最小矩阵宽度

华为OD机试真题目录: [点击查看](#)

2025B卷 200分题型

题目描述

给定一个矩阵，包含 $N * M$ 个整数，和一个包含 K 个整数的数组。

现在要求在这个矩阵中找一个宽度最小的子矩阵，要求子矩阵包含数组中所有的整数。

输入描述

第一行输入两个正整数 N, M ，表示矩阵大小。

接下来 N 行 M 列表示矩阵内容。

下一行包含一个正整数 K 。

下一行包含 K 个整数，表示所需包含的数组， K 个整数可能存在重复数字。

所有输入数据小于1000。

输出描述

输出包含一个整数，表示满足要求子矩阵的最小宽度，若找不到，输出-1。

用例1

输入

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

输出

Plain Text

1 2

说明

矩阵第0、1列包含了1, 2, 3, 矩阵第3, 4列包含了1, 2, 3

用例2

输入

Plain Text

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

输出

Plain Text

1 5

说明

矩阵第1、2、3、4、5列包含了1、1、4

题解

思路： 双指针 处理，基本逻辑如下，判断双指针区域中的数是否能完全匹配上待匹配数组

- 不能：让右指针右移，增加区域。
- 能：让左指针右移，尝试获取更少的宽度。

代码可以使用哈希表来加速判断双指针区域是否已经完全匹配。

定义 count 表示双指针区域中已经匹配上待匹配数字的数量，使用 map<int, int> mp 存储统计待匹配数组中各个数字出现次数，使用 map<int, int> total

统计双指针区域中出现数字的数量，处理逻辑如下：

- 增加右边界，新增一列时，假设当前新增这一列元素为 nums1 nums2..numsn 需要依次去变更 total 值，例如处理这一列的第一个值时，会进行 total[num1]++，同时判断 total[num1] <= mp[num1]，如果满足条件则说明额外匹配上一个元素，count+= 1。为这一列重复执行前面的操作

作。如果新增这一列之后 `count == k` 说明已经完整匹配上了。没有完整匹配则右边界继续右移，完整匹配考虑缩小区域，进行左指针右移。

2. 左指针右移，类似于增加右边界的反操作。减少 `total` 的值，`total[nums1]--`，如果此时 `total[num1] < mp[num1]` 则 `count -= 1`。左指针右移的结束条件为 `count < k`。
3. 重复1 2 操作，直到 `right == m` 时结束。输出记录的最小宽度即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<climits>
11 using namespace std;
12
13 int main() {
14     int n , m;
15     cin >> n >> m;
16     vector<vector<int>> grid(n, vector<int>(m, 0));
17     for (int i = 0; i < n; i++) {
18         for (int j = 0; j < m; j++) {
19             cin >> grid[i][j];
20         }
21     }
22     int k;
23     cin >> k;
24     vector<int> ans(k);
25     // 统计待匹配数组中各个数字出现次数
26     map<int, int> mp;
27     for (int i = 0; i < k; i++) {
28         cin>>ans[i];
29         mp[ans[i]]++;
30     }
31     // 统计双指针区域中出现数字的数量
32     map<int, int> total;
33     int res = INT_MAX;
34     // 双指针区域中已经匹配上 待匹配数字的数量
35     int count  = 0;
36
37     // 统计第一列出现的各个数字的数量 以及更新匹配上的数量
38     for (int i = 0; i < n; i++) {
39         int num = grid[i][0];
40         total[num]++;
41         if (total[num] <= mp[num]) {
42             count++;
43         }
44     }
```

```

45     // 已经完全匹配
46     if (count == k) {
47         cout << 1;
48         return 0;
49     }
50
51     int left = 0, right = 0;
52     while (right < m) {
53         // 增加右边界 双指针区域并没有完全匹配上所有元素，右边界右移，增加区域
54         if (count < k) {
55             right++;
56             if (right >= m) {
57                 continue;
58             }
59             for (int i = 0; i < n; i++) {
60                 int num = grid[i][right];
61                 total[num]++;
62                 if (total[num] <= mp[num]) {
63                     count++;
64                 }
65             }
66             if (count == k) {
67                 res = min(res, right - left + 1);
68             }
69         // 尝试减少左边界 已经完全匹配的情况下，尝试左边界右移缩小宽度
70     } else {
71         for (int i = 0; i < n; i++) {
72             int num = grid[i][left];
73             total[num]--;
74             if (total[num] < mp[num]) {
75                 count--;
76             }
77         }
78         left++;
79         if (count == k) {
80             res = min(res, right - left + 1);
81         }
82     }
83 }
84
85 if (res != INT_MAX) {
86     cout << res;
87 } else {
88     cout << -1;
89 }
90
91 return 0;

```

Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取 n 和 m
8         int n = scanner.nextInt();
9         int m = scanner.nextInt();
10
11        // 读取矩阵 grid
12        int[][] grid = new int[n][m];
13        for (int i = 0; i < n; i++) {
14            for (int j = 0; j < m; j++) {
15                grid[i][j] = scanner.nextInt();
16            }
17        }
18
19        // 读取 k
20        int k = scanner.nextInt();
21        int[] ans = new int[k];
22        Map<Integer, Integer> mp = new HashMap<>();
23
24        // 统计待匹配数组中各个数字出现次数
25        for (int i = 0; i < k; i++) {
26            ans[i] = scanner.nextInt();
27            mp.put(ans[i], mp.getOrDefault(ans[i], 0) + 1);
28        }
29
30        // 统计双指针区域中出现数字的数量
31        Map<Integer, Integer> total = new HashMap<>();
32        int count = 0, res = Integer.MAX_VALUE;
33
34        // 统计第一列出现的各个数字的数量，并更新匹配上的数量
35        for (int i = 0; i < n; i++) {
36            int num = grid[i][0];
37            total.put(num, total.getOrDefault(num, 0) + 1);
38            if (total.get(num) <= mp.getOrDefault(num, 0)) {
39                count++;
40            }
41        }
42
43        // 已经完全匹配
44        if (count == k) {
```

```

45         System.out.println(1);
46         return;
47     }
48
49     int left = 0, right = 0;
50     while (right < m) {
51         // 增加右边界，双指针区域并没有完全匹配上所有元素
52         if (count < k) {
53             right++;
54             if (right >= m) continue;
55
56             for (int i = 0; i < n; i++) {
57                 int num = grid[i][right];
58                 total.put(num, total.getOrDefault(num, 0) + 1);
59                 if (total.get(num) <= mp.getOrDefault(num, 0)) {
60                     count++;
61                 }
62             }
63
64             if (count == k) {
65                 res = Math.min(res, right - left + 1);
66             }
67             // 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
68         } else {
69             for (int i = 0; i < n; i++) {
70                 int num = grid[i][left];
71                 total.put(num, total.get(num) - 1);
72                 if (total.get(num) < mp.getOrDefault(num, 0)) {
73                     count--;
74                 }
75             }
76             left++;
77             if (count == k) {
78                 res = Math.min(res, right - left + 1);
79             }
80         }
81     }
82
83     System.out.println(res != Integer.MAX_VALUE ? res : -1);
84 }
85 }
```

Python

```
1 import sys
2
3 # 读取输入
4 n, m = map(int, sys.stdin.readline().split())
5
6 # 读取矩阵 grid
7 grid = [list(map(int, sys.stdin.readline().split())) for _ in range(n)]
8
9 # 读取 k 和 ans
10 k = int(sys.stdin.readline())
11 ans = list(map(int, sys.stdin.readline().split()))
12
13 # 统计待匹配数组中各个数字出现次数
14 mp = {}
15 for num in ans:
16     mp[num] = mp.get(num, 0) + 1
17
18 # 统计双指针区域中出现数字的数量
19 total = {}
20 count = 0
21 res = float('inf')
22
23 # 统计第一列出现的各个数字的数量，并更新匹配上的数量
24 for i in range(n):
25     num = grid[i][0]
26     total[num] = total.get(num, 0) + 1
27     if total[num] <= mp.get(num, 0):
28         count += 1
29
30 # 已经完全匹配
31 if count == k:
32     print(1)
33     sys.exit()
34
35 left, right = 0, 0
36 while right < m:
37     # 增加右边界，双指针区域并没有完全匹配上所有元素
38     if count < k:
39         right += 1
40         if right >= m:
41             continue
42
43         for i in range(n):
44             num = grid[i][right]
```

```
45         total[num] = total.get(num, 0) + 1
46         if total[num] <= mp.get(num, 0):
47             count += 1
48
49     if count == k:
50         res = min(res, right - left + 1)
51 # 尝试减少左边界, 已经完全匹配的情况下尝试缩小宽度
52 else:
53     for i in range(n):
54         num = grid[i][left]
55         total[num] -= 1
56         if total[num] < mp.get(num, 0):
57             count -= 1
58
59     left += 1
60     if count == k:
61         res = min(res, right - left + 1)
62
63 print(res if res != float('inf') else -1)
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = [];
9 rl.on('line', (line) => {
10     input.push(line);
11 }).on('close', () => {
12     // 读取 n 和 m
13     let [n, m] = input[0].split(' ').map(Number);
14
15     // 读取矩阵 grid
16     let grid = [];
17     for (let i = 1; i <= n; i++) {
18         grid.push(input[i].split(' ').map(Number));
19     }
20
21     // 读取 k
22     let k = Number(input[n + 1]);
23     let ans = input[n + 2].split(' ').map(Number);
24
25     // 统计待匹配数组中各个数字出现次数
26     let mp = {};
27     ans.forEach(num => mp[num] = (mp[num] || 0) + 1);
28
29     // 统计双指针区域中出现数字的数量
30     let total = {};
31     let count = 0, res = Infinity;
32
33     // 统计第一列出现的各个数字的数量，并更新匹配上的数量
34     for (let i = 0; i < n; i++) {
35         let num = grid[i][0];
36         total[num] = (total[num] || 0) + 1;
37         if (total[num] <= (mp[num] || 0)) {
38             count++;
39         }
40     }
41
42     // 已经完全匹配
43     if (count === k) {
44         console.log(1);
45     }
46 }
```

```

45         return;
46     }
47
48     let left = 0, right = 0;
49     while (right < m) {
50         // 增加右边界，双指针区域并没有完全匹配上所有元素
51         if (count < k) {
52             right++;
53             if (right >= m) continue;
54
55             for (let i = 0; i < n; i++) {
56                 let num = grid[i][right];
57                 total[num] = (total[num] || 0) + 1;
58                 if (total[num] <= (mp[num] || 0)) {
59                     count++;
60                 }
61             }
62
63             if (count === k) {
64                 res = Math.min(res, right - left + 1);
65             }
66             // 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
67         } else {
68             for (let i = 0; i < n; i++) {
69                 let num = grid[i][left];
70                 total[num]--;
71                 if (total[num] < (mp[num] || 0)) {
72                     count--;
73                 }
74             }
75             left++;
76             if (count === k) {
77                 res = Math.min(res, right - left + 1);
78             }
79         }
80     }
81
82     console.log(res !== Infinity ? res : -1);
83 }:
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13
14     // 读取 n 和 m
15     nm, _ := reader.ReadString('\n')
16     nmSlice := splitToInt(nm)
17     n, m := nmSlice[0], nmSlice[1]
18
19     // 读取矩阵 grid
20     grid := make([][]int, n)
21     for i := 0; i < n; i++ {
22         line, _ := reader.ReadString('\n')
23         grid[i] = splitToInt(line)
24     }
25
26     // 读取 k
27     kStr, _ := reader.ReadString('\n')
28     k, _ := strconv.Atoi(strings.TrimSpace(kStr))
29
30     // 读取 ans 数组
31     ansStr, _ := reader.ReadString('\n')
32     ans := splitToInt(ansStr)
33
34     // 统计待匹配数组中各个数字出现次数
35     mp := make(map[int]int)
36     for _, num := range ans {
37         mp[num]++
38     }
39
40     // 统计双指针区域中出现数字的数量
41     total := make(map[int]int)
42     count := 0
43     res := int(^uint(0) >> 1) // 设置 res 为 int 最大值
44 }
```

```
45     // 统计第一列出现的各个数字的数量，并更新匹配上的数量
46     for i := 0; i < n; i++ {
47         num := grid[i][0]
48         total[num]++
49         if total[num] <= mp[num] {
50             count++
51         }
52     }
53
54     // 已经完全匹配
55     if count == k {
56         fmt.Println(1)
57         return
58     }
59
60     left, right := 0, 0
61     for right < m {
62         // 增加右边界，双指针区域并没有完全匹配上所有元素
63         if count < k {
64             right++
65             if right >= m {
66                 continue
67             }
68
69             for i := 0; i < n; i++ {
70                 num := grid[i][right]
71                 total[num]++
72                 if total[num] <= mp[num] {
73                     count++
74                 }
75             }
76
77             if count == k {
78                 res = min(res, right-left+1)
79             }
80         } else { // 尝试减少左边界，已经完全匹配的情况下尝试缩小宽度
81             for i := 0; i < n; i++ {
82                 num := grid[i][left]
83                 total[num]--
84                 if total[num] < mp[num] {
85                     count--
86                 }
87             }
88             left++
89             if count == k {
90                 res = min(res, right-left+1)
91             }
92 }
```

```
92     }
93 }
94
95 if res != int(^uint(0)>>1) {
96     fmt.Println(res)
97 } else {
98     fmt.Println(-1)
99 }
100 }
101
102 // splitToInt 将输入字符串按空格拆分并转换为整数数组
103 func splitToInt(s string) []int {
104     fields := strings.Fields(s)
105     res := make([]int, len(fields))
106     for i, v := range fields {
107         res[i], _ = strconv.Atoi(v)
108     }
109     return res
110 }
111
112 // min 返回两个整数中的最小值
113 func min(a, b int) int {
114     if a < b {
115         return a
116     }
117     return b
118 }
```

| 来自: 华为od 机试 – 最小矩阵宽度 (2025 B卷 200分)-CSDN博客

华为OD机试 - 二叉树的广度优先遍历 (2025 B卷 200分)_华为od机试 二叉树的广度优先遍历-CSDN博客

二叉树的广度优先遍历

华为OD机试 真题目录: [点击查看](#)

2025B卷 200分题型

题目描述

有一棵二叉树，每个节点由一个大写字母标识(最多26个节点)。

现有两组字母，分别表示后序遍历（左孩子->右孩子->父节点）和中序遍历（左孩子->父节点->右孩子）的结果，请你输出层序遍历的结果。

输入描述

每个输入文件一行，第一个字符串表示后序遍历结果，第二个字符串表示中序遍历结果。（每串只包含大写字母）

中间用单空格分隔。

输出描述

输出仅一行，表示层序遍历的结果，结尾换行。

用例1

输入

```
1 CBEFDA CBAEDF
```

输出

```
1 ABDCEF
```

说明

Plain Text |

```
1 二叉树为:  
2  
3      A  
4      /   \  
5      B     D  
6      /     /   \  
7      C     E     F
```

题解

思路： 二叉树， 中序遍历(左 -> 父 -> 右), 后序遍历(左 -> 右 -> 父亲)

1. 通过后序和中序遍历的序列还原出原二叉树的结构，下面简单说一下还原规律，以示例1为说明. 后

序： CBEFDA 中序： CBAEDF

- 后序遍历规律(左孩子->右孩子->父节点)可以得出最后一个值肯定是父节点的值，此时父节点为 A
- 在中序遍历中找到父节点的值， CB A EDF ,由中序遍历的规律(左孩子->父节点->右孩子)可得 A 的左子树包含 CB 两个元素(这也是它们中序遍历的输出) 右子树包含 EDF 三个元素(中序遍历输出)
- 根据2得出左右子树的长度，可以得出左子树的后序遍历输出为 CB 右子树的后序遍历结果为 EFD. 左子树的中序遍历的输出为 CB ，右子树的中序遍历的输出为 EDF
- 借助上面提到的三个规律，使用 递归 就可以还原出二叉树的结构。整个树的还原其实是 自下而上的 . 递归终止条件是到达叶子节点。

2. 通过第一步递归还原出二叉树的结构之后，接下来就是对结果进行层序遍历，一层一层进行输出二叉树的值，从上到下，从左到右。一般层序遍历都是采用 BFS 去模拟的。下面代码中使用 栈 去实现 BFS，并将记录BFS遍历的节点值输出顺序，最后拼接成字符串输出即可。

二叉树还是属于比较基础的数据结构，建议多花点时间看看二叉树相关的。额外思考，如果是给出二叉树的前序遍历和中序遍历，你能还原出二叉树的原始结构吗？

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<queue>
8 using namespace std;
9
10 struct Node{
11     Node* left;
12     Node* right;
13     char value;
14     Node(){}
15     Node(char value):value(value), left(nullptr), right(nullptr){}
16 };
17
18 // 通用 split 函数
19 vector<string> split(const string& str, const string& delimiter) {
20     vector<string> result;
21     size_t start = 0;
22     size_t end = str.find(delimiter);
23     while (end != string::npos) {
24         result.push_back(str.substr(start, end - start));
25         start = end + delimiter.length();
26         end = str.find(delimiter, start);
27     }
28     // 添加最后一个部分
29     result.push_back(str.substr(start));
30     return result;
31 }
32
33 // 通过后序和中序还原二叉树
34 Node* generateNode(string postOrder, string inOrder) {
35     if (postOrder.empty()) {
36         return nullptr;
37     }
38     int n = postOrder.size();
39     Node* root = new Node(postOrder[n-1]);
40     // 确定左子树的长度
41     int j = 0;
42     for (; j < n; j++) {
43         if (postOrder[n-1] == inOrder[j]) {
44             break;
```

```

45         }
46     }
47     // 确定左右子树中序遍历结果
48     string inOrderLeft = inOrder.substr(0, j);
49     string inOrderRight = inOrder.substr(j+1);
50     // 确定左右子树后序遍历结果
51     string postOrderLeft = postOrder.substr(0, j);
52     string postOrderRight = postOrder.substr(j, n-j-1);
53     if (!inOrderLeft.empty()) {
54         root->left = generateNode(postOrderLeft, inOrderLeft);
55     }
56     if (!inOrderRight.empty()) {
57         root->right = generateNode(postOrderRight, inOrderRight);
58     }
59     return root;
60 }
61
62 string BFS(Node* root) {
63     string res = "";
64     if (root == nullptr) {
65         return res;
66     }
67     queue<Node*> q;
68     q.push(root);
69     while (!q.empty()) {
70         Node* tmp = q.front();
71         q.pop();
72         res += string(1, tmp->value);
73         if (tmp -> left != nullptr) {
74             q.push(tmp->left);
75         }
76         if (tmp ->right != nullptr) {
77             q.push(tmp->right);
78         }
79     }
80     return res;
81 }
82
83
84
85
86
87 int main() {
88     string s;
89     getline(cin , s);
90     if (s.empty()) {
91         cout << "";

```

