

od200

[华为OD机考2025B卷 - 英文输入法（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 水仙花数 I （Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - VLAN资源池（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 简单的自动曝光/平均像素（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 用户调度问题（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 符号运算（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 机房布局（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 字符串重新排序（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 荒岛求生（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机试2025B卷 - 返回矩阵中非1的元素、个数/数值同化（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 查找一个有向网络的头节点和尾节点（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 删除重复数字后的最大数字（Java & Python& JS & C++ & C） -CSDN博客](#)

[华为OD机考2025B卷 - 最大岛屿体积（Java & Python& JS & C++ & C） -CSDN博客](#)

华为OD机考2025B卷 - 英文输入法（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

主管期望你来实现英文[输入法](#)单词联想功能。

需求如下：

- 依据用户输入的单词前缀，从已输入的英文语句中联想出用户想输入的单词，按字典序输出联想到的单词序列，
- 如果联想不到，请输出用户输入的单词前缀。

注意：

1. 英文单词联想时，区分大小写
2. 缩略形式如”don’t”，判定为两个单词，”don”和”t”
3. 输出的单词序列，不能有重复单词，且只能是英文单词，不能有标点符号

输入描述

输入为两行。

首行输入一段由英文单词word和标点符号组成的语句str；

接下来一行为一个英文单词前缀pre。

- $0 < \text{word.length()} \leq 20$
- $0 < \text{str.length} \leq 10000$
- $0 < \text{pre} \leq 20$

输出描述

输出符合要求的单词序列或单词前缀，存在多个时，单词之间以单个空格分割

示例1

输入

Plain Text |

1 I love you
2 He

输出

Plain Text |

1 He

说明

从用户已输入英文语句”I love you”中提炼出“l”、“love”、“you”三个单词，接下来用户输入“He”，从已输入信息中无法联想到任何符合要求的单词，因此输出用户输入的单词前缀。

示例2

输入

Plain Text |

1 The furthest distance in the world, Is not between life and death, But when I stand in front of you, Yet you don't know that I love you.
2 f

输出

Plain Text |

1 front furthest

说明

从用户已输入英文语句”The furthestdistance in the world, Is not between life and death, But when I stand in frontof you, Yet you dont know that I love you.”中提炼出的单词，符合“f”作为前缀的，有“furthest”和“front”，按字典序排序并在单词间添加空格后输出，结果为“front furthest”。

解题思路

简单题

```

1 import java.util.*;
2 import java.io.*;
3
4 public class Main {
5     public static void main(String[] args) throws IOException {
6         BufferedReader br = new BufferedReader(new InputStreamReader(Syste
m.in));
7         String sentence = br.readLine(); // 输入一段由英文单词word和标点符号组
成的语句
8         String prefix = br.readLine(); // 输入一个英文单词前缀
9         sentence = sentence.replaceAll("[^a-zA-Z]", " "); // 将标点符号替换
为空格
10        Set<String> wordSet = new TreeSet<>(); // 存储单词的集合，自动去重且按
照字典序排序
11        String[] words = sentence.split("\\s+");
12        for (String word : words) {
13            wordSet.add(word);
14        }
15        StringBuilder ans = new StringBuilder();
16        for (String s : wordSet) { // 遍历单词集合
17            if (s.startsWith(prefix)) { // 如果单词以前缀开头
18                ans.append(s).append(" "); // 将单词加入答案字符串
19            }
20        }
21        if (ans.length() > 0) { // 如果答案字符串不为空
22            System.out.println(ans.toString().trim()); // 输出单词序列
23        } else {
24            System.out.println(prefix); // 否则输出前缀
25        }
26    }
27}

```

Python

```
1 import string
2
3 sentence = input() # 输入一段由英文单词word和标点符号组成的语句
4 prefix = input() # 输入一个英文单词前缀
5 sentence = sentence.translate(str.maketrans(string.punctuation, ' ' * len
(string.punctuation))) # 将标点符号替换为空格
6 word_set = set(sentence.split()) # 存储单词的集合，自动去重且按照字典序排序
7 ans = ''
8 for s in sorted(word_set): # 遍历单词集合
9     if s.startswith(prefix): # 如果单词以前缀开头
10         ans += s + ' ' # 将单词加入答案字符串
11 if ans: # 如果答案字符串不为空
12     print(ans) # 输出单词序列
13 else:
14     print(prefix) # 否则输出前缀
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 rl.on('line', (sentence) => {
9     rl.on('line', (prefix) => {
10        sentence = sentence.replace(/[^w\s]/g, ' '); // 将标点符号替换为空格
11        const wordSet = new Set(sentence.split(' ')); // 存储单词的集合，自动去重
12        // 且按照字典序排序
13        let ans = '';
14        for (const word of Array.from(wordSet).sort()) { // 遍历单词集合
15            if (word.startsWith(prefix)) { // 如果单词以前缀开头
16                ans += word + ' '; // 将单词加入答案字符串
17            }
18        }
19        if (ans) { // 如果答案字符串不为空
20            console.log(ans); // 输出单词序列
21        } else {
22            console.log(prefix); // 否则输出前缀
23        }
24    });
25});
```

C++

```
1 #include <iostream>
2 #include <algorithm>
3 #include <string>
4 #include <sstream>
5 #include <set>
6
7 using namespace std;
8
9 int main() {
10     string sentence, prefix;
11     getline(cin, sentence); // 输入一段由英文单词word和标点符号组成的语句
12     getline(cin, prefix); // 输入一个英文单词前缀
13     replace_if(sentence.begin(), sentence.end(), [] (char c){return !isalpha(c);}, ' '); // 将标点符号替换为空格
14     stringstream ss(sentence);
15     set<string> word_set; // 存储单词的集合，自动去重且按照字典序排序
16     string word;
17     while (ss >> word) {
18         word_set.insert(word);
19     }
20     string ans;
21     for (auto s : word_set) { // 遍历单词集合
22         if (s.substr(0, prefix.length()) == prefix) { // 如果单词以前缀开头
23             ans += s + " "; // 将单词加入答案字符串
24         }
25     }
26     if (ans.length() > 0) { // 如果答案字符串不为空
27         cout << ans << endl; // 输出单词序列
28     } else {
29         cout << prefix << endl; // 否则输出前缀
30     }
31     return 0;
32 }
```