```
92         return 0;
93     }
94     vector<string> tmp = split(s, " ");
95     string postOrder = tmp[0];
96     string inorder = tmp[1];
97     Node* root = generateNode(postOrder,inorder);
98     string res = BFS(root);
99     cout << res;
100    return 0;
```

JAVA

```
1 import java.util.*;
2
3 class Main {
4     static class Node {
5         Node left, right;
6         char value;
7         Node(char value) {
8             this.value = value;
9             this.left = null;
10            this.right = null;
11        }
12    }
13
14 // 根据后序遍历和中序遍历构造二叉树
15 static Node generateNode(String postOrder, String inOrder) {
16     if (postOrder.isEmpty()) return null;
17
18     int n = postOrder.length();
19     Node root = new Node(postOrder.charAt(n - 1));
20
21     int j = inOrder.indexOf(postOrder.charAt(n - 1)); // 确定左子树大小
22
23     // 递归构造左右子树
24     root.left = generateNode(postOrder.substring(0, j), inOrder.substring(0, j));
25     root.right = generateNode(postOrder.substring(j, n - 1), inOrder.substring(j + 1));
26
27     return root;
28 }
29
30 // 层序遍历 BFS
31 static String BFS(Node root) {
32     if (root == null) return "";
33
34     StringBuilder res = new StringBuilder();
35     Queue<Node> queue = new LinkedList<>();
36     queue.add(root);
37
38     while (!queue.isEmpty()) {
39         Node tmp = queue.poll();
40         res.append(tmp.value);
41
42         if (tmp.left != null) queue.add(tmp.left);
43     }
44
45     return res.toString();
46 }
```

```
43             if (tmp.right != null) queue.add(tmp.right);
44     }
45
46     return res.toString();
47 }
48
49 public static void main(String[] args) {
50     Scanner scanner = new Scanner(System.in);
51     String[] input = scanner.nextLine().split(" ");
52     scanner.close();
53
54     String postOrder = input[0];
55     String inOrder = input[1];
56
57     Node root = generateNode(postOrder, inOrder);
58     System.out.println(BFS(root));
59 }
60 }
```

Python

```
1 import sys
2 from collections import deque
3
4 class Node:
5     def __init__(self, value):
6         self.value = value
7         self.left = None
8         self.right = None
9
10 # 根据后序遍历和中序遍历构造二叉树
11 def generate_node(post_order, in_order):
12     if not post_order:
13         return None
14
15     root = Node(post_order[-1])
16     j = in_order.index(post_order[-1]) # 找到根节点在中序遍历的位置
17
18     root.left = generate_node(post_order[:j], in_order[:j])
19     root.right = generate_node(post_order[j:-1], in_order[j+1:])
20
21     return root
22
23 # 层序遍历 BFS
24 def bfs(root):
25     if not root:
26         return ""
27
28     res = []
29     queue = deque([root])
30
31     while queue:
32         tmp = queue.popleft()
33         res.append(tmp.value)
34
35         if tmp.left:
36             queue.append(tmp.left)
37         if tmp.right:
38             queue.append(tmp.right)
39
40     return "".join(res)
41
42 if __name__ == "__main__":
43     # 读取输入
44     post_order, in_order = sys.stdin.readline().strip().split()
```

```
45  
46     # 生成二叉树  
47     root = generate_node(post_order, in_order)  
48  
49     # 输出层序遍历结果  
50     print(hfs(root))
```

JavaScript

```
1 const readline = require("readline");
2
3 class Node {
4     constructor(value) {
5         this.value = value;
6         this.left = null;
7         this.right = null;
8     }
9 }
10
11 // 根据后序遍历和中序遍历构造二叉树
12 function generateNode(postOrder, inOrder) {
13     if (!postOrder.length) return null;
14
15     let root = new Node(postOrder[postOrder.length - 1]);
16     let j = inOrder.indexOf(postOrder[postOrder.length - 1]); // 找到根节点
17
18     root.left = generateNode(postOrder.substring(0, j), inOrder.substring(0, j));
19     root.right = generateNode(postOrder.substring(j, postOrder.length - 1), inOrder.substring(j + 1));
20
21     return root;
22 }
23
24 // 层序遍历 BFS
25 function bfs(root) {
26     if (!root) return "";
27
28     let res = "";
29     let queue = [root];
30
31     while (queue.length > 0) {
32         let tmp = queue.shift();
33         res += tmp.value;
34
35         if (tmp.left) queue.push(tmp.left);
36         if (tmp.right) queue.push(tmp.right);
37     }
38
39     return res;
40 }
41
42 // 读取输入并执行
```

```
43 const rl = readline.createInterface({  
44     input: process.stdin,  
45     output: process.stdout  
46 });  
47  
48 rl.on("line", (input) => {  
49     let [postOrder, inOrder] = input.trim().split(" ");  
50     let root = generateNode(postOrder, inOrder);  
51     console.log(bfs(root));  
52     rl.close();  
53 }).  
54 .
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 type Node struct {
11     left, right *Node
12     value        byte
13 }
14
15 // 通过后序遍历和中序遍历构造二叉树
16 func generateNode(postOrder, inOrder string) *Node {
17     if len(postOrder) == 0 {
18         return nil
19     }
20
21     n := len(postOrder)
22     root := &Node{value: postOrder[n-1]}
23
24     // 找到根节点在中序遍历中的位置
25     j := strings.IndexByte(inOrder, postOrder[n-1])
26
27     root.left = generateNode(postOrder[:j], inOrder[:j])
28     root.right = generateNode(postOrder[j:n-1], inOrder[j+1:])
29
30     return root
31 }
32
33 // 层序遍历 BFS
34 func bfs(root *Node) string {
35     if root == nil {
36         return ""
37     }
38
39     var res strings.Builder
40     queue := []*Node{root}
41
42     for len(queue) > 0 {
43         tmp := queue[0]
44         queue = queue[1:] // 出队列
```

```
45     res.WriteByte(tmp.value)
46
47     if tmp.left != nil {
48         queue = append(queue, tmp.left)
49     }
50     if tmp.right != nil {
51         queue = append(queue, tmp.right)
52     }
53 }
54
55     return res.String()
56 }
57
58 func main() {
59     scanner := bufio.NewScanner(os.Stdin)
60     scanner.Scan()
61     input := strings.Split(scanner.Text(), " ")
62
63     postOrder, inOrder := input[0], input[1]
64
65     root := generateNode(postOrder, inOrder)
66     fmt.Println(bfs(root))
67 }
```

二叉树的广度优先遍历

华为OD机试真题目录: [点击查看](#)

2025B卷 200分题型

题目描述

有一棵二叉树，每个节点由一个大写字母标识(最多26个节点)。

现有两组字母，分别表示后序遍历（左孩子->右孩子->父节点）和中序遍历（左孩子->父节点->右孩子）的结果，请你输出层序遍历的结果。

输入描述

每个输入文件一行，第一个字符串表示后序遍历结果，第二个字符串表示中序遍历结果。（每串只包含大写字母）

中间用单空格分隔。

输出描述

输出仅一行，表示层序遍历的结果，结尾换行。

用例1

输入

```
Plain Text |  
1 CBEFDA CBAEDF
```

输出

```
Plain Text |  
1 ABDCEF
```

说明

```
Plain Text |  
1 二叉树为：  
2  
3     A  
4     /   \  
5     B     D  
6     /     /   \  
7     C     E     F
```

题解

思路： 二叉树， 中序遍历(左 -> 父 -> 右), 后序遍历(左 -> 右 -> 父亲)

1. 通过后序和中序遍历的序列还原出原二叉树的结构，下面简单说一下还原规律，以示例1为说明。
后序： CBEFDA 中序： CBAEDF
 - 后序遍历规律(左孩子->右孩子->父节点)可以得出最后一个值肯定是父节点的值，此时父节点为A
 - 在中序遍历中找到父节点的值， CB A EDF ,由中序遍历的规律(左孩子->父节点->右孩子)可得A的左子树包含 CB 两个元素(这也是它们中序遍历的输出) 右子树包含 EDF 三个元素(中序遍历输出)
 - 根据2得出左右子树的长度，可以得出左子树的后序遍历输出为 CB 右子树的后序遍历结果为 EDF. 左子树的中序遍历的输出为 **CB**， 右子树的中序遍历的输出为 **EDF**
 - 借助上面提到的三个规律，使用 **递归** 就可以还原出二叉树的结构。整个树的还原其实是 **自下而上的**. 递归终止条件是到达叶子节点。
2. 通过第一步递归还原出二叉树的结构之后，接下来就是对结果进行层序遍历，一层一层进行输出二叉树的值，从上到下，从左到右。一般层序遍历都是采用 **BFS** 去模拟的。下面代码中使用 **栈** 去实现BFS，并将记录BFS遍历的节点值输出顺序，最后拼接成字符串输出即可。

二叉树还是属于比较基础的数据结构，建议多花点时间看看二叉树相关的。额外思考，如果是给出二叉树的前序遍历和中序遍历，你能还原出二叉树的原始结构吗？

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<queue>
8 using namespace std;
9
10 struct Node{
11     Node* left;
12     Node* right;
13     char value;
14     Node(){}
15     Node(char value):value(value), left(nullptr), right(nullptr){}
16 };
17
18 // 通用 split 函数
19 vector<string> split(const string& str, const string& delimiter) {
20     vector<string> result;
21     size_t start = 0;
22     size_t end = str.find(delimiter);
23     while (end != string::npos) {
24         result.push_back(str.substr(start, end - start));
25         start = end + delimiter.length();
26         end = str.find(delimiter, start);
27     }
28     // 添加最后一个部分
29     result.push_back(str.substr(start));
30     return result;
31 }
32
33 // 通过后序和中序还原二叉树
34 Node* generateNode(string postOrder, string inOrder) {
35     if (postOrder.empty()) {
36         return nullptr;
37     }
38     int n = postOrder.size();
39     Node* root = new Node(postOrder[n-1]);
40     // 确定左子树的长度
41     int j = 0;
42     for (; j < n; j++) {
43         if (postOrder[n-1] == inOrder[j]) {
44             break;
```

```

45         }
46     }
47     // 确定左右子树中序遍历结果
48     string inOrderLeft = inOrder.substr(0, j);
49     string inOrderRight = inOrder.substr(j+1);
50     // 确定左右子树后序遍历结果
51     string postOrderLeft = postOrder.substr(0, j);
52     string postOrderRight = postOrder.substr(j, n-j-1);
53     if (!inOrderLeft.empty()) {
54         root->left = generateNode(postOrderLeft, inOrderLeft);
55     }
56     if (!inOrderRight.empty()) {
57         root->right = generateNode(postOrderRight, inOrderRight);
58     }
59     return root;
60 }
61
62 string BFS(Node* root) {
63     string res = "";
64     if (root == nullptr) {
65         return res;
66     }
67     queue<Node*> q;
68     q.push(root);
69     while (!q.empty()) {
70         Node* tmp = q.front();
71         q.pop();
72         res += string(1, tmp->value);
73         if (tmp -> left != nullptr) {
74             q.push(tmp->left);
75         }
76         if (tmp ->right != nullptr) {
77             q.push(tmp->right);
78         }
79     }
80     return res;
81 }
82
83
84
85
86
87 int main() {
88     string s;
89     getline(cin , s);
90     if (s.empty()) {
91         cout << "";

```

```
92         return 0;
93     }
94     vector<string> tmp = split(s, " ");
95     string postOrder = tmp[0];
96     string inorder = tmp[1];
97     Node* root = generateNode(postOrder,inorder);
98     string res = BFS(root);
99     cout << res;
100    return 0;
```

JAVA

```
1 import java.util.*;
2
3 class Main {
4     static class Node {
5         Node left, right;
6         char value;
7         Node(char value) {
8             this.value = value;
9             this.left = null;
10            this.right = null;
11        }
12    }
13
14 // 根据后序遍历和中序遍历构造二叉树
15 static Node generateNode(String postOrder, String inOrder) {
16     if (postOrder.isEmpty()) return null;
17
18     int n = postOrder.length();
19     Node root = new Node(postOrder.charAt(n - 1));
20
21     int j = inOrder.indexOf(postOrder.charAt(n - 1)); // 确定左子树大小
22
23     // 递归构造左右子树
24     root.left = generateNode(postOrder.substring(0, j), inOrder.substring(0, j));
25     root.right = generateNode(postOrder.substring(j, n - 1), inOrder.substring(j + 1));
26
27     return root;
28 }
29
30 // 层序遍历 BFS
31 static String BFS(Node root) {
32     if (root == null) return "";
33
34     StringBuilder res = new StringBuilder();
35     Queue<Node> queue = new LinkedList<>();
36     queue.add(root);
37
38     while (!queue.isEmpty()) {
39         Node tmp = queue.poll();
40         res.append(tmp.value);
41
42         if (tmp.left != null) queue.add(tmp.left);
43     }
44
45     return res.toString();
46 }
```

```
43             if (tmp.right != null) queue.add(tmp.right);
44     }
45
46     return res.toString();
47 }
48
49 public static void main(String[] args) {
50     Scanner scanner = new Scanner(System.in);
51     String[] input = scanner.nextLine().split(" ");
52     scanner.close();
53
54     String postOrder = input[0];
55     String inOrder = input[1];
56
57     Node root = generateNode(postOrder, inOrder);
58     System.out.println(BFS(root));
59 }
60 }
```

Python

```
1 import sys
2 from collections import deque
3
4 class Node:
5     def __init__(self, value):
6         self.value = value
7         self.left = None
8         self.right = None
9
10 # 根据后序遍历和中序遍历构造二叉树
11 def generate_node(post_order, in_order):
12     if not post_order:
13         return None
14
15     root = Node(post_order[-1])
16     j = in_order.index(post_order[-1]) # 找到根节点在中序遍历的位置
17
18     root.left = generate_node(post_order[:j], in_order[:j])
19     root.right = generate_node(post_order[j:-1], in_order[j+1:])
20
21     return root
22
23 # 层序遍历 BFS
24 def bfs(root):
25     if not root:
26         return ""
27
28     res = []
29     queue = deque([root])
30
31     while queue:
32         tmp = queue.popleft()
33         res.append(tmp.value)
34
35         if tmp.left:
36             queue.append(tmp.left)
37         if tmp.right:
38             queue.append(tmp.right)
39
40     return "".join(res)
41
42 if __name__ == "__main__":
43     # 读取输入
44     post_order, in_order = sys.stdin.readline().strip().split()
```

```
45
46      # 生成二叉树
47      root = generate_node(post_order, in_order)
48
49      # 输出层序遍历结果
50      print(hfs(root))
```

JavaScript

```
1 const readline = require("readline");
2
3 class Node {
4     constructor(value) {
5         this.value = value;
6         this.left = null;
7         this.right = null;
8     }
9 }
10
11 // 根据后序遍历和中序遍历构造二叉树
12 function generateNode(postOrder, inOrder) {
13     if (!postOrder.length) return null;
14
15     let root = new Node(postOrder[postOrder.length - 1]);
16     let j = inOrder.indexOf(postOrder[postOrder.length - 1]); // 找到根节点
17
18     root.left = generateNode(postOrder.substring(0, j), inOrder.substring(0, j));
19     root.right = generateNode(postOrder.substring(j, postOrder.length - 1), inOrder.substring(j + 1));
20
21     return root;
22 }
23
24 // 层序遍历 BFS
25 function bfs(root) {
26     if (!root) return "";
27
28     let res = "";
29     let queue = [root];
30
31     while (queue.length > 0) {
32         let tmp = queue.shift();
33         res += tmp.value;
34
35         if (tmp.left) queue.push(tmp.left);
36         if (tmp.right) queue.push(tmp.right);
37     }
38
39     return res;
40 }
41
42 // 读取输入并执行
```

```
43 const rl = readline.createInterface({  
44     input: process.stdin,  
45     output: process.stdout  
46 });  
47  
48 rl.on("line", (input) => {  
49     let [postOrder, inOrder] = input.trim().split(" ");  
50     let root = generateNode(postOrder, inOrder);  
51     console.log(bfs(root));  
52     rl.close();  
53 }).  
54 .
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 type Node struct {
11     left, right *Node
12     value        byte
13 }
14
15 // 通过后序遍历和中序遍历构造二叉树
16 func generateNode(postOrder, inOrder string) *Node {
17     if len(postOrder) == 0 {
18         return nil
19     }
20
21     n := len(postOrder)
22     root := &Node{value: postOrder[n-1]}
23
24     // 找到根节点在中序遍历中的位置
25     j := strings.IndexByte(inOrder, postOrder[n-1])
26
27     root.left = generateNode(postOrder[:j], inOrder[:j])
28     root.right = generateNode(postOrder[j:n-1], inOrder[j+1:])
29
30     return root
31 }
32
33 // 层序遍历 BFS
34 func bfs(root *Node) string {
35     if root == nil {
36         return ""
37     }
38
39     var res strings.Builder
40     queue := []*Node{root}
41
42     for len(queue) > 0 {
43         tmp := queue[0]
44         queue = queue[1:] // 出队列
```

```
45     res.WriteByte(tmp.value)
46
47     if tmp.left != nil {
48         queue = append(queue, tmp.left)
49     }
50     if tmp.right != nil {
51         queue = append(queue, tmp.right)
52     }
53 }
54
55     return res.String()
56 }
57
58 func main() {
59     scanner := bufio.NewScanner(os.Stdin)
60     scanner.Scan()
61     input := strings.Split(scanner.Text(), " ")
62
63     postOrder, inOrder := input[0], input[1]
64
65     root := generateNode(postOrder, inOrder)
66     fmt.Println(bfs(root))
67 }
```

来自: 华为OD机试 – 二叉树的广度优先遍历 (2025 B卷 200分)_华为od机试 二叉树的广度优先遍历–CSDN博客

二叉树的广度优先遍历

华为OD机试真题目录: [点击查看](#)

2025B卷 200分题型

题目描述

有一棵二叉树，每个节点由一个大写字母标识(最多26个节点)。

现有两组字母，分别表示后序遍历（左孩子→右孩子→父节点）和中序遍历（左孩子→父节点→右孩子）的结果，请你输出层序遍历的结果。

输入描述

每个输入文件一行，第一个字符串表示后序遍历结果，第二个字符串表示中序遍历结果。（每串只包含大写字母）

中间用单空格分隔。

输出描述

输出仅一行，表示层序遍历的结果，结尾换行。

用例1

输入

```
Plain Text |  
1 CBEFDA CBAEDF
```

输出

```
Plain Text |  
1 ABDCEF
```

说明

```
Plain Text |  
1 二叉树为：  
2  
3      A  
4      /   \  
5      B     D  
6      /       /   \  
7      C       E     F
```