C语言

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <ctype.h>
4 #include <stdlib.h>
5
6 #define MAX_WORDS 1000
7 #define MAX_WORD_LENGTH 21
8 #define MAX_SENTENCE_LENGTH 10001
9
10 int compare(const void *a, const void *b) {
11     return strcmp(*(const char **)a, *(const char **)b);
12 }
13
14 int main() {
15     char sentence[MAX_SENTENCE_LENGTH], prefix[MAX_WORD_LENGTH];
16     fgets(sentence, MAX_SENTENCE_LENGTH, stdin); // 输入一段由英文单词和标点符号组成的语句
17     fgets(prefix, MAX_WORD_LENGTH, stdin); // 输入一个英文单词前缀
18
19     // 去除前缀字符串末尾的换行符
20     size_t prefix_len = strlen(prefix);
21     if (prefix[prefix_len - 1] == '\n') {
22         prefix[prefix_len - 1] = '\0';
23         prefix_len--;
24     }
25
26     // 将标点符号替换为空格
27     for (int i = 0; sentence[i] != '\0'; i++) {
28         if (!isalpha(sentence[i])) {
29             sentence[i] = ' ';
30         }
31     }
32
33     // 存储单词的数组
34     char *words[MAX_WORDS];
35     int word_count = 0;
36     char *word = strtok(sentence, " ");
37
38     // 分割单词并存储
39     while (word != NULL) {
40         words[word_count] = (char *)malloc(strlen(word) + 1);
41         strcpy(words[word_count], word);
42         word_count++;
43         word = strtok(NULL, " ");
44     }
}
```

```
45  
46     // 对单词数组进行排序  
47     qsort(words, word_count, sizeof(char *), compare);  
48  
49     // 输出结果  
50     int found = 0;  
51     for (int i = 0; i < word_count; i++) {  
52         if (strncmp(words[i], prefix, prefix_len) == 0) {  
53             printf("%s ", words[i]);  
54             found = 1;  
55         }  
56         free(words[i]); // 释放分配的内存  
57     }  
58  
59     // 如果没有找到任何匹配的单词，输出前缀  
60     if (!found) {  
61         printf("%s", prefix);  
62     }  
63  
64     return 0;  
65 }
```

完整用例

用例1

▼ Plain Text |

```
1 I love you  
2 He
```

用例2

▼ Plain Text |

```
1 The furthest distance in the world, Is not between life and death, But whe  
n I stand in front of you, Yet you don't know that I love you.  
2 f
```

用例3

▼ Plain Text |

```
1 Hello world
2 W
```

用例4

▼ Plain Text |

```
1 I am a student
2 S
```

用例5

▼ Plain Text |

```
1 This is a test
2 T
```

用例6

▼ Plain Text |

```
1 I love you
2 L
```

用例7

▼ Plain Text |

```
1 The furthest distance in the world, Is not between life and death, But whe
n I stand in front of you, Yet you don't know that I love you.
2 d
```

用例8

▼ Plain Text |

```
1 This is a test. This is only a test.
2 o
```

用例9

```
Plain Text |  
1 Hello world, how are you? I'm fine, thank you.  
2 h
```

用例10

```
Plain Text |  
1 I am a student. I study in a university.  
2 u
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [解题思路](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)
- [完整用例](#)
 - [用例1](#)
 - [用例2](#)
 - [用例3](#)
 - [用例4](#)
 - [用例5](#)
 - [用例6](#)
 - [用例7](#)
 - [用例8](#)
 - [用例9](#)
 - [用例10](#)

机考真题 华为OD

No No No



CSDN @算法大师

| 来自: 华为OD机考2025B卷 – 英文输入法 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 水仙花数 I (Java & Python& JS & C++ & C) -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

所谓水仙花数，是指一个n位的正整数，其各位数字的n次方和等于该数本身。

例如153是水仙花数，153是一个3位数，并且 $1^3 + 5^3 + 3^3 = 153$ 。

输入描述

第一行输入一个整数n，表示一个n位的正整数。n在3到7之间，包含3和7。

第二行输入一个正整数m，表示需要返回第m个水仙花数。

输出描述

返回长度是n的第m个水仙花数。个数从0开始编号。

若m大于水仙花数的个数，返回最后一个水仙花数和m的乘积。

若输入不合法，返回-1。

示例1

输入

```
1 3
2 0
```

输出

```
1 153
```

说明

| 153是第一个水仙花数

示例2

输入

▼ Plain Text |

1	9
2	1

输出

▼ Plain Text |

1	-1
---	----

说明

| 9超出范围

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         // 输入n和m
8         int n = sc.nextInt();
9         int m = sc.nextInt();
10
11        // 判断输入是否合法
12        if (n < 3 || n > 7) {
13            System.out.println("-1");
14            return;
15        }
16
17        // 计算水仙花数的范围
18        int start = (int) Math.pow(10, n - 1);
19        int end = (int) Math.pow(10, n);
20
21        // 存储水仙花数的列表
22        List<Integer> narcissusList = new ArrayList<>();
23
24        // 遍历范围内的数，判断是否为水仙花数并加入列表
25        for (int i = start; i < end; i++) {
26            if (isNarcissusNumber(i, n)) {
27                narcissusList.add(i);
28            }
29        }
30
31        // 获取水仙花数列表的长度
32        int size = narcissusList.size();
33
34        // 若列表为空，输出-1
35        if (size == 0) {
36            System.out.println("-1");
37            return;
38        }
39
40        // 输出第m个水仙花数，若m大于列表长度，则输出最后一个水仙花数乘以m
41        System.out.println(m >= size ? m * narcissusList.get(size - 1) : n
42        arcissusList.get(m));
43
44        // 判断一个数是否为水仙花数
```

```
45     public static boolean isNarcissusNumber(int num, int n) {  
46         int sum = 0;  
47         String numStr = String.valueOf(num);  
48  
49         // 计算各位数字的n次方和  
50         for (int i = 0; i < n; i++) {  
51             sum += Math.pow(Integer.parseInt(numStr.substring(i, i + 1)),  
52             n);  
53         }  
54         // 判断是否为水仙花数  
55         return sum == num;  
56     }  
57 }
```

Python

```
1 def isNarcissusNumber(num, n):
2     sum = 0
3     numStr = str(num)
4
5     for i in range(n):
6         sum += int(numStr[i]) ** n
7
8     return sum == num
9 n = int(input())
10 m = int(input())
11
12 if n < 3 or n > 7:
13     print("-1")
14     exit()
15
16 start = 10 ** (n - 1)
17 end = 10 ** n
18
19 narcissusList = []
20
21 for i in range(start, end):
22     if isNarcissusNumber(i, n):
23         narcissusList.append(i)
24
25 size = len(narcissusList)
26
27 if size == 0:
28     print("-1")
29     exit()
30
31 print(m * narcissusList[size - 1] if m >= size else narcissusList[m])
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 rl.on('line', (n) => {
9     rl.on('line', (m) => {
10        n= parseInt(n);
11        m= parseInt(m);
12        // 判断输入是否合法
13        if (n < 3 || n > 7) {
14            console.log("-1");
15            rl.close();
16            return;
17        }
18
19        // 计算水仙花数的范围
20        const start = Math.pow(10, n - 1);
21        const end = Math.pow(10, n);
22
23        // 存储水仙花数的列表
24        const narcissusList = [];
25
26        // 遍历范围内的数，判断是否为水仙花数并加入列表
27        for (let i = start; i < end; i++) {
28            if (isNarcissusNumber(i, n)) {
29                narcissusList.push(i);
30            }
31        }
32
33        // 获取水仙花数列表的长度
34        const size = narcissusList.length;
35
36        // 若列表为空，输出-1
37        if (size === 0) {
38            console.log("-1");
39            rl.close();
40            return;
41        }
42
43        // 输出第m个水仙花数，若m大于列表长度，则输出最后一个水仙花数乘以m
44        console.log(m >= size ? m * narcissusList[size - 1] : narcissusList[m]);
45
```

```
46     rl.close();
47   });
48 });
49
50
51 // 判断一个数是否为水仙花数
52 function isNarcissusNumber(num, n) {
53   let sum = 0;
54   const numStr = String(num);
55
56   // 计算各位数字的n次方和
57   for (let i = 0; i < n; i++) {
58     sum += Math.pow(parseInt(numStr.substring(i, i + 1)), n);
59   }
60
61   // 判断是否为水仙花数
62   return sum === num;
63 }
```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 using namespace std;
5
6 bool isNarcissusNumber(int num, int n) {
7     int sum = 0;
8     string numStr = to_string(num);
9
10    for (int i = 0; i < n; i++) {
11        sum += pow(stoi(numStr.substr(i, 1)), n);
12    }
13
14    return sum == num;
15 }
16
17 int main() {
18     int n, m;
19     cin >> n >> m;
20
21     if (n < 3 || n > 7) {
22         cout << "-1" << endl;
23         return 0;
24     }
25
26     int start = pow(10, n - 1);
27     int end = pow(10, n);
28
29     vector<int> narcissusList;
30
31     for (int i = start; i < end; i++) {
32         if (isNarcissusNumber(i, n)) {
33             narcissusList.push_back(i);
34         }
35     }
36
37     int size = narcissusList.size();
38
39     if (size == 0) {
40         cout << "-1" << endl;
41         return 0;
42     }
43
44     cout << (m >= size ? m * narcissusList[size - 1] : narcissusList[m]) <
< endl;
```

```
45     return 0;
46 }
47 }
```

C语言

```
1 #include <stdio.h>
2 #include <math.h>
3 #include <stdlib.h>
4
5 // 判断一个数是否为水仙花数的函数声明
6 int isNarcissusNumber(int num, int n);
7
8 int main() {
9     int n, m;
10
11     // 输入n和m
12     if (scanf("%d %d", &n, &m) != 2) {
13         printf("-1\n");
14         return -1;
15     }
16
17     // 判断输入是否合法, n必须在3到7之间
18     if (n < 3 || n > 7) {
19         printf("-1\n");
20         return -1;
21     }
22
23     // 计算n位数的范围
24     int start = pow(10, n - 1); // n位数的起始值, 例如3位数从100开始
25     int end = pow(10, n); // n位数的结束值, 例如3位数到999结束
26
27     // 存储水仙花数的数组, 最大长度为end - start
28     int narcissusList[end - start];
29     int count = 0; // 用于记录找到的水仙花数数量
30
31     // 遍历范围内的数, 判断是否为水仙花数
32     for (int i = start; i < end; i++) {
33         if (isNarcissusNumber(i, n)) {
34             narcissusList[count++] = i; // 将水仙花数加入数组
35         }
36     }
37
38     // 若没有找到任何水仙花数, 输出-1
39     if (count == 0) {
40         printf("-1\n");
41         return 0;
42     }
43
44     // 判断m的值是否超出找到的水仙花数的数量
45     if (m >= count) {
```

```

46         // m大于或等于水仙花数的数量时，返回最后一个水仙花数乘以m
47         printf("%ld\n", narcissusList[count - 1] * m);
48     } else {
49         // 否则，返回第m个水仙花数
50         printf("%ld\n", narcissusList[m]);
51     }
52
53     return 0;
54 }
55
56 // 判断一个数是否为水仙花数
57 int isNarcissusNumber(int num, int n) {
58     int sum = 0;           // 用于存储各位数字的n次方和
59     int original_num = num; // 保留原始数值用于最后比较
60
61     // 逐位提取数字并计算n次方和
62     while (num > 0) {
63         int digit = num % 10;           // 获取当前数的最后一位数字
64         sum += pow(digit, n);          // 计算该数字的n次方并加到总和中
65         num /= 10;                    // 移除最后一位数字，继续处理剩下的数字
66     }
67
68     // 若n次方和等于原始数值，则该数为水仙花数
69     return sum == original_num;
70 }
```