题解

思路： 二叉树， 中序遍历(左 -> 父 -> 右), 后序遍历(左 -> 右 -> 父亲)

1. 通过后序和中序遍历的序列还原出原二叉树的结构，下面简单说一下还原规律，以示例1为说明。后

序： CBEFDA 中序： CBAEDF

- 后序遍历规律(左孩子->右孩子->父节点)可以得出最后一个值肯定是父节点的值，此时父节点为 A
- 在中序遍历中找到父节点的值， CB A EDF ,由中序遍历的规律(左孩子->父节点->右孩子)可得 A 的左子树包含 CB 两个元素(这也是它们中序遍历的输出) 右子树包含 EDF 三个元素(中序遍历输出)
- 根据2得出左右子树的长度，可以得出左子树的后序遍历输出为 CB 右子树的后序遍历结果为 EFD. 左子树的中序遍历的输出为 CB ，右子树的中序遍历的输出为 EDF

- 借助上面提到的三个规律，使用 递归 就可以还原出二叉树的结构。整个树的还原其实是 自下而上的。递归终止条件是到达叶子节点。
2. 通过第一步递归还原出二叉树的结构之后，接下来就是对结果进行层序遍历，一层一层进行输出二叉树的值，从上到下，从左到右。一般层序遍历都是采用 BFS 去模拟的。下面代码中使用 栈 去实现 BFS，并将记录BFS遍历的节点值输出顺序，最后拼接成字符串输出即可。

二叉树还是属于比较基础的数据结构，建议多花点时间看看二叉树相关的。额外思考，如果是给出二叉树的前序遍历和中序遍历，你能还原出二叉树的原始结构吗？

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<queue>
8 using namespace std;
9
10 struct Node{
11     Node* left;
12     Node* right;
13     char value;
14     Node(){}
15     Node(char value):value(value), left(nullptr), right(nullptr){}
16 };
17
18 // 通用 split 函数
19 vector<string> split(const string& str, const string& delimiter) {
20     vector<string> result;
21     size_t start = 0;
22     size_t end = str.find(delimiter);
23     while (end != string::npos) {
24         result.push_back(str.substr(start, end - start));
25         start = end + delimiter.length();
26         end = str.find(delimiter, start);
27     }
28     // 添加最后一个部分
29     result.push_back(str.substr(start));
30     return result;
31 }
32
33 // 通过后序和中序还原二叉树
34 Node* generateNode(string postOrder, string inOrder) {
35     if (postOrder.empty()) {
36         return nullptr;
37     }
38     int n = postOrder.size();
39     Node* root = new Node(postOrder[n-1]);
40     // 确定左子树的长度
41     int j = 0;
42     for (; j < n; j++) {
43         if (postOrder[n-1] == inOrder[j]) {
44             break;
```

```

45         }
46     }
47     // 确定左右子树中序遍历结果
48     string inOrderLeft = inOrder.substr(0, j);
49     string inOrderRight = inOrder.substr(j+1);
50     // 确定左右子树后序遍历结果
51     string postOrderLeft = postOrder.substr(0, j);
52     string postOrderRight = postOrder.substr(j, n-j-1);
53     if (!inOrderLeft.empty()) {
54         root->left = generateNode(postOrderLeft, inOrderLeft);
55     }
56     if (!inOrderRight.empty()) {
57         root->right = generateNode(postOrderRight, inOrderRight);
58     }
59     return root;
60 }
61
62 string BFS(Node* root) {
63     string res = "";
64     if (root == nullptr) {
65         return res;
66     }
67     queue<Node*> q;
68     q.push(root);
69     while (!q.empty()) {
70         Node* tmp = q.front();
71         q.pop();
72         res += string(1, tmp->value);
73         if (tmp -> left != nullptr) {
74             q.push(tmp->left);
75         }
76         if (tmp ->right != nullptr) {
77             q.push(tmp->right);
78         }
79     }
80     return res;
81 }
82
83
84
85
86
87 int main() {
88     string s;
89     getline(cin , s);
90     if (s.empty()) {
91         cout << "";

```

```
92         return 0;
93     }
94     vector<string> tmp = split(s, " ");
95     string postOrder = tmp[0];
96     string inorder = tmp[1];
97     Node* root = generateNode(postOrder,inorder);
98     string res = BFS(root);
99     cout << res;
100    return 0;
```

JAVA

```
1 import java.util.*;
2
3 class Main {
4     static class Node {
5         Node left, right;
6         char value;
7         Node(char value) {
8             this.value = value;
9             this.left = null;
10            this.right = null;
11        }
12    }
13
14 // 根据后序遍历和中序遍历构造二叉树
15 static Node generateNode(String postOrder, String inOrder) {
16     if (postOrder.isEmpty()) return null;
17
18     int n = postOrder.length();
19     Node root = new Node(postOrder.charAt(n - 1));
20
21     int j = inOrder.indexOf(postOrder.charAt(n - 1)); // 确定左子树大小
22
23     // 递归构造左右子树
24     root.left = generateNode(postOrder.substring(0, j), inOrder.substring(0, j));
25     root.right = generateNode(postOrder.substring(j, n - 1), inOrder.substring(j + 1));
26
27     return root;
28 }
29
30 // 层序遍历 BFS
31 static String BFS(Node root) {
32     if (root == null) return "";
33
34     StringBuilder res = new StringBuilder();
35     Queue<Node> queue = new LinkedList<>();
36     queue.add(root);
37
38     while (!queue.isEmpty()) {
39         Node tmp = queue.poll();
40         res.append(tmp.value);
41
42         if (tmp.left != null) queue.add(tmp.left);
43     }
44
45     return res.toString();
46 }
```

```
43             if (tmp.right != null) queue.add(tmp.right);
44     }
45
46     return res.toString();
47 }
48
49 public static void main(String[] args) {
50     Scanner scanner = new Scanner(System.in);
51     String[] input = scanner.nextLine().split(" ");
52     scanner.close();
53
54     String postOrder = input[0];
55     String inOrder = input[1];
56
57     Node root = generateNode(postOrder, inOrder);
58     System.out.println(BFS(root));
59 }
60 }
```

Python

```
1 import sys
2 from collections import deque
3
4 class Node:
5     def __init__(self, value):
6         self.value = value
7         self.left = None
8         self.right = None
9
10 # 根据后序遍历和中序遍历构造二叉树
11 def generate_node(post_order, in_order):
12     if not post_order:
13         return None
14
15     root = Node(post_order[-1])
16     j = in_order.index(post_order[-1]) # 找到根节点在中序遍历的位置
17
18     root.left = generate_node(post_order[:j], in_order[:j])
19     root.right = generate_node(post_order[j:-1], in_order[j+1:])
20
21     return root
22
23 # 层序遍历 BFS
24 def bfs(root):
25     if not root:
26         return ""
27
28     res = []
29     queue = deque([root])
30
31     while queue:
32         tmp = queue.popleft()
33         res.append(tmp.value)
34
35         if tmp.left:
36             queue.append(tmp.left)
37         if tmp.right:
38             queue.append(tmp.right)
39
40     return "".join(res)
41
42 if __name__ == "__main__":
43     # 读取输入
44     post_order, in_order = sys.stdin.readline().strip().split()
```

```
45
46      # 生成二叉树
47      root = generate_node(post_order, in_order)
48
49      # 输出层序遍历结果
50      print(hfs(root))
```

JavaScript

```
1 const readline = require("readline");
2
3 class Node {
4     constructor(value) {
5         this.value = value;
6         this.left = null;
7         this.right = null;
8     }
9 }
10
11 // 根据后序遍历和中序遍历构造二叉树
12 function generateNode(postOrder, inOrder) {
13     if (!postOrder.length) return null;
14
15     let root = new Node(postOrder[postOrder.length - 1]);
16     let j = inOrder.indexOf(postOrder[postOrder.length - 1]); // 找到根节点
17
18     root.left = generateNode(postOrder.substring(0, j), inOrder.substring(0, j));
19     root.right = generateNode(postOrder.substring(j, postOrder.length - 1), inOrder.substring(j + 1));
20
21     return root;
22 }
23
24 // 层序遍历 BFS
25 function bfs(root) {
26     if (!root) return "";
27
28     let res = "";
29     let queue = [root];
30
31     while (queue.length > 0) {
32         let tmp = queue.shift();
33         res += tmp.value;
34
35         if (tmp.left) queue.push(tmp.left);
36         if (tmp.right) queue.push(tmp.right);
37     }
38
39     return res;
40 }
41
42 // 读取输入并执行
```

```
43 const rl = readline.createInterface({  
44     input: process.stdin,  
45     output: process.stdout  
46 });  
47  
48 rl.on("line", (input) => {  
49     let [postOrder, inOrder] = input.trim().split(" ");  
50     let root = generateNode(postOrder, inOrder);  
51     console.log(bfs(root));  
52     rl.close();  
53 }).  
54 .
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 type Node struct {
11     left, right *Node
12     value        byte
13 }
14
15 // 通过后序遍历和中序遍历构造二叉树
16 func generateNode(postOrder, inOrder string) *Node {
17     if len(postOrder) == 0 {
18         return nil
19     }
20
21     n := len(postOrder)
22     root := &Node{value: postOrder[n-1]}
23
24     // 找到根节点在中序遍历中的位置
25     j := strings.IndexByte(inOrder, postOrder[n-1])
26
27     root.left = generateNode(postOrder[:j], inOrder[:j])
28     root.right = generateNode(postOrder[j:n-1], inOrder[j+1:])
29
30     return root
31 }
32
33 // 层序遍历 BFS
34 func bfs(root *Node) string {
35     if root == nil {
36         return ""
37     }
38
39     var res strings.Builder
40     queue := []*Node{root}
41
42     for len(queue) > 0 {
43         tmp := queue[0]
44         queue = queue[1:] // 出队列
```

```
45     res.WriteByte(tmp.value)
46
47     if tmp.left != nil {
48         queue = append(queue, tmp.left)
49     }
50     if tmp.right != nil {
51         queue = append(queue, tmp.right)
52     }
53 }
54
55     return res.String()
56 }
57
58 func main() {
59     scanner := bufio.NewScanner(os.Stdin)
60     scanner.Scan()
61     input := strings.Split(scanner.Text(), " ")
62
63     postOrder, inOrder := input[0], input[1]
64
65     root := generateNode(postOrder, inOrder)
66     fmt.Println(bfs(root))
67 }
```

来自: 华为OD机试 – 二叉树的广度优先遍历 (2025 B卷 200分)_华为od机试 二叉树的广度优先遍历–CSDN博客

来自: 华为OD机试 – 二叉树的广度优先遍历 (2025 B卷 200分)_华为od机试 二叉树的广度优先遍历–CSDN博客

华为OD机试 - 池化资源共享(2025 B卷 100分)- CSDN博客

池化资源共享

华为OD机试真题目录: [点击查看](#)

2025B卷 100分题型

题目描述

有一个局部互联区域内的 n 台设备，每台设备都有一定数量的空闲资源，这些资源可以池化共享。用户会发起两种操作：

1. 申请资源：输入 `1 x`，表示本次申请需要 x 个资源。系统要返回当前资源池中能满足此申请且剩余资源最少的设备 ID；如果有多台设备满足条件，返回设备 ID 最小的；如果没有任何设备能满足，返回 `0` 并不做任何分配。
2. 释放资源：输入 `2 y`，表示将第 y 次申请（不一定是成功分配的那一次）释放回原设备。释放时，资源立即归还，且空闲资源自动连续，无需考虑空洞。

给定 n ($1 \leq n \leq 1000$)、操作次数 m ($1 \leq m \leq 105$)，以及初始时每台设备的空闲资源数 $d_1 \dots d_n$ ($1 \leq d_i \leq 1000$)，以及接下来 m 行操作，输出每次“申请”操作的返回值。

输入描述

第一行，输入 n 和 m

第二行输入 n 台设备的初始空闲资源数。

接下来 m 行，输出要执行的操作。

输出描述

输出每次“申请”操作的返回值。

用例1

输入

```
Plain Text |  
  
1 2 2  
2 100 500  
3 1 40  
4 1 450
```

输出

```
Plain Text |  
1 1 2
```

题解

思路： 模拟

1. 接收输入的每台设备的空闲资源数。
2. 使用数组存储每次分配选择的 `id 分配内存大小`，用于后续进行释放资源。这里简单说说申请资源过程，
 - a. 循环找出当前设备中能够满足申请资源前提下，分配之后剩余资源最少，且id最小的设备id。
 - 如果不能找到满足分配的情况，本次分配方案设置为 `0 0`
 - 如果能够找到满足分配的情况，本次分配设置为`选择设备id 申请资源数`。并更新被选择设备的申请资源剩余的资源数
 - b. 存储本次分配方案。
3. 释放资源逻辑比较简单，不多赘述，不懂得可以看看代码。
4. 输出每次申请资源的选择的设备id即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<climits>
8 using namespace std;
9
10 int main() {
11     int n,m;
12     cin >> n >> m;
13
14     vector<int> free(n+1, 0);
15     for (int i = 1; i <= n; i++) {
16         cin >> free[i];
17     }
18     // 记录每一次分配的id 和 大小
19     vector<pair<int, int>> alloc;
20
21     vector<int> result;
22
23     while (m--) {
24         int type, x;
25         cin >> type >> x;
26         // 分配
27         if (type == 1) {
28             int bestID = 0;
29             int bestRemoveFree = INT_MAX;
30             for (int i = 1; i <= n; i++) {
31                 // 无法满足分配
32                 if (free[i] < x) {
33                     continue;
34                 }
35                 int rem = free[i] - x;
36                 if (bestID == 0|| bestRemoveFree > rem) {
37                     bestID = i;
38                     bestRemoveFree = rem;
39                 }
40             }
41             // 分配失败
42             if (bestID == 0) {
43                 alloc.push_back({0, 0});
44             } else {
```

```
45         free[bestID] -= x;
46         alloc.push_back({bestID, x});
47     }
48     result.push_back(bestID);
49 } else { // 释放
50     // 输入数据有问题
51     if (x > alloc.size()) {
52         continue;
53     }
54     pair<int, int> p = alloc[x - 1];
55     // 没有分配成功
56     if (p.first == 0) {
57         continue;
58     }
59     free[p.first] += p.second;
60 }
61 }
62
63 // 输出结果
64 for (int i = 0; i < result.size(); i++) {
65     cout << result[i];
66     if (i != result.size() - 1) {
67         cout << " ";
68     }
69 }
70 return 0;
71 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int n = sc.nextInt(), m = sc.nextInt();
7
8         // 存储每个节点的空闲内存
9         int[] free = new int[n + 1];
10        for (int i = 1; i <= n; i++) {
11            free[i] = sc.nextInt();
12        }
13
14        // 记录每一次分配的id 和 大小
15        List<int[]> alloc = new ArrayList<>();
16        List<Integer> result = new ArrayList<>();
17
18        while (m-- > 0) {
19            int type = sc.nextInt();
20            int x = sc.nextInt();
21
22            if (type == 1) {
23                // 分配内存
24                int bestID = 0;
25                int bestRemoveFree = Integer.MAX_VALUE;
26                for (int i = 1; i <= n; i++) {
27                    // 无法满足分配
28                    if (free[i] < x) continue;
29                    int rem = free[i] - x;
30                    if (bestID == 0 || bestRemoveFree > rem) {
31                        bestID = i;
32                        bestRemoveFree = rem;
33                    }
34                }
35                // 分配失败
36                if (bestID == 0) {
37                    alloc.add(new int[]{0, 0});
38                } else {
39                    free[bestID] -= x;
40                    alloc.add(new int[]{bestID, x});
41                }
42                result.add(bestID);
43            } else {
44                // 释放内存
45            }
46        }
47    }
48}
```

```
45         if (x > alloc.size()) continue;
46         int[] p = alloc.get(x - 1);
47         // 没有分配成功
48         if (p[0] == 0) continue;
49         free[p[0]] += p[1];
50     }
51 }
52
53 // 输出结果
54 for (int i = 0; i < result.size(); i++) {
55     System.out.print(result.get(i));
56     if (i != result.size() - 1) {
57         System.out.print(" ");
58     }
59 }
60 }
61 }
```

Python

```
1 import sys
2 input = sys.stdin.readline
3
4 n, m = map(int, input().split())
5
6 # 存储每个节点的空闲内存
7 free = [0] + list(map(int, input().split()))
8
9 # 记录每一次分配的id 和 大小
10 alloc = []
11 result = []
12
13 for _ in range(m):
14     line = input()
15     if not line.strip():
16         continue
17     type_, x = map(int, line.strip().split())
18
19     if type_ == 1:
20         # 分配内存
21         bestID = 0
22         bestRemoveFree = float('inf')
23         for i in range(1, n + 1):
24             # 无法满足分配
25             if free[i] < x:
26                 continue
27             rem = free[i] - x
28             if bestID == 0 or bestRemoveFree > rem:
29                 bestID = i
30                 bestRemoveFree = rem
31         # 分配失败
32         if bestID == 0:
33             alloc.append((0, 0))
34         else:
35             free[bestID] -= x
36             alloc.append((bestID, x))
37             result.append(bestID)
38     else:
39         # 释放内存
40         if x > len(alloc):
41             continue
42         p = alloc[x - 1]
43         # 没有分配成功
44         if p[0] == 0:
```

```
45         continue
46         free[p[0]] += p[1]
47
48     # 输出结果
49     print(" ".join(map(str, result)))
```