完整用例

用例1

3
0

用例2

4
1

用例3

5
2

用例4

3
10

用例5

4

5

用例6

5

0

用例7

4

2

用例8

4

7

用例9

4

4

用例10

5

10

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)
 - [完整用例](#)
 - [用例1](#)
 - [用例2](#)
 - [用例3](#)
 - [用例4](#)
 - [用例5](#)
 - [用例6](#)

- 用例7
- 用例8
- 用例9
- 用例10



| 来自: 华为OD机考2025B卷 – 水仙花数 | (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - VLAN资源池（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

```
▼ Plain Text |  
1 2025华为od 机试2025B卷-华为机考OD2025年B卷
```

题目描述

VLAN是一种对局域网设备进行逻辑划分的技术，为了标识不同的VLAN，引入VLAN ID(1-4094之间的整数)的概念。

定义一个VLAN ID的资源池(下称VLAN资源池)，资源池中连续的VLAN用开始VLAN-结束VLAN表示，不连续的用单个整数表示，所有的VLAN用英文逗号连接起来。

现在有一个VLAN资源池，业务需要从资源池中申请一个VLAN，需要你输出从VLAN资源池中移除申请的VLAN后的资源池。

输入描述

第一行为字符串格式的VLAN资源池，第二行为业务要申请的VLAN，VLAN的取值范围为[1,4094]之间的整数。

输出描述

从输入VLAN资源池中移除申请的VLAN后字符串格式的VLAN资源池，输出要求满足题目描述中的格式，并且按照VLAN从小到大升序输出。

如果申请的VLAN不在原VLAN资源池内，输出原VLAN资源池升序排序后的字符串即可。

示例1

输入

```
▼ Plain Text |  
1 1-5  
2 2
```

输出

▼ Plain Text |

```
1 1,3-5
```

说明

原VLAN资源池中有VLAN 1、2、3、4、5，从资源池中移除2后，剩下VLAN 1、3、4、5，按照题目描述格式并升序后的结果为1,3-5

示例2

输入

▼ Plain Text |

```
1 20-21,15,18,30,5-10
2 15
```

输出

▼ Plain Text |

```
1 5-10,18,20-21,30
```

说明

原VLAN资源池中有VLAN 5、6、7、8、9、10、15、18、20、21、30，从资源池中移除15后，资源池中剩下的VLAN为 5、6、7、8、9、10、18、20、21、30，按照题目描述格式并升序后的结果为5–10,18,20–21,30。

示例3

输入

▼ Plain Text |

```
1 5,1-3
2 10
```

输出

▼

Plain Text |

1 1-3,5

说明

原VLAN资源池中有VLAN 1、2、3，5，申请的VLAN 10不在原资源池中，将原资源池按照题目描述格式并按升序排序后输出的结果为1-3,5。

解题思路

- VLAN（虚拟局域网）是一种网络技术，用来对局域网中的设备进行逻辑划分。每个VLAN通过一个 VLAN ID 来标识，取值范围是 1 到 4094 的整数。题目中的任务是处理一个字符串格式的 VLAN 资源池，模拟从资源池中申请并移除某个 VLAN，然后返回剩余的资源池。

任务：

- a. 输入

包含两个部分：

- VLAN资源池：用字符串表示，可能是单个 VLAN ID，或者是多个 VLAN ID 或 VLAN ID 范围（用“-”连接），各个 VLAN 或范围之间用逗号连接。
- 要申请的 VLAN：一个需要移除的 VLAN ID。

- b. 输出

:

- 从资源池中移除申请的 VLAN 后，输出剩余的 VLAN 资源池，按从小到大的顺序排列，且格式必须符合题目的描述。

输出格式：

- 连续的 VLAN ID 应该用范围的方式表示，如 1-5 表示 VLAN 1, 2, 3, 4, 5。
- 不连续的 VLAN ID 用逗号分隔，如 1, 3-5。

示例分析：

示例 1：

输入：

▼

Plain Text |

1 1-5
2 2

解释：

- 原始资源池有 VLAN 1, 2, 3, 4, 5。
- 申请的 VLAN 是 2，所以移除 2 后，剩下的 VLAN 是 1, 3, 4, 5。

- 格式化输出为: `1,3-5`。

示例 2:

输入:

```
▼ Plain Text |  
1 20-21,15,18,30,5-10  
2 15
```

解释:

- 原始资源池有 VLAN 5, 6, 7, 8, 9, 10, 15, 18, 20, 21, 30。
- 申请的 VLAN 是 15, 移除 15 后, 剩下的 VLAN 是 5, 6, 7, 8, 9, 10, 18, 20, 21, 30。
- 格式化输出为: `5-10,18,20-21,30`。

示例 3:

输入:

```
▼ Plain Text |  
1 5,1-3  
2 10
```

解释:

- 原始资源池有 VLAN 1, 2, 3, 5。
- 申请的 VLAN 是 10, 不在资源池中, 所以资源池保持不变。
- 格式化输出为: `1-3,5`。

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         // 输入VLAN资源池
8         String input = sc.nextLine();
9         // 输入业务要申请的VLAN
10        Integer destVlan = Integer.parseInt(sc.nextLine());
11
12        // 解析VLAN资源池
13        List<Integer> vlanPool = parseVlanPool(input);
14
15        // 对VLAN资源池进行升序排序
16        Collections.sort(vlanPool);
17
18        // 从VLAN资源池中移除申请的VLAN
19        vlanPool.remove(destVlan);
20
21        // 格式化VLAN资源池
22        String result = formatVlanPool(vlanPool);
23        System.out.println(result);
24    }
25
26    // 解析VLAN资源池
27    private static List<Integer> parseVlanPool(String input) {
28        List<Integer> vlanPool = new ArrayList<Integer>();
29        // 根据逗号分割VLAN资源池中的VLAN
30        String[] vlanGroup = input.split(",");
31        for (String vlanItem : vlanGroup) {
32            if (vlanItem.contains("-")) {
33                // 如果VLAN是连续的，根据连字符分割开始VLAN和结束VLAN
34                String[] vlanItems = vlanItem.split("-");
35                Integer start = Integer.parseInt(vlanItems[0]);
36                Integer end = Integer.parseInt(vlanItems[1]);
37                // 将连续的VLAN添加到VLAN资源池中
38                for (int j = start; j <= end; j++) {
39                    vlanPool.add(j);
40                }
41            } else {
42                // 如果VLAN是单个的，直接添加到VLAN资源池中
43                vlanPool.add(Integer.parseInt(vlanItem));
44            }
45        }
46    }
47}
```

```

46         return vlanPool;
47     }
48
49     // 格式化VLAN资源池
50     private static String formatVlanPool(List<Integer> vlanPool) {
51         StringBuilder result = new StringBuilder();
52         Integer last = null;
53         for (int index = 0; index < vlanPool.size(); index++) {
54             if (last == null) {
55                 // 如果是第一个VLAN, 直接添加到结果中
56                 result.append(vlanPool.get(index));
57                 last = vlanPool.get(index);
58             } else {
59                 if (vlanPool.get(index) - last == 1) {
60                     // 如果与上一个VLAN相差1, 表示是连续的VLAN
61                     if (result.toString().endsWith("-" + last)) {
62                         // 如果结果中最后一个VLAN已经是连续的VLAN的结束VLAN, 替换
63                         // 为当前VLAN
64                         result.replace(result.lastIndexOf(last.toString())
65                         , result.length(), vlanPool.get(index).toString());
66                     } else {
67                         // 否则添加连字符和当前VLAN
68                         result.append("-").append(vlanPool.get(index));
69                     }
70                 } else {
71                     // 如果与上一个VLAN不连续, 直接添加逗号和当前VLAN
72                     result.append(",").append(vlanPool.get(index));
73                 }
74             }
75         }
76     }
77 }

```

Python

```
1 import sys
2
3 # 输入VLAN资源池
4 vlan_pool_input = input()
5 # 输入业务要申请的VLAN
6 dest_vlan = int(input())
7
8 # 定义存储VLAN的列表
9 vlan_pool = []
10
11 # 将输入的VLAN资源池按逗号分隔为多个VLAN组
12 vlan_group = vlan_pool_input.split(",")
13
14 # 遍历每个VLAN组
15 for vlan_item in vlan_group:
16     # 如果VLAN组中包含连续的VLAN
17     if "--" in vlan_item:
18         # 将连续的VLAN拆分为开始VLAN和结束VLAN
19         vlan_items = vlan_item.split("-")
20         start_vlan = int(vlan_items[0])
21         end_vlan = int(vlan_items[1])
22         # 将连续的VLAN添加到VLAN资源池中
23         for j in range(start_vlan, end_vlan + 1):
24             vlan_pool.append(j)
25         continue
26     # 如果VLAN组中只有一个VLAN
27     vlan_pool.append(int(vlan_item))
28
29 # 对VLAN资源池进行升序排序
30 vlan_pool.sort()
31
32 # 如果申请的VLAN在VLAN资源池中
33 if dest_vlan in vlan_pool:
34     # 从VLAN资源池中移除申请的VLAN
35     vlan_pool.remove(dest_vlan)
36
37 # 定义存储结果的列表
38 result = []
39 # 定义上一个VLAN的变量
40 last_vlan = None
41
42 # 遍历VLAN资源池中的每个VLAN
43 for index in range(len(vlan_pool)):
44     # 如果是第一个VLAN
45     if last_vlan is None:
```

```
46     result.append(str(vlan_pool[index]))
47     last_vlan = vlan_pool[index]
48     continue
49     # 如果当前VLAN与上一个VLAN连续
50     if vlan_pool[index] - last_vlan == 1:
51         # 如果结果列表中的最后一个元素以"-上一个VLAN"结尾
52         if result[-1].endswith("-" + str(last_vlan)):
53             # 将结果列表中的最后一个元素更新为"-当前VLAN"
54             result[-1] = result[-1][:result[-1].rindex(str(last_vlan))] +
55             str(vlan_pool[index])
56         else:
57             # 在结果列表中添加"-当前VLAN"
58             result.append("-" + str(vlan_pool[index]))
59     else:
60         # 在结果列表中添加",当前VLAN"
61         result.append(", " + str(vlan_pool[index]))
62     last_vlan = vlan_pool[index]
63     # 输出结果列表中的VLAN资源池
64     print("".join(result))
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  // 输入VLAN资源池
9  rl.on('line', (vlan_pool_input) => {
10     // 输入业务要申请的VLAN
11     rl.on('line', (dest_vlan_input) => {
12         // 关闭读取接口
13         rl.close();
14
15         const dest_vlan = parseInt(dest_vlan_input);
16
17         // 定义存储VLAN的列表
18         const vlan_pool = [];
19
20         // 将输入的VLAN资源池按逗号分隔为多个VLAN组
21         const vlan_group = vlan_pool_input.split(",");
22
23         // 遍历每个VLAN组
24         for (let vlan_item of vlan_group) {
25             // 如果VLAN组中包含连续的VLAN
26             if (vlan_item.includes("-")) {
27                 // 将连续的VLAN拆分为开始VLAN和结束VLAN
28                 const vlan_items = vlan_item.split("-");
29                 const start_vlan = parseInt(vlan_items[0]);
30                 const end_vlan = parseInt(vlan_items[1]);
31                 // 将连续的VLAN添加到VLAN资源池中
32                 for (let j = start_vlan; j <= end_vlan; j++) {
33                     vlan_pool.push(j);
34                 }
35                 continue;
36             }
37             // 如果VLAN组中只有一个VLAN
38             vlan_pool.push(parseInt(vlan_item));
39         }
40
41         // 对VLAN资源池进行升序排序
42         vlan_pool.sort((a, b) => a - b);
43
44         // 如果申请的VLAN在VLAN资源池中
45         if (vlan_pool.includes(dest_vlan)) {
```

```

46     // 从VLAN资源池中移除申请的VLAN
47     vlan_pool.splice(vlan_pool.indexOf(dest_vlan), 1);
48 }
49
50     // 定义存储结果的列表
51     const result = [];
52     // 定义上一个VLAN的变量
53     let last_vlan = null;
54
55     // 遍历VLAN资源池中的每个VLAN
56     for (let index = 0; index < vlan_pool.length; index++) {
57         // 如果是第一个VLAN
58         if (last_vlan === null) {
59             result.push(vlan_pool[index].toString());
60             last_vlan = vlan_pool[index];
61             continue;
62         }
63         // 如果当前VLAN与上一个VLAN连续
64         if (vlan_pool[index] - last_vlan === 1) {
65             // 如果结果列表中的最后一个元素以"-上一个VLAN"结尾
66             if (result[result.length - 1].endsWith("-" + last_vlan)) {
67                 // 将结果列表中的最后一个元素更新为"-当前VLAN"
68                 result[result.length - 1] = result[result.length - 1].slice(0, r
esult[result.length - 1].lastIndexOf(last_vlan.toString())) + vlan_pool[in
dex].toString();
69             } else {
70                 // 在结果列表中添加"-当前VLAN"
71                 result.push("-" + vlan_pool[index].toString());
72             }
73         } else {
74             // 在结果列表中添加",当前VLAN"
75             result.push("," + vlan_pool[index].toString());
76         }
77         last_vlan = vlan_pool[index];
78     }
79
80     // 输出结果列表中的VLAN资源池
81     console.log(result.join(""));
82 });
83 });

```

C++

```

1 #include <iostream>
2 #include <vector>
3 #include <sstream>
4 #include <algorithm>
5
6 using namespace std;
7 int main() {
8     string vlan_pool_input; // 存储输入的VLAN资源池的字符串
9     getline(cin, vlan_pool_input); // 获取输入的VLAN资源池的字符串
10    int dest_vlan; // 存储业务要申请的VLAN
11    cin >> dest_vlan; // 获取业务要申请的VLAN
12
13    vector<int> vlan_pool; // 存储VLAN资源池中的VLAN
14    stringstream ss(vlan_pool_input); // 使用字符串流解析VLAN资源池的字符串
15    string vlan_item; // 存储解析出的每个VLAN
16    while (getline(ss, vlan_item, ',')) { // 按逗号分隔字符串, 获取每个VLAN
17        if (vlan_item.find('-') != string::npos) { // 如果VLAN是连续的范围
18            stringstream range_ss(vlan_item); // 使用字符串流解析连续范围的字符串
19            string start_vlan_str, end_vlan_str; // 存储连续范围的起始VLAN和结束VLAN
20            getline(range_ss, start_vlan_str, '-'); // 获取起始VLAN
21            getline(range_ss, end_vlan_str, '-'); // 获取结束VLAN
22            int start_vlan = stoi(start_vlan_str); // 将起始VLAN转换为整数
23            int end_vlan = stoi(end_vlan_str); // 将结束VLAN转换为整数
24            for (int j = start_vlan; j <= end_vlan; j++) { // 将连续范围内的VLAN添加到VLAN资源池中
25                vlan_pool.push_back(j);
26            }
27        } else { // 如果VLAN是单个整数
28            vlan_pool.push_back(stoi(vlan_item)); // 将VLAN转换为整数并添加到VLAN资源池中
29        }
30    }
31
32    sort(vlan_pool.begin(), vlan_pool.end()); // 对VLAN资源池中的VLAN进行排序
33
34    auto it = find(vlan_pool.begin(), vlan_pool.end(), dest_vlan); // 查找业务要申请的VLAN在VLAN资源池中的位置
35    if (it != vlan_pool.end()) { // 如果找到了业务要申请的VLAN
36        vlan_pool.erase(it); // 从VLAN资源池中移除业务要申请的VLAN
37    }
38
39    vector<string> result; // 存储最终输出结果的字符串向量
40    int last_vlan = -1; // 存储上一个输出的VLAN

```