JavaScript

```
1 const readline = require('readline');
2 const rl = readline.createInterface({
3     input: process.stdin,
4     output: process.stdout
5 });
6
7 let lines = [];
8 rl.on('line', line => {
9     lines.push(line);
10 });
11
12 rl.on('close', () => {
13     const [n, m] = lines[0].split(' ').map(Number);
14     const free = [0, ...lines[1].split(' ').map(Number)];
15
16     // 记录每一次分配的id 和 大小
17     const alloc = [];
18     const result = [];
19
20     for (let i = 0; i < m; i++) {
21         const [type, x] = lines[2 + i].split(' ').map(Number);
22
23         if (type === 1) {
24             // 分配内存
25             let bestID = 0;
26             let bestRemoveFree = Infinity;
27
28             for (let j = 1; j <= n; j++) {
29                 // 无法满足分配
30                 if (free[j] < x) continue;
31                 let rem = free[j] - x;
32                 if (bestID === 0 || bestRemoveFree > rem) {
33                     bestID = j;
34                     bestRemoveFree = rem;
35                 }
36             }
37
38             // 分配失败
39             if (bestID === 0) {
40                 alloc.push([0, 0]);
41             } else {
42                 free[bestID] -= x;
43                 alloc.push([bestID, x]);
44             }
45         }
46     }
47
48     console.log(result);
49 }
50 );
```

```
45         result.push(bestID);
46     } else {
47         // 释放内存
48         if (x > alloc.length) continue;
49         const p = alloc[x - 1];
50         // 没有分配成功
51         if (p[0] === 0) continue;
52         free[p[0]] += p[1];
53     }
54 }
55
56 // 输出结果
57 console.log(result.join(' '));
58 }):
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13
14     line1, _ := reader.ReadString('\n')
15     nm := strings.Fields(line1)
16     n, _ := strconv.Atoi(nm[0])
17     m, _ := strconv.Atoi(nm[1])
18
19     line2, _ := reader.ReadString('\n')
20     nums := strings.Fields(line2)
21     // 存储每个节点的空闲内存
22     free := make([]int, n+1)
23     for i := 1; i <= n; i++ {
24         free[i], _ = strconv.Atoi(nums[i-1])
25     }
26
27     // 记录每一次分配的id 和 大小
28     var alloc [][]int
29     var result []int
30
31     for i := 0; i < m; i++ {
32         line, _ := reader.ReadString('\n')
33         parts := strings.Fields(line)
34         if len(parts) < 2 {
35             continue
36         }
37         typ, _ := strconv.Atoi(parts[0])
38         x, _ := strconv.Atoi(parts[1])
39
40         if typ == 1 {
41             // 分配内存
42             bestID := 0
43             bestRemoveFree := 1 << 30
44             for j := 1; j <= n; j++ {
```

```
45     // 无法满足分配
46     if free[j] < x {
47         continue
48     }
49     rem := free[j] - x
50     if bestID == 0 || bestRemoveFree > rem {
51         bestID = j
52         bestRemoveFree = rem
53     }
54 }
55 // 分配失败
56 if bestID == 0 {
57     alloc = append(alloc, [2]int{0, 0})
58 } else {
59     free[bestID] -= x
60     alloc = append(alloc, [2]int{bestID, x})
61 }
62 result = append(result, bestID)
63 } else {
64     // 释放内存
65     if x > len(alloc) {
66         continue
67     }
68     p := alloc[x-1]
69     // 没有分配成功
70     if p[0] == 0 {
71         continue
72     }
73     free[p[0]] += p[1]
74 }
75 }
76
77 // 输出结果
78 for i, v := range result {
79     if i > 0 {
80         fmt.Print(" ")
81     }
82     fmt.Print(v)
83 }
84 }
```

来自: 华为OD 机试 – 池化资源共享 (2025 B卷 100分)-CSDN博客

华为OD 2025 B卷 - 选修课 (机试 100分)-CSDN 博客

选修课

华为OD机试真题目录: [点击查看](#)

2025B卷 100分题型

题目描述

现有两门选修课，每门选修课都有一部分学生选修，每个学生都有选修课的成绩，需要你找出同时选修了两门选修课的学生，先按照班级进行划分，班级编号小的先输出，每个班级按照两门选修课成绩和的降序排序，成绩相同时按照学生的学号升序排序。

输入描述

第一行为第一门选修课学生的成绩，

第二行为第二门选修课学生的成绩，每行数据中学生之间以英文分号分隔，每个学生的学号和成绩以英文逗号分隔，

学生学号的格式为8位数字(2位院系编号+入学年份后2位+院系内部1位专业编号+所在班级3位学号)，

学生成绩的取值范围为[0,100]之间的整数，

两门选修课选修学生数的取值范围为[1-2000]之间的整数。

输出描述

同时选修了两门选修课的学生的学号，如果没有同时选修两门选修课的学生输出NULL，

否则，先按照班级划分，班级编号小的先输出，每个班级先输出班级编号(学号前五位)，然后另起一行输出这个班级同时选修两门选修课的学生学号，学号按照要求排序(按照两门选修课成绩和的降序，成绩和相同时按照学号升序)，学生之间以英文分号分隔。

示例1

输入

```
1 01202021,75;01201033,95;01202008,80;01203006,90;01203088,100
2 01202008,70;01203088,85;01202111,80;01202021,75;01201100,88
```

输出

Plain Text

```
1 01202
2 01202008;01202021
3 01203
4 01203088
```

说明

同时选修了两门选修课的学生01202021、01202008、01203088，这三个学生两门选修课的成绩和分别为150、150、185，01202021、01202008属于01202班的学生，按照成绩和降序，成绩相同时按学号升序输出的结果为01202008;01202021,01203088属于01203班的学生，按照成绩和降序，成绩相同时按学号升序输出的结果为01203088，01202的班级编号小于01203的班级编号，需要先输出。

示例2

输入

Plain Text

```
1 01201022,75;01202033,95;01202018,80;01203006,90;01202066,100
2 01202008,70;01203102,85;01202111,80;01201021,75;01201100,88
```

输出

Plain Text

```
1 NULL
```

说明

没有同时选修了两门选修课的学生，输出NULL。

题解

思路： 模拟

- 接收选修第一堂课的学生信息，使用 哈希表<学号， 分数> 或者类似结构存储选修第一门的学生信息。
- 接收选修第二堂课的学生信息时，处理选修当前课的是否选修了第一门，如果选修了，存储至另一个 哈希表 键为 班级 值为 属于这个班级的学生信息。 分类存储
- 处理完第二步之后，首先进行班级内同学的排序，按照 每个班级按照两门选修课成绩和的降序排序，成绩相同时按照学生的学号升序排序 进行排序。 多条件排序
- 处理完第三步之后，处理班与班之间排序，按照 班级编号编号进行升序排序。

5. 按顺序输出每一个班级的每一个学生id。

C++

```
1 #include <iostream>
2 #include <unordered_map>
3 #include <map>
4 #include <set>
5 #include <sstream>
6 #include <string>
7 #include <vector>
8
9 using namespace std;
10
11 struct Student {
12     string id;
13     int score;
14     Student(const string& id, int score) : id(id), score(score) {}
15
16     // 自定义排序规则 (按成绩降序, id 升序)
17     bool operator<(const Student& other) const {
18         return score != other.score ? score > other.score : id < other.id;
19     }
20 };
21
22 // 解析输入字符串, 返回分割后的数据
23 vector<vector<string>> parseStudents(const string& line, char delim1, char delim2) {
24     vector<vector<string>> result;
25     stringstream ss(line), ss2;
26     string token, subtoken;
27
28     while (getline(ss, token, delim1)) {
29         vector<string> student;
30         ss2.clear();
31         ss2.str(token);
32         while (getline(ss2, subtoken, delim2)) {
33             student.push_back(subtoken);
34         }
35         result.push_back(student);
36     }
37     return result;
38 }
39
40 // 检查学生是否在教师列表中, 计算总成绩
41 bool isValidStudent(const unordered_map<string, int>& tIds, const vector<string>& sStu, int& totalScore) {
42     if (tIds.count(sStu[0])) {
```

```
43         totalScore = stoi(sStu[1]) + tIds.at(sStu[0]);
44     return true;
45 }
46 return false;
47 }

48

49 int main() {
50     string line_one, line_two;
51     getline(cin, line_one);
52     getline(cin, line_two);

53     // 解析学生ID 和分数映射表
54     unordered_map<string, int> tIds;
55     for (const auto& tStu : parseStudents(line_two, ',', ',')) {
56         tIds[tStu[0]] = stoi(tStu[1]);
57     }

58

59     // 存储班级学生列表 (按班级 key 自动排序)
60     map<string, set<Student>> classMap;

61

62     for (const auto& sStu : parseStudents(line_one, ',', ',')) {
63         int totalScore;
64         if (isValidStudent(tIds, sStu, totalScore)) {
65             string cls = sStu[0].substr(0, 5); // 提取班级 ID
66             classMap[cls].emplace(sStu[0], totalScore);
67         }
68     }
69 }

70     // 输出结果
71     if (classMap.empty()) {
72         cout << "NULL\n";
73     } else {
74         for (const auto& [cls, students] : classMap) {
75             cout << cls << "\n";
76             string res;
77             for (const auto& student : students) {
78                 res += student.id + ";";
79             }
80             res.pop_back(); // 移除最后的 ";"
81             cout << res << "\n";
82         }
83     }
84 }

85

86     return 0;
87 }
```

Java

```
1 import java.util.*;
2
3 class Student implements Comparable<Student> {
4     String id;
5     int score;
6
7     public Student(String id, int score) {
8         this.id = id;
9         this.score = score;
10    }
11
12    @Override
13    public int compareTo(Student other) {
14        return this.score != other.score ? Integer.compare(other.score, th
is.score) : this.id.compareTo(other.id);
15    }
16 }
17
18 public class Main {
19     // 解析输入字符串，返回分割后的数据
20     private static List<String[]> parseStudents(String line, String delim
1, String delim2) {
21         List<String[]> result = new ArrayList<>();
22         for (String token : line.split(delim1)) {
23             result.add(token.split(delim2));
24         }
25         return result;
26     }
27
28     public static void main(String[] args) {
29         Scanner sc = new Scanner(System.in);
30         String lineOne = sc.nextLine();
31         String lineTwo = sc.nextLine();
32         sc.close();
33
34         // 解析学生ID 和分数映射表
35         Map<String, Integer> tIds = new HashMap<>();
36         for (String[] tStu : parseStudents(lineTwo, ";", ",")) {
37             tIds.put(tStu[0], Integer.parseInt(tStu[1]));
38         }
39
40         // 存储班级学生列表 (按班级 key 自动排序)
41         TreeMap<String, TreeSet<Student>> classMap = new TreeMap<>();
42 }
```

```
43     for (String[] sStu : parseStudents(lineOne, ";", ",")) {
44         String id = sStu[0];
45         if (tIds.containsKey(id)) {
46             int totalScore = Integer.parseInt(sStu[1]) + tIds.get(id);
47             String cls = id.substring(0, 5);
48             classMap.putIfAbsent(cls, new TreeSet<>());
49             classMap.get(cls).add(new Student(id, totalScore));
50         }
51     }
52
53     // 输出结果
54     if (classMap.isEmpty()) {
55         System.out.println("NULL");
56     } else {
57         for (Map.Entry<String, TreeSet<Student>> entry : classMap.entrySet()) {
58             System.out.println(entry.getKey());
59             System.out.println(String.join(";", entry.getValue().stream()
60                                         .map(s -> s.id).toArray(String[]::new)));
61         }
62     }
63 }
```

Python

```
1 import sys
2 from collections import defaultdict
3
4 class Student:
5     """学生类, 定义排序规则 (按成绩降序, ID 升序) """
6     def __init__(self, student_id, score):
7         self.student_id = student_id
8         self.score = score
9
10    def __lt__(self, other):
11        """定义排序规则: 成绩高的排前面, 成绩相同按ID字典序排"""
12        return (self.score, other.student_id) > (other.score, self.student_id)
13
14
15    def parse_students(line, delim1, delim2):
16        """解析输入字符串, 返回分割后的数据"""
17        students = []
18        for token in line.strip().split(delim1):
19            students.append(token.split(delim2))
20        return students
21
22
23    def main():
24        # 读取输入
25        line_one = sys.stdin.readline().strip()
26        line_two = sys.stdin.readline().strip()
27
28        # 解析教师评分列表
29        teacher_scores = {t[0]: int(t[1]) for t in parse_students(line_two,
30            ';;', ',')}
31
32        # 存储班级学生 (使用字典, 值是有序集合)
33        class_map = defaultdict(set)
34
35        for student in parse_students(line_one, ';;', ','):
36            student_id, score = student[0], int(student[1])
37            if student_id in teacher_scores:
38                total_score = score + teacher_scores[student_id]
39                class_id = student_id[:5] # 班级 ID 取前 5 位
40                class_map[class_id].add(Student(student_id, total_score))
41
42        # 输出结果
43        if not class_map:
```

```
43     print("NULL")
44 else:
45     for class_id in sorted(class_map.keys()):
46         print(class_id)
47         sorted_students = sorted(class_map[class_id]) # 按成绩和 ID 排
    序
48         print(";" .join(student.student_id for student in sorted_studen
ts))
49
50
51 if __name__ == "__main__":
52     main()
```

JavaScript

```

1 const fs = require("fs");
2
3 // 解析输入字符串，返回分割后的数据
4 function parseStudents(line, delim1, delim2) {
5     return line.trim().split(delim1).map(token => token.split(delim2));
6 }
7
8 // 读取输入
9 const input = fs.readFileSync(0, "utf-8").trim().split("\n");
10 const lineOne = input[0];
11 const lineTwo = input[1];
12
13 // 解析教师 ID 和分数映射表
14 const tIds = Object.fromEntries(parseStudents(lineTwo, ";", ",").map(([i
d, score]) => [id, parseInt(score)]));
15
16 // 存储班级学生列表（按班级 key 自动排序）
17 const classMap = new Map();
18
19 parseStudents(lineOne, ";", ",").forEach(([sId, sScore]) => {
20     if (tIds.hasOwnProperty(sId)) {
21         const totalScore = parseInt(sScore) + tIds[sId];
22         const cls = sId.slice(0, 5);
23         if (!classMap.has(cls)) classMap.set(cls, new Set());
24         classMap.get(cls).add({ id: sId, score: totalScore });
25     }
26 });
27
28 // 输出结果
29 if (classMap.size === 0) {
30     console.log("NULL");
31 } else {
32     [...classMap.keys()].sort().forEach(cls => {
33         console.log(cls);
34         console.log([...classMap.get(cls)].sort((a, b) => b.score - a.scor
e || a.id.localeCompare(b.id)).map(s => s.id).join(";"));
35     });
36 }

```

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strings"
9     "strconv"
10    )
11
12 // 解析输入字符串，返回分割后的数据
13 func parseStudents(line, delim1, delim2 string) [][]string {
14     tokens := strings.Split(strings.TrimSpace(line), delim1)
15     result := make([][]string, len(tokens))
16     for i, token := range tokens {
17         result[i] = strings.Split(token, delim2)
18     }
19     return result
20 }
21
22 // Student 结构体
23 type Student struct {
24     id    string
25     score int
26 }
27
28 // 自定义排序
29 type ByScore []Student
30
31 func (s ByScore) Len() int          { return len(s) }
32 func (s ByScore) Swap(i, j int)     { s[i], s[j] = s[j], s[i] }
33 func (s ByScore) Less(i, j int) bool {
34     if s[i].score != s[j].score {
35         return s[i].score > s[j].score
36     }
37     return s[i].id < s[j].id
38 }
39
40 func main() {
41     scanner := bufio.NewScanner(os.Stdin)
42
43     scanner.Scan()
44     lineOne := scanner.Text()
```

```

45     scanner.Scan()
46     lineTwo := scanner.Text()
47
48     // 解析教师 ID 和分数映射表
49     tIds := make(map[string]int)
50     for _, tStu := range parseStudents(lineTwo, ";", ",") {
51         tIds[tStu[0]], _ = strconv.Atoi(tStu[1])
52     }
53
54     // 存储班级学生列表
55     classMap := make(map[string][]Student)
56
57     for _, sStu := range parseStudents(lineOne, ";", ",") {
58         id := sStu[0]
59         if score, ok := tIds[id]; ok {
60             totalScore, _ := strconv.Atoi(sStu[1])
61             cls := id[:5]
62             classMap[cls] = append(classMap[cls], Student{id, totalScore + score})
63         }
64     }
65
66     if len(classMap) == 0 {
67         fmt.Println("NULL")
68         return
69     }
70
71     // 输出排序结果
72     keys := make([]string, 0, len(classMap))
73     for k := range classMap {
74         keys = append(keys, k)
75     }
76     sort.Strings(keys)
77
78     for _, cls := range keys {
79         fmt.Println(cls)
80         sort.Sort(ByScore(classMap[cls]))
81         var ids []string
82         for _, s := range classMap[cls] {
83             ids = append(ids, s.id)
84         }
85         fmt.Println(strings.Join(ids, ";"))
86     }
87 }
```

选修课

题目描述

现有两门选修课，每门选修课都有一部分学生选修，每个学生都有选修课的成绩，需要你找出同时选修了两门选修课的学生，先按照班级进行划分，班级编号小的先输出，每个班级按照两门选修课成绩和的降序排序，成绩相同时按照学生的学号升序排序。

输入描述

第一行为第一门选修课学生的成绩，

第二行为第二门选修课学生的成绩，每行数据中学生之间以英文分号分隔，每个学生的学号和成绩以英文逗号分隔，

学生学号的格式为8位数字(2位院系编号+入学年份后2位+院系内部1位专业编号+所在班级3位学号)，

学生成绩的取值范围为[0,100]之间的整数，

两门选修课选修学生数的取值范围为[1–2000]之间的整数。

输出描述

同时选修了两门选修课的学生的学号，如果没有同时选修两门选修课的学生输出NULL，

否则，先按照班级划分，班级编号小的先输出，每个班级先输出班级编号(学号前五位)，然后另起一行输出这个班级同时选修两门选修课的学生学号，学号按照要求排序(按照两门选修课成绩和的降序，成绩和相同时按照学号升序)，学生之间以英文分号分隔。

示例1

输入



Plain Text

```
1 01202021,75;01201033,95;01202008,80;01203006,90;01203088,100
2 01202008,70;01203088,85;01202111,80;01202021,75;01201100,88
```

输出



Plain Text

```
1 01202
2 01202008;01202021
3 01203
4 01203088
```

说明

同时选修了两门选修课的学生01202021、01202008、01203088，这三个学生两门选修课的成绩和分别为150、150、185，01202021、01202008属于01202班的学生，按照成绩和降序，成绩相同时按学号升序输出的结果为01202008;01202021,01203088属于01203班的学生，按照成绩和降序，成绩相同时按学号升序输出的结果为01203088，01202的班级编号小于01203的班级编号，需要先输出。

示例2

输入

```
Plain Text |  
1 01201022,75;01202033,95;01202018,80;01203006,90;01202066,100  
2 01202008,70;01203102,85;01202111,80;01201021,75;01201100,88
```

输出