```

41
42     for (int i = 0; i < vlan_pool.size(); i++) { // 遍历VLAN资源池中的VLAN
43         if (last_vlan == -1) { // 如果是第一个输出的VLAN
44             result.push_back(to_string(vlan_pool[i])); // 将VLAN转换为字符串
45             并添加到结果向量中
46             last_vlan = vlan_pool[i]; // 更新上一个输出的VLAN
47             continue; // 继续下一次循环
48         }
49         if (vlan_pool[i] - last_vlan == 1) { // 如果当前VLAN与上一个输出的VLA
50             N相差1
51                 if (result.back().find('-' + to_string(last_vlan)) != string::
52            npos) { // 如果结果向量的最后一个字符串包含连续范围的结束VLAN
53                 result.back() = result.back().substr(0, result.back().rfin
54             d(to_string(last_vlan))) + to_string(vlan_pool[i]); // 更新连续范围的结束VLAN
55             } else { // 如果结果向量的最后一个字符串不包含连续范围的结束VLAN
56                 result.push_back("-" + to_string(vlan_pool[i])); // 在结果
57             向量中添加连续范围的结束VLAN
58             }
59         }
60         if (last_vlan == vlan_pool[i]) { // 如果当前VLAN与上一个输出的VLAN不相差1
61             result.push_back(",," + to_string(vlan_pool[i])); // 在结果向量中
62             添加当前VLAN
63         }
64     }
65 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 // 定义最大VLAN池的大小
6 #define MAX_VLAN 4096
7
8 // 函数声明
9 void parseAndFilterVlanPool(char *input, int *vlanPool, int *size, int de
10 stVlan);
11 void formatVlanPool(int *vlanPool, int size, char *result);
12
13 int main() {
14     // 存储输入的VLAN资源池和业务申请的VLAN
15     char input[1000];
16     int destVlan;
17
18     fgets(input, sizeof(input), stdin);
19     input[strcspn(input, "\n")] = 0; // 去除换行符
20
21     scanf("%d", &destVlan);
22
23     // 定义VLAN池数组，用于存储解析后的VLAN，最大为MAX_VLAN
24     int vlanPool[MAX_VLAN];
25     int size = 0;
26
27     // 解析并过滤VLAN资源池（在解析过程中自动移除目标VLAN）
28     parseAndFilterVlanPool(input, vlanPool, &size, destVlan);
29
30     // 定义用于格式化输出的字符串
31     char result[1000] = "";
32
33     // 格式化VLAN资源池
34     formatVlanPool(vlanPool, size, result);
35
36     // 输出结果
37     printf("%s\n", result);
38
39     return 0;
40 }
41
42 void parseAndFilterVlanPool(char *input, int *vlanPool, int *size, int de
43 stVlan) {
        char *token = strtok(input, ","); // 使用逗号分割字符串
```

```

44
45 // 解析每个VLAN或VLAN范围
46 while (token != NULL) {
47     if (strchr(token, '-')) {
48         // 如果包含连字符"-", 则为VLAN范围
49         int start, end;
50         sscanf(token, "%d-%d", &start, &end); // 解析范围的起始和结束VL
      AN
51
52         // 将范围内的VLAN逐个加入VLAN池, 同时移除目标VLAN
53         for (int i = start; i <= end; i++) {
54             if (i == destVlan) {
55                 continue; // 跳过目标VLAN
56             }
57             // 将VLAN加入有序数组
58             int j;
59             for (j = *size - 1; j >= 0 && vlanPool[j] > i; j--) {
60                 vlanPool[j + 1] = vlanPool[j];
61             }
62             vlanPool[j + 1] = i;
63             (*size)++;
64         }
65     } else {
66         // 如果是单个VLAN, 直接处理
67         int vlan = atoi(token);
68         if (vlan == destVlan) {
69             token = strtok(NULL, ",");
70             continue; // 跳过目标VLAN
71         }
72         // 将VLAN插入到有序数组中
73         int j;
74         for (j = *size - 1; j >= 0 && vlanPool[j] > vlan; j--) {
75             vlanPool[j + 1] = vlanPool[j];
76         }
77         vlanPool[j + 1] = vlan;
78         (*size)++;
79     }
80     // 获取下一个VLAN或VLAN范围
81     token = strtok(NULL, ",");
82 }
83 }
84
85 // 格式化VLAN池为要求的字符串格式
86 void formatVlanPool(int *vlanPool, int size, char *result) {
87     int last = -1; // 上一个处理的VLAN
88     int start = -1; // 范围的起始VLAN
89
90     for (int i = 0; i < size; i++) {

```

```

91         if (last == -1) {
92             // 第一个VLAN直接添加
93             start = vlanPool[i];
94             last = vlanPool[i];
95         } else if (vlanPool[i] == last + 1) {
96             // 如果当前VLAN与上一个连续, 继续处理
97             last = vlanPool[i];
98         } else {
99             // 如果不连续, 检查是否是一个范围
100            if (start == last) {
101                // 如果是单个VLAN, 直接添加
102                char temp[20];
103                sprintf(temp, "%d,", start);
104                strcat(result, temp);
105            } else {
106                // 否则为范围, 添加"start-end"格式
107                char temp[40];
108                sprintf(temp, "%d-%d,", start, last);
109                strcat(result, temp);
110            }
111            // 处理下一个VLAN范围
112            start = vlanPool[i];
113            last = vlanPool[i];
114        }
115    }
116
117    // 处理最后一个范围或VLAN
118    if (start == last) {
119        char temp[20];
120        sprintf(temp, "%d", start);
121        strcat(result, temp);
122    } else {
123        char temp[40];
124        sprintf(temp, "%d-%d", start, last);
125        strcat(result, temp);
126    }
127
128    // 去除最后的逗号
129    if (result[strlen(result) - 1] == ',') {
130        result[strlen(result) - 1] = '\0';
131    }
132}

```

完整用例

用例1

Plain Text |

1 1-5
2 2

用例2

Plain Text |

1 20-21,15,18,30,5-10
2 15

用例3

Plain Text |

1 5,1-3
2 10

用例4

Plain Text |

1 1-3,5,100,200,300,400,500
2 1

用例5

Plain Text |

1 1-3,5,100,200,300,400,500
2 5

用例6

Plain Text |

1 1-3,5,100,200,300,400,500
2 100

用例7

▼

Plain Text |

```
1 1-3,5  
2 3
```

用例8

▼

Plain Text |

```
1 20-21,15,18,30,5-10  
2 15
```

用例9

▼

Plain Text |

```
1 1-3,5,100,200,300,400,500  
2 100
```

用例10

▼

Plain Text |

```
1 1-3,5,100,200,300,400,500  
2 200
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [示例3](#)
- [解题思路](#)
 - [任务：](#)
 - [输出格式：](#)
 - [示例分析：](#)
 - [示例 1：](#)
 - [示例 2：](#)
 - [示例 3：](#)