```
Plain Text |  
1 NULL
```

说明

没有同时选修了两门选修课的学生，输出NULL。

题解

思路： 模拟

- 接收选修第一堂课的学生信息，使用 哈希表<学号，分数> 或者类似结构存储选修第一门的学生信息。
- 接收选修第二堂课的学生信息时，处理选修当前课的是否选修了第一门，如果选修了，存储至另一个 哈希表 键为 班级 值为 属于这个班级的学生信息。 分类存储
- 处理完第二步之后，首先进行班级内同学的排序，按照 每个班级按照两门选修课成绩和的降序排序，成绩相同时按照学生的学号升序排序 进行排序。 多条件排序
- 处理完第三步之后，处理班与班之间排序，按照 班级编号编号进行升序排序。
- 按顺序输出每一个班级的每一个学生id。

C++

```
1 #include <iostream>
2 #include <unordered_map>
3 #include <map>
4 #include <set>
5 #include <sstream>
6 #include <string>
7 #include <vector>
8
9 using namespace std;
10
11 struct Student {
12     string id;
13     int score;
14     Student(const string& id, int score) : id(id), score(score) {}
15
16     // 自定义排序规则 (按成绩降序, id 升序)
17     bool operator<(const Student& other) const {
18         return score != other.score ? score > other.score : id < other.id;
19     }
20 };
21
22 // 解析输入字符串, 返回分割后的数据
23 vector<vector<string>> parseStudents(const string& line, char delim1, char delim2) {
24     vector<vector<string>> result;
25     stringstream ss(line), ss2;
26     string token, subtoken;
27
28     while (getline(ss, token, delim1)) {
29         vector<string> student;
30         ss2.clear();
31         ss2.str(token);
32         while (getline(ss2, subtoken, delim2)) {
33             student.push_back(subtoken);
34         }
35         result.push_back(student);
36     }
37     return result;
38 }
39
40 // 检查学生是否在教师列表中, 计算总成绩
41 bool isValidStudent(const unordered_map<string, int>& tIds, const vector<string>& sStu, int& totalScore) {
42     if (tIds.count(sStu[0])) {
```

```
43         totalScore = stoi(sStu[1]) + tIds.at(sStu[0]);
44     return true;
45 }
46 return false;
47 }

48

49 int main() {
50     string line_one, line_two;
51     getline(cin, line_one);
52     getline(cin, line_two);

53     // 解析学生ID 和分数映射表
54     unordered_map<string, int> tIds;
55     for (const auto& tStu : parseStudents(line_two, ',', ',')) {
56         tIds[tStu[0]] = stoi(tStu[1]);
57     }

58

59     // 存储班级学生列表 (按班级 key 自动排序)
60     map<string, set<Student>> classMap;
61

62     for (const auto& sStu : parseStudents(line_one, ',', ',')) {
63         int totalScore;
64         if (isValidStudent(tIds, sStu, totalScore)) {
65             string cls = sStu[0].substr(0, 5); // 提取班级 ID
66             classMap[cls].emplace(sStu[0], totalScore);
67         }
68     }
69 }

70     // 输出结果
71     if (classMap.empty()) {
72         cout << "NULL\n";
73     } else {
74         for (const auto& [cls, students] : classMap) {
75             cout << cls << "\n";
76             string res;
77             for (const auto& student : students) {
78                 res += student.id + ";";
79             }
80             res.pop_back(); // 移除最后的 ";"
81             cout << res << "\n";
82         }
83     }
84 }

85

86     return 0;
87 }
```

Java

```
1 import java.util.*;
2
3 class Student implements Comparable<Student> {
4     String id;
5     int score;
6
7     public Student(String id, int score) {
8         this.id = id;
9         this.score = score;
10    }
11
12    @Override
13    public int compareTo(Student other) {
14        return this.score != other.score ? Integer.compare(other.score, this.score) : this.id.compareTo(other.id);
15    }
16 }
17
18 public class Main {
19     // 解析输入字符串，返回分割后的数据
20     private static List<String[]> parseStudents(String line, String delim1, String delim2) {
21         List<String[]> result = new ArrayList<>();
22         for (String token : line.split(delim1)) {
23             result.add(token.split(delim2));
24         }
25         return result;
26     }
27
28     public static void main(String[] args) {
29         Scanner sc = new Scanner(System.in);
30         String lineOne = sc.nextLine();
31         String lineTwo = sc.nextLine();
32         sc.close();
33
34         // 解析学生ID 和分数映射表
35         Map<String, Integer> tIds = new HashMap<>();
36         for (String[] tStu : parseStudents(lineTwo, ";", ",")) {
37             tIds.put(tStu[0], Integer.parseInt(tStu[1]));
38         }
39
40         // 存储班级学生列表 (按班级 key 自动排序)
41         TreeMap<String, TreeSet<Student>> classMap = new TreeMap<>();
42 }
```

```
43     for (String[] sStu : parseStudents(lineOne, ";", ",")) {
44         String id = sStu[0];
45         if (tIds.containsKey(id)) {
46             int totalScore = Integer.parseInt(sStu[1]) + tIds.get(id);
47             String cls = id.substring(0, 5);
48             classMap.putIfAbsent(cls, new TreeSet<>());
49             classMap.get(cls).add(new Student(id, totalScore));
50         }
51     }
52
53     // 输出结果
54     if (classMap.isEmpty()) {
55         System.out.println("NULL");
56     } else {
57         for (Map.Entry<String, TreeSet<Student>> entry : classMap.entrySet()) {
58             System.out.println(entry.getKey());
59             System.out.println(String.join(";", entry.getValue().stream()
60                                         .map(s -> s.id).toArray(String[]::new)));
61         }
62     }
63 }
```

Python

```
1 import sys
2 from collections import defaultdict
3
4 class Student:
5     """学生类, 定义排序规则 (按成绩降序, ID 升序) """
6     def __init__(self, student_id, score):
7         self.student_id = student_id
8         self.score = score
9
10    def __lt__(self, other):
11        """定义排序规则: 成绩高的排前面, 成绩相同按ID字典序排"""
12        return (self.score, other.student_id) > (other.score, self.student_id)
13
14
15    def parse_students(line, delim1, delim2):
16        """解析输入字符串, 返回分割后的数据"""
17        students = []
18        for token in line.strip().split(delim1):
19            students.append(token.split(delim2))
20        return students
21
22
23    def main():
24        # 读取输入
25        line_one = sys.stdin.readline().strip()
26        line_two = sys.stdin.readline().strip()
27
28        # 解析教师评分列表
29        teacher_scores = {t[0]: int(t[1]) for t in parse_students(line_two,
30            ';;', ',')}
31
32        # 存储班级学生 (使用字典, 值是有序集合)
33        class_map = defaultdict(set)
34
35        for student in parse_students(line_one, ';;', ','):
36            student_id, score = student[0], int(student[1])
37            if student_id in teacher_scores:
38                total_score = score + teacher_scores[student_id]
39                class_id = student_id[:5] # 班级 ID 取前 5 位
40                class_map[class_id].add(Student(student_id, total_score))
41
42        # 输出结果
43        if not class_map:
```

```
43         print("NULL")
44     else:
45         for class_id in sorted(class_map.keys()):
46             print(class_id)
47             sorted_students = sorted(class_map[class_id]) # 按成绩和 ID 排
序
48             print(";" .join(student.student_id for student in sorted_studen
ts))
49
50
51 if __name__ == "__main__":
52     main()
```

JavaScript

```

1 const fs = require("fs");
2
3 // 解析输入字符串，返回分割后的数据
4 function parseStudents(line, delim1, delim2) {
5     return line.trim().split(delim1).map(token => token.split(delim2));
6 }
7
8 // 读取输入
9 const input = fs.readFileSync(0, "utf-8").trim().split("\n");
10 const lineOne = input[0];
11 const lineTwo = input[1];
12
13 // 解析教师 ID 和分数映射表
14 const tIds = Object.fromEntries(parseStudents(lineTwo, ";", ",").map(([i
d, score]) => [id, parseInt(score)]));
15
16 // 存储班级学生列表（按班级 key 自动排序）
17 const classMap = new Map();
18
19 parseStudents(lineOne, ";", ",").forEach(([sId, sScore]) => {
20     if (tIds.hasOwnProperty(sId)) {
21         const totalScore = parseInt(sScore) + tIds[sId];
22         const cls = sId.slice(0, 5);
23         if (!classMap.has(cls)) classMap.set(cls, new Set());
24         classMap.get(cls).add({ id: sId, score: totalScore });
25     }
26 });
27
28 // 输出结果
29 if (classMap.size === 0) {
30     console.log("NULL");
31 } else {
32     [...classMap.keys()].sort().forEach(cls => {
33         console.log(cls);
34         console.log([...classMap.get(cls)].sort((a, b) => b.score - a.scor
e || a.id.localeCompare(b.id)).map(s => s.id).join(";"));
35     });
36 }

```

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strings"
9     "strconv"
10    )
11
12 // 解析输入字符串，返回分割后的数据
13 func parseStudents(line, delim1, delim2 string) [][]string {
14     tokens := strings.Split(strings.TrimSpace(line), delim1)
15     result := make([][]string, len(tokens))
16     for i, token := range tokens {
17         result[i] = strings.Split(token, delim2)
18     }
19     return result
20 }
21
22 // Student 结构体
23 type Student struct {
24     id    string
25     score int
26 }
27
28 // 自定义排序
29 type ByScore []Student
30
31 func (s ByScore) Len() int          { return len(s) }
32 func (s ByScore) Swap(i, j int)     { s[i], s[j] = s[j], s[i] }
33 func (s ByScore) Less(i, j int) bool {
34     if s[i].score != s[j].score {
35         return s[i].score > s[j].score
36     }
37     return s[i].id < s[j].id
38 }
39
40 func main() {
41     scanner := bufio.NewScanner(os.Stdin)
42
43     scanner.Scan()
44     lineOne := scanner.Text()
```

```
45     scanner.Scan()
46     lineTwo := scanner.Text()
47
48     // 解析教师 ID 和分数映射表
49     tIds := make(map[string]int)
50     for _, tStu := range parseStudents(lineTwo, ";", ",") {
51         tIds[tStu[0]], _ = strconv.Atoi(tStu[1])
52     }
53
54     // 存储班级学生列表
55     classMap := make(map[string][]Student)
56
57     for _, sStu := range parseStudents(lineOne, ";", ",") {
58         id := sStu[0]
59         if score, ok := tIds[id]; ok {
60             totalScore, _ := strconv.Atoi(sStu[1])
61             cls := id[:5]
62             classMap[cls] = append(classMap[cls], Student{id, totalScore + score})
63         }
64     }
65
66     if len(classMap) == 0 {
67         fmt.Println("NULL")
68         return
69     }
70
71     // 输出排序结果
72     keys := make([]string, 0, len(classMap))
73     for k := range classMap {
74         keys = append(keys, k)
75     }
76     sort.Strings(keys)
77
78     for _, cls := range keys {
79         fmt.Println(cls)
80         sort.Sort(ByScore(classMap[cls]))
81         var ids []string
82         for _, s := range classMap[cls] {
83             ids = append(ids, s.id)
84         }
85         fmt.Println(strings.Join(ids, ";"))
86     }
87 }
```

| 来自: 华为OD 2025 B卷 – 选修课 (机试 100分)–CSDN博客

| 来自: 华为OD 2025 B卷 – 选修课 (机试 100分)–CSDN博客

2025 华为OD机试 - 考勤信息 (2025 B卷 100分)-CSDN博客

考勤信息

华为OD机试2025B卷真题目录: [点击查看](#)

2025 B卷 100分题型

题目描述

公司用一个[字符串](#)来表示员工的出勤信息

- absent: 缺勤
- late: 迟到
- leaveearly: 早退
- present: 正常上班

现需根据员工出勤信息，判断本次是否能获得出勤奖，能获得出勤奖的条件如下：

- 缺勤不超过一次；
- 没有连续的迟到/早退；
- 任意连续7次考勤，缺勤/迟到/早退不超过3次。

输入描述

用户的考勤数据字符串

- 记录条数 ≥ 1 ；
- 输入字符串长度 < 10000 ；
- 不存在非法输入；

输出描述

根据考勤数据字符串，如果能得到考勤奖，输出”true”；否则输出”false”，

对于输入示例的结果应为：

true false

示例1

输入

```
1 2  
2 present  
3 present present
```

Plain Text |

输出

```
1 true true
```

Plain Text |

示例2

输入

```
1 2  
2 present  
3 present absent present present leaveearly present absent
```

Plain Text |

输出

```
1 true false
```

Plain Text |

题解

思路： 模拟 + 滑动窗口，处理每个人的考勤信息，判断是否能获得全勤奖的逻辑如下：

- 判断每个打卡记录是否存在缺勤，存在返回false。
- 使用 滑动窗口 统计连续7天内非出勤计数，超过3返回false。
- 判断是否存在连续非出勤情况，存在返回false。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 using namespace std;
6
7 // 判断是否能获得考勤奖
8 bool can_receive_award(const vector<string>& records) {
9     int absent_count = 0; // 缺勤计数
10    int count_in_7_days = 0; // 连续7天内非出勤计数
11    int n = records.size(); // 记录总天数
12
13    for (int i = 0; i < n; ++i) {
14        if (records[i] == "absent") {
15            if (++absent_count > 1) return false; // 超过1次缺勤
16        }
17        if (i > 0 && (records[i] == "late" || records[i] == "leaveearly"))
18            &&
19            (records[i - 1] == "late" || records[i - 1] == "leaveearly"))
20    {
21        return false; // 连续迟到或早退
22    }
23
24    // 维护滑动窗口
25    if (records[i] != "present") count_in_7_days++;
26    if (i >= 7 && records[i - 7] != "present") count_in_7_days--;
27    移除滑动窗口外的天数
28
29    if (i >= 6 && count_in_7_days > 3) return false; // 连续7天内非出勤
30    天数超过3
31    }
32    return true;
33 }
34
35 int main() {
36     int test_cases;
37     cin >> test_cases;
38     cin.ignore(); // 忽略换行符
39     while (test_cases--) {
40         string line;
41         getline(cin, line);
42         vector<string> records;
43         istringstream ss(line);
44         string word;
```

```
41     while (ss >> word) {
42         records.push_back(word);
43     }
44     cout << (can_receive_award(records) ? "true" : "false") << " ";
45 }
46 cout << endl;
47 return 0;
48 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否能获得考勤奖
5     public static boolean canReceiveAward(List<String> records) {
6         int absentCount = 0; // 缺勤计数
7         int countIn7Days = 0; // 连续7天内非出勤计数
8         int n = records.size(); // 记录总天数
9
10        for (int i = 0; i < n; i++) {
11            if (records.get(i).equals("absent")) {
12                if (++absentCount > 1) return false; // 超过1次缺勤
13            }
14            if (i > 0 && (records.get(i).equals("late") || records.get(i).
15                equals("leaveearly")) &&
16                (records.get(i - 1).equals("late") || records.get(i - 1).e
17                quals("leaveearly"))) {
18                return false; // 连续迟到或早退
19            }
20
21            // 维护滑动窗口
22            if (!records.get(i).equals("present")) countIn7Days++; // 当前
23            天不是出勤
24            if (i >= 7 && !records.get(i - 7).equals("present")) countIn7D
25            ays--; // 移除滑动窗口外的天数
26
27            if (i >= 6 && countIn7Days > 3) return false; // 连续7天内非出勤
28            天数超过3
29        }
30        return true;
31    }
32
33    public static void main(String[] args) {
34        Scanner scanner = new Scanner(System.in);
35        int testCases = Integer.parseInt(scanner.nextLine()); // 读取测试用
36        例数量
37        while (testCases-- > 0) {
38            List<String> records = Arrays.asList(scanner.nextLine().split
39                (" ")); // 读取考勤记录
40            System.out.print(canReceiveAward(records) ? "true " : "false
41                ");
42        }
43        System.out.println();
44        scanner.close();
45    }
46}
```

```
37     }
38 }
```

Python

```
Plain Text |
```

```
1 import sys
2
3 def can_receive_award(records):
4     absent_count = 0 # 缺勤计数
5     count_in_7_days = 0 # 连续7天内非出勤计数
6     n = len(records) # 记录总天数
7
8     for i in range(n):
9         if records[i] == "absent":
10            absent_count += 1
11            if absent_count > 1:
12                return False # 超过1次缺勤
13
14            if i > 0 and records[i] in ("late", "leaveearly") and records[i - 1] in ("late", "leaveearly"):
15                return False # 连续迟到或早退
16
17            # 维护滑动窗口
18            if records[i] != "present":
19                count_in_7_days += 1
20            if i >= 7 and records[i - 7] != "present":
21                count_in_7_days -= 1 # 移除滑动窗口外的天数
22
23            if i >= 6 and count_in_7_days > 3:
24                return False # 连续7天内非出勤天数超过3
25
26    return True
27
28 if __name__ == "__main__":
29     test_cases = int(sys.stdin.readline().strip()) # 读取测试用例数量
30     results = []
31     for _ in range(test_cases):
32         records = sys.stdin.readline().strip().split() # 读取考勤记录
33         results.append("true" if can_receive_award(records) else "false")
34
35 print(" ".join(results))
```

JavaScript

```

1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let testCases = -1;
9  let results = [];
10
11 rl.on("line", (line) => {
12     if (testCases === -1) {
13         testCases = parseInt(line.trim()); // 读取测试用例数量
14     } else {
15         let records = line.trim().split(" "); // 读取考勤记录
16         results.push(canReceiveAward(records) ? "true" : "false");
17         if (--testCases === 0) {
18             console.log(results.join(" "));
19             rl.close();
20         }
21     }
22 });
23
24 // 判断是否能获得考勤奖
25 function canReceiveAward(records) {
26     let absentCount = 0; // 缺勤计数
27     let countIn7Days = 0; // 连续7天内非出勤计数
28     let n = records.length; // 记录总天数
29
30     for (let i = 0; i < n; i++) {
31         if (records[i] === "absent") {
32             if (++absentCount > 1) return false; // 超过1次缺勤
33         }
34         if (i > 0 && (records[i] === "late" || records[i] === "leaveearly") &&
35             (records[i - 1] === "late" || records[i - 1] === "leaveearly")) {
36             return false; // 连续迟到或早退
37         }
38
39         // 维护滑动窗口
40         if (records[i] !== "present") countIn7Days++; // 当前天不是出勤
41         if (i >= 7 && records[i - 7] !== "present") countIn7Days--; // 移
除滑动窗口外的天数

```

```
42             if (i >= 6 && countIn7Days > 3) return false; // 连续7天内非出勤天数  
43             超过3  
44         }  
45     return true;  
46 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 判断是否能获得考勤奖
11 func canReceiveAward(records []string) bool {
12     absentCount := 0    // 缺勤计数
13     countIn7Days := 0   // 连续7天内非出勤计数
14     n := len(records)  // 记录总天数
15
16     for i := 0; i < n; i++ {
17         if records[i] == "absent" {
18             absentCount++
19             if absentCount > 1 {
20                 return false // 超过1次缺勤
21             }
22         }
23         if i > 0 && (records[i] == "late" || records[i] == "leaveearly") &&
24             (records[i-1] == "late" || records[i-1] == "leaveearly") {
25             return false // 连续迟到或早退
26         }
27
28         // 维护滑动窗口
29         if records[i] != "present" {
30             countIn7Days++
31         }
32         if i >= 7 && records[i-7] != "present" {
33             countIn7Days-- // 移除滑动窗口外的天数
34         }
35
36         if i >= 6 && countIn7Days > 3 {
37             return false // 连续7天内非出勤天数超过3
38         }
39     }
40     return true
41 }
42
43 func main() {
44     scanner := bufio.NewScanner(os.Stdin)
```

```
45     scanner.Scan()
46     testCases := 0
47     fmt.Sscanf(scanner.Text(), "%d", &testCases)
48
49     var results []string
50     for i := 0; i < testCases; i++ {
51         scanner.Scan()
52         records := strings.Split(scanner.Text(), " ")
53         if canReceiveAward(records) {
54             results = append(results, "true")
55         } else {
56             results = append(results, "false")
57         }
58     }
59     fmt.Println(strings.Join(results, " "))
60 }
```

考勤信息

华为OD机试2025B卷真题目录: [点击查看](#)

2025 B卷 100分题型

题目描述

公司用一个[字符串](#)来表示员工的出勤信息

- absent: 缺勤
- late: 迟到
- leaveearly: 早退
- present: 正常上班

现需根据员工出勤信息，判断本次是否能获得出勤奖，能获得出勤奖的条件如下：

- 缺勤不超过一次；
- 没有连续的迟到/早退；
- 任意连续7次考勤，缺勤/迟到/早退不超过3次。

输入描述

用户的考勤数据字符串

- 记录条数 ≥ 1 ；
- 输入字符串长度 < 10000 ；
- 不存在非法输入；

输出描述

根据考勤数据字符串，如果能得到考勤奖，输出”true”；否则输出”false”，
对于输入示例的结果应为：

```
true false
```

示例1

输入

```
1 2
2 present
3 present present
```

Plain Text |

输出

```
1 true true
```

Plain Text |

示例2

输入

```
1 2
2 present
3 present absent present present leaveearly present absent
```

Plain Text |

输出

```
1 true false
```

Plain Text |

题解

思路： 模拟 + 滑动窗口，处理每个人的考勤信息，判断是否能获得全勤奖的逻辑如下：

- 判断每个打卡记录是否存在缺勤，存在返回false。
- 使用 滑动窗口 统计连续7天内非出勤计数，超过3返回false。
- 判断是否存在连续非出勤情况，存在返回false。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 using namespace std;
6
7 // 判断是否能获得考勤奖
8 bool can_receive_award(const vector<string>& records) {
9     int absent_count = 0; // 缺勤计数
10    int count_in_7_days = 0; // 连续7天内非出勤计数
11    int n = records.size(); // 记录总天数
12
13    for (int i = 0; i < n; ++i) {
14        if (records[i] == "absent") {
15            if (++absent_count > 1) return false; // 超过1次缺勤
16        }
17        if (i > 0 && (records[i] == "late" || records[i] == "leaveearly"))
18            &&
19            (records[i - 1] == "late" || records[i - 1] == "leaveearly"))
20    {
21        return false; // 连续迟到或早退
22    }
23
24    // 维护滑动窗口
25    if (records[i] != "present") count_in_7_days++;
26    if (i >= 7 && records[i - 7] != "present") count_in_7_days--;
27    移除滑动窗口外的天数
28
29    if (i >= 6 && count_in_7_days > 3) return false; // 连续7天内非出勤
30    天数超过3
31    }
32    return true;
33 }
34
35 int main() {
36     int test_cases;
37     cin >> test_cases;
38     cin.ignore(); // 忽略换行符
39     while (test_cases--) {
40         string line;
41         getline(cin, line);
42         vector<string> records;
43         istringstream ss(line);
44         string word;
```

```
41     while (ss >> word) {
42         records.push_back(word);
43     }
44     cout << (can_receive_award(records) ? "true" : "false") << " ";
45 }
46 cout << endl;
47 return 0;
48 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否能获得考勤奖
5     public static boolean canReceiveAward(List<String> records) {
6         int absentCount = 0; // 缺勤计数
7         int countIn7Days = 0; // 连续7天内非出勤计数
8         int n = records.size(); // 记录总天数
9
10        for (int i = 0; i < n; i++) {
11            if (records.get(i).equals("absent")) {
12                if (++absentCount > 1) return false; // 超过1次缺勤
13            }
14            if (i > 0 && (records.get(i).equals("late") || records.get(i).
15                equals("leaveearly")) &&
16                (records.get(i - 1).equals("late") || records.get(i - 1).e
17                quals("leaveearly"))) {
18                return false; // 连续迟到或早退
19            }
20
21            // 维护滑动窗口
22            if (!records.get(i).equals("present")) countIn7Days++; // 当前
23            天不是出勤
24            if (i >= 7 && !records.get(i - 7).equals("present")) countIn7D
25            ays--; // 移除滑动窗口外的天数
26
27            if (i >= 6 && countIn7Days > 3) return false; // 连续7天内非出勤
28            天数超过3
29        }
30        return true;
31    }
32
33    public static void main(String[] args) {
34        Scanner scanner = new Scanner(System.in);
35        int testCases = Integer.parseInt(scanner.nextLine()); // 读取测试用
36        例数量
37        while (testCases-- > 0) {
38            List<String> records = Arrays.asList(scanner.nextLine().split
39                (" ")); // 读取考勤记录
40            System.out.print(canReceiveAward(records) ? "true " : "false
41                ");
42        }
43        System.out.println();
44        scanner.close();
45    }
46}
```

```
37     }
38 }
```

Python

```
Plain Text |
```

```
1 import sys
2
3 def can_receive_award(records):
4     absent_count = 0 # 缺勤计数
5     count_in_7_days = 0 # 连续7天内非出勤计数
6     n = len(records) # 记录总天数
7
8     for i in range(n):
9         if records[i] == "absent":
10            absent_count += 1
11            if absent_count > 1:
12                return False # 超过1次缺勤
13
14            if i > 0 and records[i] in ("late", "leaveearly") and records[i - 1] in ("late", "leaveearly"):
15                return False # 连续迟到或早退
16
17            # 维护滑动窗口
18            if records[i] != "present":
19                count_in_7_days += 1
20            if i >= 7 and records[i - 7] != "present":
21                count_in_7_days -= 1 # 移除滑动窗口外的天数
22
23            if i >= 6 and count_in_7_days > 3:
24                return False # 连续7天内非出勤天数超过3
25
26    return True
27
28 if __name__ == "__main__":
29     test_cases = int(sys.stdin.readline().strip()) # 读取测试用例数量
30     results = []
31     for _ in range(test_cases):
32         records = sys.stdin.readline().strip().split() # 读取考勤记录
33         results.append("true" if can_receive_award(records) else "false")
34
35 print(" ".join(results))
```

JavaScript

```

1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let testCases = -1;
9  let results = [];
10
11 rl.on("line", (line) => {
12     if (testCases === -1) {
13         testCases = parseInt(line.trim()); // 读取测试用例数量
14     } else {
15         let records = line.trim().split(" "); // 读取考勤记录
16         results.push(canReceiveAward(records) ? "true" : "false");
17         if (--testCases === 0) {
18             console.log(results.join(" "));
19             rl.close();
20         }
21     }
22 });
23
24 // 判断是否能获得考勤奖
25 function canReceiveAward(records) {
26     let absentCount = 0; // 缺勤计数
27     let countIn7Days = 0; // 连续7天内非出勤计数
28     let n = records.length; // 记录总天数
29
30     for (let i = 0; i < n; i++) {
31         if (records[i] === "absent") {
32             if (++absentCount > 1) return false; // 超过1次缺勤
33         }
34         if (i > 0 && (records[i] === "late" || records[i] === "leaveearly") &&
35             (records[i - 1] === "late" || records[i - 1] === "leaveearly")) {
36             return false; // 连续迟到或早退
37         }
38
39         // 维护滑动窗口
40         if (records[i] !== "present") countIn7Days++; // 当前天不是出勤
41         if (i >= 7 && records[i - 7] !== "present") countIn7Days--; // 移
除滑动窗口外的天数

```

```
42             if (i >= 6 && countIn7Days > 3) return false; // 连续7天内非出勤天数  
43             超过3  
44         }  
45     return true;  
46 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 判断是否能获得考勤奖
11 func canReceiveAward(records []string) bool {
12     absentCount := 0    // 缺勤计数
13     countIn7Days := 0   // 连续7天内非出勤计数
14     n := len(records)  // 记录总天数
15
16     for i := 0; i < n; i++ {
17         if records[i] == "absent" {
18             absentCount++
19             if absentCount > 1 {
20                 return false // 超过1次缺勤
21             }
22         }
23         if i > 0 && (records[i] == "late" || records[i] == "leaveearly") &&
24             (records[i-1] == "late" || records[i-1] == "leaveearly") {
25             return false // 连续迟到或早退
26         }
27
28         // 维护滑动窗口
29         if records[i] != "present" {
30             countIn7Days++
31         }
32         if i >= 7 && records[i-7] != "present" {
33             countIn7Days-- // 移除滑动窗口外的天数
34         }
35
36         if i >= 6 && countIn7Days > 3 {
37             return false // 连续7天内非出勤天数超过3
38         }
39     }
40     return true
41 }
42
43 func main() {
44     scanner := bufio.NewScanner(os.Stdin)
```

```
45     scanner.Scan()
46     testCases := 0
47     fmt.Sscanf(scanner.Text(), "%d", &testCases)
48
49     var results []string
50     for i := 0; i < testCases; i++ {
51         scanner.Scan()
52         records := strings.Split(scanner.Text(), " ")
53         if canReceiveAward(records) {
54             results = append(results, "true")
55         } else {
56             results = append(results, "false")
57         }
58     }
59     fmt.Println(strings.Join(results, " "))
60 }
```

来自: 2025 华为OD机试 – 考勤信息 (2025 B卷 100分)-CSDN博客

考勤信息

华为OD机试2025B卷真题目录: [点击查看](#)

2025 B卷 100分题型

题目描述

公司用一个**字符串**来表示员工的出勤信息

- absent: 缺勤
- late: 迟到
- leaveearly: 早退
- present: 正常上班

现需根据员工出勤信息, 判断本次是否能获得出勤奖, 能获得出勤奖的条件如下:

- 缺勤不超过一次;
- 没有连续的迟到/早退;
- 任意连续7次考勤, 缺勤/迟到/早退不超过3次。

输入描述

用户的考勤数据字符串

- 记录条数 ≥ 1 ;
- 输入字符串长度 < 10000 ;
- 不存在非法输入;

输出描述

根据考勤数据字符串，如果能得到考勤奖，输出”true”；否则输出”false”，对于输入示例的结果应为：

```
true false
```

示例1

输入

```
Plain Text  
1 2  
2 present  
3 present present
```

输出

```
Plain Text  
1 true true
```

示例2

输入

```
Plain Text  
1 2  
2 present  
3 present absent present present leaveearly present absent
```

输出

```
Plain Text  
1 true false
```

题解

思路： 模拟 + 滑动窗口，处理每个人的考勤信息，判断是否能获得全勤奖的逻辑如下：

- 判断每个打卡记录是否存在缺勤，存在返回false。
- 使用 滑动窗口 统计连续7天内非出勤计数，超过3返回false。

- 判断是否存在连续非出勤情况，存在返回false。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5 using namespace std;
6
7 // 判断是否能获得考勤奖
8 bool can_receive_award(const vector<string>& records) {
9     int absent_count = 0; // 缺勤计数
10    int count_in_7_days = 0; // 连续7天内非出勤计数
11    int n = records.size(); // 记录总天数
12
13    for (int i = 0; i < n; ++i) {
14        if (records[i] == "absent") {
15            if (++absent_count > 1) return false; // 超过1次缺勤
16        }
17        if (i > 0 && (records[i] == "late" || records[i] == "leaveearly"))
18            &&
19            (records[i - 1] == "late" || records[i - 1] == "leaveearly"))
20    {
21        return false; // 连续迟到或早退
22    }
23
24    // 维护滑动窗口
25    if (records[i] != "present") count_in_7_days++; // 当前天不是出勤
26    if (i >= 7 && records[i - 7] != "present") count_in_7_days--; // 移除滑动窗口外的天数
27
28    if (i >= 6 && count_in_7_days > 3) return false; // 连续7天内非出勤天数超过3
29    }
30
31    return true;
32
33    int main() {
34        int test_cases;
35        cin >> test_cases;
36        cin.ignore(); // 忽略换行符
37        while (test_cases--) {
38            string line;
39            getline(cin, line);
40            vector<string> records;
41            istringstream ss(line);
42            string word;
```

```
41     while (ss >> word) {
42         records.push_back(word);
43     }
44     cout << (can_receive_award(records) ? "true" : "false") << " ";
45 }
46 cout << endl;
47 return 0;
48 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否能获得考勤奖
5     public static boolean canReceiveAward(List<String> records) {
6         int absentCount = 0; // 缺勤计数
7         int countIn7Days = 0; // 连续7天内非出勤计数
8         int n = records.size(); // 记录总天数
9
10        for (int i = 0; i < n; i++) {
11            if (records.get(i).equals("absent")) {
12                if (++absentCount > 1) return false; // 超过1次缺勤
13            }
14            if (i > 0 && (records.get(i).equals("late") || records.get(i).
15                equals("leaveearly")) &&
16                (records.get(i - 1).equals("late") || records.get(i - 1).e
17                quals("leaveearly"))) {
18                return false; // 连续迟到或早退
19            }
20
21            // 维护滑动窗口
22            if (!records.get(i).equals("present")) countIn7Days++; // 当前
23            天不是出勤
24            if (i >= 7 && !records.get(i - 7).equals("present")) countIn7D
25            ays--; // 移除滑动窗口外的天数
26
27            if (i >= 6 && countIn7Days > 3) return false; // 连续7天内非出勤
28            天数超过3
29        }
30        return true;
31    }
32
33    public static void main(String[] args) {
34        Scanner scanner = new Scanner(System.in);
35        int testCases = Integer.parseInt(scanner.nextLine()); // 读取测试用
36        例数量
37        while (testCases-- > 0) {
38            List<String> records = Arrays.asList(scanner.nextLine().split
39                (" ")); // 读取考勤记录
40            System.out.print(canReceiveAward(records) ? "true " : "false
41                ");
42        }
43        System.out.println();
44        scanner.close();
45    }
46}
```

```
37     }
38 }
```

Python

```
Plain Text |
```

```
1 import sys
2
3 def can_receive_award(records):
4     absent_count = 0 # 缺勤计数
5     count_in_7_days = 0 # 连续7天内非出勤计数
6     n = len(records) # 记录总天数
7
8     for i in range(n):
9         if records[i] == "absent":
10            absent_count += 1
11            if absent_count > 1:
12                return False # 超过1次缺勤
13
14            if i > 0 and records[i] in ("late", "leaveearly") and records[i - 1] in ("late", "leaveearly"):
15                return False # 连续迟到或早退
16
17            # 维护滑动窗口
18            if records[i] != "present":
19                count_in_7_days += 1
20            if i >= 7 and records[i - 7] != "present":
21                count_in_7_days -= 1 # 移除滑动窗口外的天数
22
23            if i >= 6 and count_in_7_days > 3:
24                return False # 连续7天内非出勤天数超过3
25
26    return True
27
28 if __name__ == "__main__":
29     test_cases = int(sys.stdin.readline().strip()) # 读取测试用例数量
30     results = []
31     for _ in range(test_cases):
32         records = sys.stdin.readline().strip().split() # 读取考勤记录
33         results.append("true" if can_receive_award(records) else "false")
34
35 print(" ".join(results))
```

JavaScript

```

1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let testCases = -1;
9  let results = [];
10
11 rl.on("line", (line) => {
12     if (testCases === -1) {
13         testCases = parseInt(line.trim()); // 读取测试用例数量
14     } else {
15         let records = line.trim().split(" "); // 读取考勤记录
16         results.push(canReceiveAward(records) ? "true" : "false");
17         if (--testCases === 0) {
18             console.log(results.join(" "));
19             rl.close();
20         }
21     }
22 });
23
24 // 判断是否能获得考勤奖
25 function canReceiveAward(records) {
26     let absentCount = 0; // 缺勤计数
27     let countIn7Days = 0; // 连续7天内非出勤计数
28     let n = records.length; // 记录总天数
29
30     for (let i = 0; i < n; i++) {
31         if (records[i] === "absent") {
32             if (++absentCount > 1) return false; // 超过1次缺勤
33         }
34         if (i > 0 && (records[i] === "late" || records[i] === "leaveearly") &&
35             (records[i - 1] === "late" || records[i - 1] === "leaveearly")) {
36             return false; // 连续迟到或早退
37         }
38
39         // 维护滑动窗口
40         if (records[i] !== "present") countIn7Days++; // 当前天不是出勤
41         if (i >= 7 && records[i - 7] !== "present") countIn7Days--; // 移
除滑动窗口外的天数

```

```
42             if (i >= 6 && countIn7Days > 3) return false; // 连续7天内非出勤天数  
43             超过3  
44         }  
45     return true;  
46 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 判断是否能获得考勤奖
11 func canReceiveAward(records []string) bool {
12     absentCount := 0    // 缺勤计数
13     countIn7Days := 0   // 连续7天内非出勤计数
14     n := len(records)  // 记录总天数
15
16     for i := 0; i < n; i++ {
17         if records[i] == "absent" {
18             absentCount++
19             if absentCount > 1 {
20                 return false // 超过1次缺勤
21             }
22         }
23         if i > 0 && (records[i] == "late" || records[i] == "leaveearly") &&
24             (records[i-1] == "late" || records[i-1] == "leaveearly") {
25             return false // 连续迟到或早退
26         }
27
28         // 维护滑动窗口
29         if records[i] != "present" {
30             countIn7Days++
31         }
32         if i >= 7 && records[i-7] != "present" {
33             countIn7Days-- // 移除滑动窗口外的天数
34         }
35
36         if i >= 6 && countIn7Days > 3 {
37             return false // 连续7天内非出勤天数超过3
38         }
39     }
40     return true
41 }
42
43 func main() {
44     scanner := bufio.NewScanner(os.Stdin)
```

```
45     scanner.Scan()
46     testCases := 0
47     fmt.Sscanf(scanner.Text(), "%d", &testCases)
48
49     var results []string
50     for i := 0; i < testCases; i++ {
51         scanner.Scan()
52         records := strings.Split(scanner.Text(), " ")
53         if canReceiveAward(records) {
54             results = append(results, "true")
55         } else {
56             results = append(results, "false")
57         }
58     }
59     fmt.Println(strings.Join(results, " "))
60 }
```

| 来自: 2025 华为OD机试 – 考勤信息 (2025 B卷 100分)-CSDN博客

| 来自: 2025 华为OD机试 – 考勤信息 (2025 B卷 100分)-CSDN博客

华为OD机考 - 找等值元素 / 找数字 (2025 B卷)

100分_华为od机考题目-CSDN博客

找等值元素 / 找数字

华为OD机试真题目录: [点击查看](#)

2025B卷 100分题型

题目描述

给一个二维数组 `nums`, 对于每一个元素 `nums[i]`, 找出距离最近的且值相等的元素,
输出横纵坐标差值的绝对值之和, 如果没有等值元素, 则输出-1。

输入描述

输入第一行为二维数组的行

输入第二行为二维数组的列

输入的数字以空格隔开。

备注

- 针对数组 `nums[i][j]`, 满足 $0 < i \leq 100$, $0 < j \leq 100$
- 对于每个数字, 最多存在 100 个与其相等的数字

输出描述

数组形式返回所有坐标值。

示例1

输入

```
1 3  
2 5  
3 0 3 5 4 2  
4 2 5 7 8 3  
5 2 5 4 2 4
```

Plain Text |

输出

▼

Plain Text |

```
1  [[-1, 4, 2, 3, 3], [1, 1, -1, -1, 4], [1, 1, 2, 3, 2]]
```

题解

思路: 模拟 + 哈希表

1. 使用 哈希表 `map` 数据结构分类存储为所有值相等的坐标。定义 `result[][]` 存储结果
2. 接下来遍历所有二维数组坐标元素，判断每个坐标的结果时，首先检查 `map[ans[i][j]]` 的长度是否为1?
 - 为1直接设置 `result[i][j] = -1`，说明不存在相同值元素。
 - 不为1，遍历`map[ans[i][j]]`找出其中与之哈曼顿距离的坐标，记录最小距离。注意在遍历判断距离时一定要排除本身，不然最小距离都会变成0。找出其中最小距离 `minDistance`，设置 `result[i][j] = minDistance`。
3. 按照示例数据格式输出 `result` 即可。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4 #include <climits>
5 #include <cmath>
6
7 using namespace std;
8
9 int main() {
10     int rows, cols;
11     cin >> rows >> cols;
12
13     vector<vector<int>> matrix(rows, vector<int>(cols));
14     unordered_map<int, vector<pair<int, int>>> positionMap;
15
16     // 读取矩阵数据并记录每个数字出现的位置
17     for (int i = 0; i < rows; i++) {
18         for (int j = 0; j < cols; j++) {
19             cin >> matrix[i][j];
20             positionMap[matrix[i][j]].emplace_back(i, j);
21         }
22     }
23
24     vector<vector<int>> result(rows, vector<int>(cols, -1));
25
26     // 遍历矩阵，计算到最近相同元素的曼哈顿距离
27     for (int i = 0; i < rows; i++) {
28         for (int j = 0; j < cols; j++) {
29             int currentNum = matrix[i][j];
30
31             // 如果该数字出现多次，则计算最小距离
32             if (positionMap[currentNum].size() > 1) {
33                 int minDistance = INT_MAX;
34                 for (const auto& [x, y] : positionMap[currentNum]) {
35                     int distance = abs(x - i) + abs(y - j);
36                     if (distance > 0) { // 排除自身
37                         minDistance = min(minDistance, distance);
38                     }
39                 }
40                 result[i][j] = minDistance;
41             }
42         }
43     }
44 }
```

```
45     // 格式化输出
46     cout << "[";
47     for (int i = 0; i < rows; i++) {
48         cout << "[";
49         for (int j = 0; j < cols; j++) {
50             cout << result[i][j] << (j < cols - 1 ? ", " : "");
51         }
52         cout << "]" << (i < rows - 1 ? ", " : "");
53     }
54     cout << "]";
55
56     return 0;
57 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         int rows = scanner.nextInt();
7         int cols = scanner.nextInt();
8
9         int[][] matrix = new int[rows][cols];
10        Map<Integer, List<int[]>> positionMap = new HashMap<>();
11
12        // 读取矩阵数据并记录每个数字出现的位置
13        for (int i = 0; i < rows; i++) {
14            for (int j = 0; j < cols; j++) {
15                matrix[i][j] = scanner.nextInt();
16                positionMap.computeIfAbsent(matrix[i][j], k -> new ArrayList<>()).add(new int[]{i, j});
17            }
18        }
19
20        int[][] result = new int[rows][cols];
21        for (int[] row : result) {
22            Arrays.fill(row, -1);
23        }
24
25        // 遍历矩阵，计算到最近相同元素的曼哈顿距离
26        for (int i = 0; i < rows; i++) {
27            for (int j = 0; j < cols; j++) {
28                int currentNum = matrix[i][j];
29
30                // 如果该数字出现多次，则计算最小距离
31                if (positionMap.get(currentNum).size() > 1) {
32                    int minDistance = Integer.MAX_VALUE;
33                    for (int[] pos : positionMap.get(currentNum)) {
34                        int x = pos[0], y = pos[1];
35                        int distance = Math.abs(x - i) + Math.abs(y - j);
36                        if (distance > 0) { // 排除自身
37                            minDistance = Math.min(minDistance, distance);
38                        }
39                    }
40                    result[i][j] = minDistance;
41                }
42            }
43        }
44    }
45}
```

```
44 // 格式化输出
45 System.out.print("[");
46 for (int i = 0; i < rows; i++) {
47     System.out.print("[");
48     for (int j = 0; j < cols; j++) {
49         System.out.print(result[i][j]);
50         if (j < cols - 1) System.out.print(", ");
51     }
52     System.out.print("]");
53     if (i < rows - 1) System.out.print(", ");
54 }
55 System.out.println("]");
```

Python

```
1 import sys
2
3 def main():
4     # 读取矩阵尺寸
5     rows = int(input())
6     cols = int(input())
7     matrix = []
8     position_map = {}
9
10    # 读取矩阵数据并记录每个数字出现的位置
11    for i in range(rows):
12        row = list(map(int, sys.stdin.readline().split()))
13        matrix.append(row)
14        for j, num in enumerate(row):
15            if num not in position_map:
16                position_map[num] = []
17                position_map[num].append((i, j))
18
19    result = [[-1] * cols for _ in range(rows)]
20
21    # 遍历矩阵，计算到最近相同元素的曼哈顿距离
22    for i in range(rows):
23        for j in range(cols):
24            current_num = matrix[i][j]
25
26            # 如果该数字出现多次，则计算最小距离
27            if len(position_map[current_num]) > 1:
28                min_distance = float('inf')
29                for x, y in position_map[current_num]:
30                    distance = abs(x - i) + abs(y - j)
31                    if distance > 0: # 排除自身
32                        min_distance = min(min_distance, distance)
33                result[i][j] = min_distance
34
35    # 格式化输出
36    print("[", end="")
37    for i in range(rows):
38        print("[", end="")
39        print(", ".join(map(str, result[i])), end="")
40        print("]", end="")
41        if i < rows - 1:
42            print(", ", end="")
43    print("]")
```

```
45 if __name__ == "__main__":
46     main()
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = [];
9 rl.on("line", (line) => {
10     input.push(line);
11 }).on("close", () => {
12     let rows = parseInt(input[0]); // 读取行数
13     let cols = parseInt(input[1]); // 读取列数
14
15     let matrix = [];
16     let positionMap = new Map();
17
18     // 读取矩阵数据并记录每个数字出现的位置
19     for (let i = 2; i < 2 + rows; i++) {
20         let row = input[i].split(" ").map(Number);
21         matrix.push(row);
22         row.forEach((num, j) => {
23             if (!positionMap.has(num)) {
24                 positionMap.set(num, []);
25             }
26             positionMap.get(num).push([i - 2, j]); // 记录坐标
27         });
28     }
29
30     let result = Array.from({ length: rows }, () => Array(cols).fill(-1));
31
32     // 遍历矩阵，计算到最近相同元素的曼哈顿距离
33     for (let i = 0; i < rows; i++) {
34         for (let j = 0; j < cols; j++) {
35             let currentNum = matrix[i][j];
36
37             if (positionMap.get(currentNum).length > 1) {
38                 let minDistance = Infinity;
39                 for (let [x, y] of positionMap.get(currentNum)) {
40                     let distance = Math.abs(x - i) + Math.abs(y - j);
41                     if (distance > 0) { // 排除自身
42                         minDistance = Math.min(minDistance, distance);
43                     }
44                 }
45             }
46         }
47     }
48 }
```

```
45             result[i][j] = minDistance;
46         }
47     }
48 }
49
50 // 格式化输出
51 console.log("[");
52 console.log(result.map(row => "  [" + row.join(", ") + "]").join
(",\n"));
53     console.log("]");
54 }) .
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9     "math"
10    )
11
12 func main() {
13     scanner := bufio.NewScanner(os.Stdin)
14
15     // 读取行数和列数
16     scanner.Scan()
17     rows, _ := strconv.Atoi(scanner.Text())
18     scanner.Scan()
19     cols, _ := strconv.Atoi(scanner.Text())
20
21     // 读取矩阵数据并记录每个数字出现的位置
22     matrix := make([][]int, rows)
23     positionMap := make(map[int][][]int)
24
25     for i := 0; i < rows; i++ {
26         scanner.Scan()
27         line := strings.Fields(scanner.Text())
28         matrix[i] = make([]int, cols)
29         for j := 0; j < cols; j++ {
30             matrix[i][j], _ = strconv.Atoi(line[j])
31             positionMap[matrix[i][j]] = append(positionMap[matrix[i][j]], []int
32             {i, j})
33         }
34     }
35
36     // 结果矩阵初始化
37     result := make([][]int, rows)
38     for i := range result {
39         result[i] = make([]int, cols)
40         for j := range result[i] {
41             result[i][j] = -1
42         }
43     }
```

```

44     // 遍历矩阵，计算到最近相同元素的曼哈顿距离
45     for i := 0; i < rows; i++ {
46         for j := 0; j < cols; j++ {
47             currentNum := matrix[i][j]
48
49             // 如果该数字出现多次，则计算最小距离
50             if len(positionMap[currentNum]) > 1 {
51                 minDistance := math.MaxInt32
52                 for _, pos := range positionMap[currentNum] {
53                     x, y := pos[0], pos[1]
54                     if x != i || y != j { // 排除自身
55                         distance := abs(x-i) + abs(y-j)
56                         if distance < minDistance {
57                             minDistance = distance
58                         }
59                     }
60                 }
61                 result[i][j] = minDistance
62             }
63         }
64     }
65
66     // 格式化输出（紧凑格式）
67     fmt.Print("[")
68     for i, row := range result {
69         fmt.Print("[")
70         for j, val := range row {
71             fmt.Print(val)
72             if j < cols-1 {
73                 fmt.Print(", ")
74             }
75         }
76         fmt.Print("]")
77         if i < rows-1 {
78             fmt.Print(", ")
79         }
80     }
81     fmt.Println("]")
82 }
83
84     // 计算绝对值
85     func abs(x int) int {
86         if x < 0 {
87             return -x
88         }
89         return x
90     }

```

找等值元素 / 找数字

华为OD机试真题目录: [点击查看](#)

2025B卷 100分题型

题目描述

给一个二维数组 `nums`, 对于每一个元素 `nums[i]`, 找出距离最近的且值相等的元素, 输出横纵坐标差值的绝对值之和, 如果没有等值元素, 则输出-1。

输入描述

输入第一行为二维数组的行

输入第二行为二维数组的列

输入的数字以空格隔开。

备注

- 针对数组 `nums[i][j]`, 满足 $0 < i \leq 100$, $0 < j \leq 100$
- 对于每个数字, 最多存在 100 个与其相等的数字

输出描述

数组形式返回所有坐标值。

示例1

输入

```
1 3
2 5
3 0 3 5 4 2
4 2 5 7 8 3
5 2 5 4 2 4
```

输出

```
1 [[-1, 4, 2, 3, 3], [1, 1, -1, -1, 4], [1, 1, 2, 3, 2]]
```

题解

思路: 模拟 + 哈希表

1. 使用 哈希表 `map` 数据结构分类存储为所有值相等的坐标。定义 `result[][]` 存储结果
2. 接下来遍历所有二维数组坐标元素，判断每个坐标的结果时，首先检查 `map[ans[i][j]]` 的长度是否为1?
 - 为1直接设置 `result[i][j] = -1`，说明不存在相同值元素。
 - 不为1，遍历`map[ans[i][j]]`找出其中与之哈曼顿距离的坐标，记录最小距离。注意在遍历判断距离时一定要排除本身，不然最小距离都会变成0 .找出其中最小距离 `minDistance`，设置 `result[i][j] = minDistance` .
3. 按照示例数据格式输出 `result` 即可。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <unordered_map>
4 #include <climits>
5 #include <cmath>
6
7 using namespace std;
8
9 int main() {
10     int rows, cols;
11     cin >> rows >> cols;
12
13     vector<vector<int>> matrix(rows, vector<int>(cols));
14     unordered_map<int, vector<pair<int, int>>> positionMap;
15
16     // 读取矩阵数据并记录每个数字出现的位置
17     for (int i = 0; i < rows; i++) {
18         for (int j = 0; j < cols; j++) {
19             cin >> matrix[i][j];
20             positionMap[matrix[i][j]].emplace_back(i, j);
21         }
22     }
23
24     vector<vector<int>> result(rows, vector<int>(cols, -1));
25
26     // 遍历矩阵，计算到最近相同元素的曼哈顿距离
27     for (int i = 0; i < rows; i++) {
28         for (int j = 0; j < cols; j++) {
29             int currentNum = matrix[i][j];
30
31             // 如果该数字出现多次，则计算最小距离
32             if (positionMap[currentNum].size() > 1) {
33                 int minDistance = INT_MAX;
34                 for (const auto& [x, y] : positionMap[currentNum]) {
35                     int distance = abs(x - i) + abs(y - j);
36                     if (distance > 0) { // 排除自身
37                         minDistance = min(minDistance, distance);
38                     }
39                 }
40                 result[i][j] = minDistance;
41             }
42         }
43     }
44 }
```

```
45     // 格式化输出
46     cout << "[";
47     for (int i = 0; i < rows; i++) {
48         cout << "[";
49         for (int j = 0; j < cols; j++) {
50             cout << result[i][j] << (j < cols - 1 ? ", " : "");
51         }
52         cout << "]" << (i < rows - 1 ? ", " : "");
53     }
54     cout << "]";
55
56     return 0;
57 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         int rows = scanner.nextInt();
7         int cols = scanner.nextInt();
8
9         int[][] matrix = new int[rows][cols];
10        Map<Integer, List<int[]>> positionMap = new HashMap<>();
11
12        // 读取矩阵数据并记录每个数字出现的位置
13        for (int i = 0; i < rows; i++) {
14            for (int j = 0; j < cols; j++) {
15                matrix[i][j] = scanner.nextInt();
16                positionMap.computeIfAbsent(matrix[i][j], k -> new ArrayList<>()).add(new int[]{i, j});
17            }
18        }
19
20        int[][] result = new int[rows][cols];
21        for (int[] row : result) {
22            Arrays.fill(row, -1);
23        }
24
25        // 遍历矩阵，计算到最近相同元素的曼哈顿距离
26        for (int i = 0; i < rows; i++) {
27            for (int j = 0; j < cols; j++) {
28                int currentNum = matrix[i][j];
29
30                // 如果该数字出现多次，则计算最小距离
31                if (positionMap.get(currentNum).size() > 1) {
32                    int minDistance = Integer.MAX_VALUE;
33                    for (int[] pos : positionMap.get(currentNum)) {
34                        int x = pos[0], y = pos[1];
35                        int distance = Math.abs(x - i) + Math.abs(y - j);
36                        if (distance > 0) { // 排除自身
37                            minDistance = Math.min(minDistance, distance);
38                        }
39                    }
40                    result[i][j] = minDistance;
41                }
42            }
43        }
44    }
45}
```

```
44 // 格式化输出
45 System.out.print("[");
46 for (int i = 0; i < rows; i++) {
47     System.out.print("[");
48     for (int j = 0; j < cols; j++) {
49         System.out.print(result[i][j]);
50         if (j < cols - 1) System.out.print(", ");
51     }
52     System.out.print("]");
53     if (i < rows - 1) System.out.print(", ");
54 }
55 System.out.println("]");
```

Python

```
1 import sys
2
3 def main():
4     # 读取矩阵尺寸
5     rows = int(input())
6     cols = int(input())
7     matrix = []
8     position_map = {}
9
10    # 读取矩阵数据并记录每个数字出现的位置
11    for i in range(rows):
12        row = list(map(int, sys.stdin.readline().split()))
13        matrix.append(row)
14        for j, num in enumerate(row):
15            if num not in position_map:
16                position_map[num] = []
17                position_map[num].append((i, j))
18
19    result = [[-1] * cols for _ in range(rows)]
20
21    # 遍历矩阵，计算到最近相同元素的曼哈顿距离
22    for i in range(rows):
23        for j in range(cols):
24            current_num = matrix[i][j]
25
26            # 如果该数字出现多次，则计算最小距离
27            if len(position_map[current_num]) > 1:
28                min_distance = float('inf')
29                for x, y in position_map[current_num]:
30                    distance = abs(x - i) + abs(y - j)
31                    if distance > 0: # 排除自身
32                        min_distance = min(min_distance, distance)
33                result[i][j] = min_distance
34
35    # 格式化输出
36    print("[", end="")
37    for i in range(rows):
38        print("[", end="")
39        print(", ".join(map(str, result[i])), end="")
40        print("]", end="")
41        if i < rows - 1:
42            print(", ", end="")
43    print("]")
```

```
45 if __name__ == "__main__":
46     main()
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = [];
9 rl.on("line", (line) => {
10     input.push(line);
11 }).on("close", () => {
12     let rows = parseInt(input[0]); // 读取行数
13     let cols = parseInt(input[1]); // 读取列数
14
15     let matrix = [];
16     let positionMap = new Map();
17
18     // 读取矩阵数据并记录每个数字出现的位置
19     for (let i = 2; i < 2 + rows; i++) {
20         let row = input[i].split(" ").map(Number);
21         matrix.push(row);
22         row.forEach((num, j) => {
23             if (!positionMap.has(num)) {
24                 positionMap.set(num, []);
25             }
26             positionMap.get(num).push([i - 2, j]); // 记录坐标
27         });
28     }
29
30     let result = Array.from({ length: rows }, () => Array(cols).fill(-1));
31
32     // 遍历矩阵，计算到最近相同元素的曼哈顿距离
33     for (let i = 0; i < rows; i++) {
34         for (let j = 0; j < cols; j++) {
35             let currentNum = matrix[i][j];
36
37             if (positionMap.get(currentNum).length > 1) {
38                 let minDistance = Infinity;
39                 for (let [x, y] of positionMap.get(currentNum)) {
40                     let distance = Math.abs(x - i) + Math.abs(y - j);
41                     if (distance > 0) { // 排除自身
42                         minDistance = Math.min(minDistance, distance);
43                     }
44                 }
45             }
46         }
47     }
48 }
```

```
45             result[i][j] = minDistance;
46         }
47     }
48 }
49
50 // 格式化输出
51 console.log("[");
52 console.log(result.map(row => "  [" + row.join(", ") + "]").join
(",\n"));
53     console.log("]");
54 }) .
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9     "math"
10    )
11
12 func main() {
13     scanner := bufio.NewScanner(os.Stdin)
14
15     // 读取行数和列数
16     scanner.Scan()
17     rows, _ := strconv.Atoi(scanner.Text())
18     scanner.Scan()
19     cols, _ := strconv.Atoi(scanner.Text())
20
21     // 读取矩阵数据并记录每个数字出现的位置
22     matrix := make([][]int, rows)
23     positionMap := make(map[int][][]int)
24
25     for i := 0; i < rows; i++ {
26         scanner.Scan()
27         line := strings.Fields(scanner.Text())
28         matrix[i] = make([]int, cols)
29         for j := 0; j < cols; j++ {
30             matrix[i][j], _ = strconv.Atoi(line[j])
31             positionMap[matrix[i][j]] = append(positionMap[matrix[i][j]], []int
32             {i, j})
33         }
34     }
35
36     // 结果矩阵初始化
37     result := make([][]int, rows)
38     for i := range result {
39         result[i] = make([]int, cols)
40         for j := range result[i] {
41             result[i][j] = -1
42         }
43     }
```

```

44     // 遍历矩阵，计算到最近相同元素的曼哈顿距离
45     for i := 0; i < rows; i++ {
46         for j := 0; j < cols; j++ {
47             currentNum := matrix[i][j]
48
49             // 如果该数字出现多次，则计算最小距离
50             if len(positionMap[currentNum]) > 1 {
51                 minDistance := math.MaxInt32
52                 for _, pos := range positionMap[currentNum] {
53                     x, y := pos[0], pos[1]
54                     if x != i || y != j { // 排除自身
55                         distance := abs(x-i) + abs(y-j)
56                         if distance < minDistance {
57                             minDistance = distance
58                         }
59                     }
60                 }
61                 result[i][j] = minDistance
62             }
63         }
64     }
65
66     // 格式化输出（紧凑格式）
67     fmt.Print("[")
68     for i, row := range result {
69         fmt.Print("[")
70         for j, val := range row {
71             fmt.Print(val)
72             if j < cols-1 {
73                 fmt.Print(", ")
74             }
75         }
76         fmt.Print("]")
77         if i < rows-1 {
78             fmt.Print(", ")
79         }
80     }
81     fmt.Println("]")
82 }
83
84     // 计算绝对值
85     func abs(x int) int {
86         if x < 0 {
87             return -x
88         }
89         return x
90     }

```

| 来自: 华为OD 机考 – 找等值元素 / 找数字 (2025 B卷) 100分_华为od机考题目-CSDN博客

| 来自: 华为OD 机考 – 找等值元素 / 找数字 (2025 B卷) 100分_华为od机考题目-CSDN博客

华为OD机考 - 字符串计数匹配 (机试 2025 B卷 100分)_华为od计数器算法题-CSDN博客

字符串计数匹配

华为OD机试真题目录: [点击查看](#)

2025B卷 100分题型

题目描述

给你一个字符串str和整数k, 返回满足以下条件的所有子字符串个数:

1. 恰好包含k个字母。
2. 数字0-9各出现至少一次。

输入描述

- 第一行字符串str($1 \leq \text{length} \leq 100000$),仅包含数字和小写字母
- 第二行为整数k($0 \leq k \leq 100000$)

输出描述

输出一个整数, 表示满足所有条件的子字符串的个数。

子字符串是字符串中连续的非空字符序列

示例1

输入

```
▼ Plain Text |  
1 a0123456789aa  
2 1
```

输出

```
▼ Plain Text |  
1 2
```

题解

思路： 双指针 实现，代码的基本逻辑如下

1. 初始定义 `left = 0, right = 0`， 定义 `numberCount` 和 `digitCount[10]` 分别用于记录双指针区间内 数字字符出现的种类数量 和 各个数字字符串出现的次数， 定义 `countChar` 记录双指针区间内 英文字母出现的次数。
2. 简单 `numberCount` 和 `digitCount[10]` 是如何配合使用的，分为两种情况区间增加(`right++`)，区间减少(`left--`)情况下：
 - 区间增大情况下：当前遍历的位置 `input[]` 是数字字母，那么对应 `digitCount[input[right] - '0']++`，如果此时 `digitCount[input[right] - '0'] == 1` 说明之前区间内没有出现此字符，种类加一 `numberCount++`。否则不处理。
 - 区间减少情况下：当前遍历的位置 `input[]` 是数字字母，那么对应 `digitCount[input[right] - '0']--`，如果此时 `digitCount[input[right] - '0'] == 0` 说明缩减后区间内没有此字符了，种类加一 `numberCount--`。否则不处理。
3. 接下来就是简单讲讲双指针移动逻辑，最开始移动右指针尝试增大区间，并更新 `numberCount digitCount[] countChar` 对应的值，当出现以下情况时，需要进行处理：
 - a. 当区间中 英文字符超过 `k` 时，此时需要减少区间大小，让左指针右移，直到区间内 英文字符数量 `== k`。这也是唯一真正移动左指针的情况。
 - b. 当区间内 英文字符数量 `== k` 并且 0-9的数字字符也全部出现 此时需要 枚举当前 `left` 作为起点 范围为 `[left, right)` 终点为 `right` 的子字符串，是否满足要求，并更新满足条件的结果,此时的处理逻辑是： 创建 `left numberCount digitCount[10] countChar` 的备份，尝试模拟左边界收缩情况下，能满足的数量(不会真的移动左边界`left`)。具体逻辑可以参照代码逻辑
4. 循环处理3，直到 `right >= 'len(input)'` 结束

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include <cctype>
8 using namespace std;
9 int main() {
10     string input;
11     int k;
12     getline(cin, input);
13     cin >> k;
14     int n = input.size();
15     // 不满足
16     if (n < 10 + k) {
17         cout << 0;
18         return 0;
19     }
20     // 存储结果
21     int res = 0;
22     vector<int> digitCount(10, 0);
23     // 0 - 9 字符出现种类
24     int numberCount = 0;
25     // 英文字符出现数量
26     int countChar = 0;
27     int left = 0, right = 0;
28     while (right < n) {
29         char c = input[right];
30         if (isalpha(c)) {
31             countChar++;
32         }
33         if (isdigit(c)) {
34             digitCount[c - '0']++;
35             if (digitCount[c - '0'] == 1) {
36                 numberCount++;
37             }
38         }
39     }
40     // 超过k个字符，尝试缩小区间 移动左指针
41     while (left <= right && countChar > k) {
42         char leftChar = input[left];
43         if (isalpha(leftChar)) {
44             countChar--;
```

```

45     }
46     if (isdigit(leftChar)) {
47         digitCount[leftChar - '0']--;
48         if (digitCount[leftChar - '0'] == 0) {
49             numberCount--;
50         }
51     }
52     left++;
53 }
54 // 满足合法序列情况，尝试移动左边界寻求更多合法子串
55 if (numberCount == 10 && countChar == k) {
56     // 不真实移动左指针， 左指针移动只会发生在英文字母字符超过k时 使用备份数
      据进行模拟
57     int tmpLeft = left;
58     vector<int> tempCount = digitCount;
59     int tmpNumberCount = numberCount;
60     int tmpCountChar = countChar;
61     while (tmpLeft <= right && tmpCountChar == k && tmpNumberCoun
t == 10) {
62         res++;
63         char ch = input[tmpLeft];
64         if (isalpha(ch)) {
65             tmpCountChar--;
66         }
67         if (isdigit(ch)) {
68             tempCount[ch - '0']--;
69             if (tempCount[ch - '0'] == 0) {
70                 tmpCountChar--;
71             }
72         }
73         tmpLeft++;
74     }
75 }
76
77     right++;
78 }
79 cout << res;
80 return 0;
81 }

```

JAVA

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         String input = sc.nextLine();
7         int k = sc.nextInt();
8         int n = input.length();
9
10        // 不满足最小长度要求
11        if (n < 10 + k) {
12            System.out.println(0);
13            return;
14        }
15
16        int res = 0;           // 结果统计
17        int[] digitCount = new int[10]; // 数字出现次数
18        int numberCount = 0;          // 当前窗口中数字种类数
19        int countChar = 0;           // 当前窗口中英文字母数量
20        int left = 0;
21
22        for (int right = 0; right < n; right++) {
23            char c = input.charAt(right);
24            if (Character.isLetter(c)) {
25                countChar++;           // 字母计数+1
26            }
27            if (Character.isDigit(c)) {
28                int d = c - '0';
29                digitCount[d]++;
30                if (digitCount[d] == 1) {
31                    numberCount++; // 新增数字种类
32                }
33            }
34
35            // 超过k个字母时，移动左指针缩小窗口
36            while (left <= right && countChar > k) {
37                char leftChar = input.charAt(left);
38                if (Character.isLetter(leftChar)) {
39                    countChar--;
40                }
41                if (Character.isDigit(leftChar)) {
42                    int d = leftChar - '0';
43                    digitCount[d]--;
44                    if (digitCount[d] == 0) {
```

```
45             numberCount--; // 减少数字种类
46         }
47     }
48     left++;
49 }
50
51 // 当数字种类达到10, 字母数正好为k时, 尝试移动左指针寻找更多合法子串
52 if (numberCount == 10 && countChar == k) {
53     // 复制计数数据, 模拟移动左指针
54     int[] tempCount = digitCount.clone();
55     int tempNumberCount = numberCount;
56     int tempCountChar = countChar;
57     int tmpLeft = left;
58     while (tmpLeft <= right && tempNumberCount == 10 && tempCo
untChar == k) {
59         res++;
60         char ch = input.charAt(tmpLeft);
61         if (Character.isLetter(ch)) {
62             tempCountChar--;
63         }
64         if (Character.isDigit(ch)) {
65             int d = ch - '0';
66             tempCount[d]--;
67             if (tempCount[d] == 0) {
68                 tempNumberCount--;
69             }
70         }
71         tmpLeft++;
72     }
73 }
74 System.out.println(res);
75
76 }
77 }
```

Python

```
1 import sys
2
3 input_str = sys.stdin.readline().strip()
4 k = int(sys.stdin.readline())
5
6 n = len(input_str)
7
8 # 不满足最小长度要求, 直接返回0
9 if n < 10 + k:
10     print(0)
11     sys.exit()
12
13 res = 0
14 digit_count = [0] * 10    # 记录数字出现次数
15 unique_digits = 0          # 当前窗口数字种类数
16 letter_count = 0           # 当前窗口英文字母数
17 left = 0
18
19 for right in range(n):
20     c = input_str[right]
21     if c.isalpha():
22         letter_count += 1
23     if c.isdigit():
24         d = int(c)
25         digit_count[d] += 1
26         if digit_count[d] == 1:
27             unique_digits += 1
28
29     # 超过k个字母时, 移动左指针缩小窗口
30     while letter_count > k:
31         ch = input_str[left]
32         if ch.isalpha():
33             letter_count -= 1
34         if ch.isdigit():
35             d = int(ch)
36             digit_count[d] -= 1
37             if digit_count[d] == 0:
38                 unique_digits -= 1
39         left += 1
40
41     # 当数字种类达到10且字母数为k时, 尝试移动左指针找到更多合法子串
42     if unique_digits == 10 and letter_count == k:
43         tmp_left = left
44         temp_count = digit_count[:]
```

```
45     tmp_unique = unique_digits
46     tmp_letter = letter_count
47     while tmp_left <= right and tmp_unique == 10 and tmp_letter == k:
48         res += 1
49         ch = input_str[tmp_left]
50         if ch.isalpha():
51             tmp_letter -= 1
52         if ch.isdigit():
53             d = int(ch)
54             temp_count[d] -= 1
55             if temp_count[d] == 0:
56                 tmp_unique -= 1
57         tmp_left += 1
58
59     print(res)
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on('line', line => {
11     inputLines.push(line.trim());
12     if (inputLines.length === 2) {
13         let inputStr = inputLines[0];
14         let k = parseInt(inputLines[1]);
15         let n = inputStr.length;
16
17         // 不满足最小长度，直接输出0
18         if (n < 10 + k) {
19             console.log(0);
20             rl.close();
21             return;
22         }
23
24         let res = 0;
25         let digitCount = Array(10).fill(0); // 记录数字出现次数
26         let numberCount = 0; // 当前窗口数字种类数
27         let countChar = 0; // 当前窗口字母数
28         let left = 0;
29
30         for (let right = 0; right < n; right++) {
31             let c = inputStr[right];
32             if (/[^a-zA-Z]/.test(c)) {
33                 countChar++; // 字母计数+1
34             }
35             if (/^\d/.test(c)) {
36                 let d = c.charCodeAt(0) - '0'.charCodeAt(0);
37                 digitCount[d]++;
38                 if (digitCount[d] === 1) {
39                     numberCount++; // 新增数字种类
40                 }
41             }
42
43             // 超过k个字母，移动左指针缩小窗口
44             while (left <= right && countChar > k) {
```

```

45         let leftChar = inputStr[left];
46         if (/[^a-zA-Z]/.test(leftChar)) {
47             countChar--;
48         }
49         if (/^\d/.test(leftChar)) {
50             let d = leftChar.charCodeAt(0) - '0'.charCodeAt(0);
51             digitCount[d]--;
52             if (digitCount[d] === 0) {
53                 numberCount--;           // 减少数字种类
54             }
55         }
56         left++;
57     }
58
59     // 当数字种类为10且字母数为k时，模拟移动左指针找更多合法子串
60     if (numberCount === 10 && countChar === k) {
61         let tmpLeft = left;
62         let tempCount = digitCount.slice();
63         let tmpNumberCount = numberCount;
64         let tmpCountChar = countChar;
65         while (tmpLeft <= right && tmpNumberCount === 10 && tmpCountChar === k) {
66             res++;
67             let ch = inputStr[tmpLeft];
68             if (/[^a-zA-Z]/.test(ch)) {
69                 tmpCountChar--;
70             }
71             if (/^\d/.test(ch)) {
72                 let d = ch.charCodeAt(0) - '0'.charCodeAt(0);
73                 tempCount[d]--;
74                 if (tempCount[d] === 0) {
75                     tmpNumberCount--;
76                 }
77             }
78             tmpLeft++;
79         }
80     }
81 }
82 console.log(res);
83 rl.close();
84 }
85 }:

```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "io"
6     "os"
7     "strconv"
8     "strings"
9     "unicode"
10    )
11
12 func main() {
13     // 读取所有输入，兼容大数据量输入
14     data, err := io.ReadAll(os.Stdin)
15     if err != nil {
16         panic(err)
17     }
18
19     // 按换行拆分输入，第一行为字符串，第二行为整数k
20     lines := strings.Split(strings.TrimSpace(string(data)), "\n")
21     if len(lines) < 2 {
22         fmt.Println(0)
23         return
24     }
25
26     input := strings.TrimSpace(lines[0])
27     k, err := strconv.Atoi(strings.TrimSpace(lines[1]))
28     if err != nil {
29         fmt.Println(0)
30         return
31     }
32
33     n := len(input)
34     // 如果输入长度不足10个数字 + k个字符，直接返回0
35     if n < 10+k {
36         fmt.Println(0)
37         return
38     }
39
40     res := 0                      // 结果计数
41     digitCount := make([]int, 10)   // 记录数字0~9的出现次数
42     numberCount := 0                // 当前窗口中不同数字的个数
43     countChar := 0                 // 当前窗口中字母的个数
44     left := 0                     // 窗口左边界
```

```

45
46     for right := 0; right < n; right++ {
47         c := input[right]
48
49         // 如果是字母，增加字符计数
50         if unicode.IsLetter(rune(c)) {
51             countChar++
52         }
53
54         // 如果是数字，增加对应数字的次数
55         if c >= '0' && c <= '9' {
56             d := c - '0'
57             digitCount[d]++
58             if digitCount[d] == 1 {
59                 numberCount++
60             }
61         }
62
63         // 如果字母超过k个，移动左指针缩小窗口
64         for left <= right && countChar > k {
65             leftChar := input[left]
66             if unicode.IsLetter(rune(leftChar)) {
67                 countChar--
68             }
69             if leftChar >= '0' && leftChar <= '9' {
70                 d := leftChar - '0'
71                 digitCount[d]--
72                 if digitCount[d] == 0 {
73                     numberCount--
74                 }
75             }
76             left++
77         }
78
79         // 当前窗口合法：包含10种数字，字符数量为k
80         if numberCount == 10 && countChar == k {
81             // 模拟移动左边界，寻找所有可能的合法子串
82             tmpLeft := left
83             tmpDigitCount := make([]int, 10)
84             copy(tmpDigitCount, digitCount)
85             tmpNumberCount := numberCount
86             tmpCountChar := countChar
87
88             for tmpLeft <= right && tmpNumberCount == 10 && tmpCountChar
89             == k {
90                 res++
91                 ch := input[tmpLeft]

```

```
91             if unicode.IsLetter(rune(ch)) {
92                 tmpCountChar--
93             }
94             if ch >= '0' && ch <= '9' {
95                 d := ch - '0'
96                 tmpDigitCount[d]--
97                 if tmpDigitCount[d] == 0 {
98                     tmpNumberCount--
99                 }
100            }
101            tmpLeft++
102        }
103    }
104 }
105
106 // 输出合法子串数量
107 fmt.Println(res)
```

| 来自: 华为OD 机考 – 字符串计数匹配 (机试 2025 B卷 100分)_华为od 计数器算法题-CSDN博客