- Java
 - Python
 - JavaScript
 - C++
 - C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10



| 来自: 华为OD机考2025B卷 – VLAN资源池 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 简单的自动曝光/平均像素（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录： [点击查看目录](#)

华为OD面试真题精选： [点击立即查看](#)

题目描述

一个图像有n个**像素点**，存储在一个长度为n的数组img里，每个像素点的取值范围[0,255]的正整数。

请你给图像每个像素点值加上一个整数k（可以是负数），得到新图newImg，使得新图newImg的所有像素平均值最接近中位值128。

请输出这个整数k。

输入描述

n个整数，中间用空格分开

备注

- $1 \leq n \leq 100$
- 如有多个整数k都满足，输出小的那个k；
- 新图的像素值会自动截取到[0,255]范围。当新像素值<0，其值会更改为0；当新像素值>255，其值会更改为255；

例如newImg=" -1 -2 256",会自动更改为"0 0 255"

输出描述

一个整数k

示例1

输入

```
▼ Plain Text |  
1 129 130 129 130
```

输出

Plain Text |

```
1 -2
```

说明

-1的均值128.5，-2的均值为127.5，输出较小的数-2

示例2

输入

Plain Text |

```
1 0 0 0 0
```

输出

Plain Text |

```
1 128
```

说明

四个像素值都为0

解题思路

代码思路

本题的解题思路是通过枚举每一个可能的 k 值，计算新图像的每个像素点的值，并找出使得新图像的平均值与中位值 128 的差的绝对值最小的 k 值。

具体实现步骤如下：

1. 枚举每一个可能的 k k k 值：

- 计算新图像的每个像素点的值。
- 将这些像素点的值累加起来，得到新图像的所有像素点值的和 `sum`。

2. 计算平均值和差的绝对值：

- 计算新图像的平均值 `sum / len`。
- 计算新图像的平均值与中位值 128 之间的差的绝对值 `diff`。

3. 更新最优解：

- 如果 `diff` 小于 `min_diff`，更新 `min_diff` 为当前的 `diff`，并更新 `k_ans` 为当前的 k 值。
- 如果 `diff` 等于 `min_diff` 且 `k_ans` 不等于 0，则更新 `k_ans` 为 k 和 `k_ans` 中的较小值。

注意事项

对于每一个像素点的新值，需要确保其在 [0, 255] 的范围内：

- 如果新值小于 0，则将其设为 0。
- 如果新值大于 255，则将其设为 255。
- 如果有多个整数 k 满足条件，输出较小的那个 k 。

```

1 import java.util.Scanner; // 导入Scanner类
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in); // 创建Scanner对象
6         String input_str = sc.nextLine(); // 读取一行输入
7         Scanner ss = new Scanner(input_str); // 将输入转换为Scanner对象
8         int val, len = 0;
9         int[] img = new int[110];
10        while (ss.hasNextInt()) { // 判断Scanner对象中是否还有整数
11            val = ss.nextInt(); // 读取下一个整数
12            img[len++] = val;
13        }
14        double min_diff = Integer.MAX_VALUE; // 定义双精度浮点型变量min_diff，并初始化为整型最大值
15        int k_ans = 0; // 定义整型变量k_ans，并初始化为0
16
17        for (int k = -127; k <= 128; k++) { // 循环k的值从-127到128
18            double sum = 0; // 定义双精度浮点型变量sum，并初始化为0
19            for (int i = 0; i < len; i++) { // 循环i的值从0到len-1
20                int new_val = img[i] + k; // 定义整型变量new_val，值为img[i]+k
21                new_val = Math.max(0, Math.min(new_val, 255)); // 将new_val的值限制在0到255之间
22                sum += new_val; // 将new_val的值加入sum中
23            }
24
25            double diff = Math.abs(sum / len - 128); // 计算sum/len-128的绝对值
26
27            if (diff < min_diff) { // 如果diff小于min_diff
28                min_diff = diff; // 将min_diff的值更新为diff
29                k_ans = k; // 将k_ans的值更新为k
30            } else if (diff == min_diff && k_ans != 0) { // 如果diff等于min_diff且k_ans不等于0
31                k_ans = Math.min(k_ans, k); // 将k_ans的值更新为k和k_ans中的最小值
32            }
33        }
34
35        System.out.println(k_ans); // 输出k_ans的值，并换行
36    }
37}

```

Python

```
Plain Text | ▾  
1 import sys  
2 input_str = sys.stdin.readline().strip()  
3 img = list(map(int, input_str.split()))  
4 len = len(img)  
5 min_diff = sys.maxsize  
6 k_ans = 0  
7  
8 for k in range(-127, 129):  
9     sum = 0  
10    for i in range(len):  
11        new_val = img[i] + k  
12        new_val = max(0, min(new_val, 255))  
13        sum += new_val  
14    diff = abs(sum / len - 128)  
15    if diff < min_diff:  
16        min_diff = diff  
17        k_ans = k  
18    elif diff == min_diff and k_ans != 0:  
19        k_ans = min(k_ans, k)  
20  
21 print(k_ans)
```

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_str = '';
9
10 // 监听输入，将输入存入input_str中
11 rl.on('line', (line) => {
12     input_str += line + ' ';
13 }).on('close', () => {
14     const img = [];
15     const input_arr = input_str.trim().split(' ');
16     const len = input_arr.length;
17
18     // 将输入的字符串转为数字并存入数组img中
19     for (let i = 0; i < len; i++) {
20         img.push(parseInt(input_arr[i]));
21     }
22
23     // 初始化最小差值和答案k
24     let min_diff = Number.MAX_SAFE_INTEGER;
25     let k_ans = 0;
26
27     // 枚举k的取值范围
28     for (let k = -127; k <= 128; k++) {
29         let sum = 0;
30
31         // 计算新图的所有像素点的值之和
32         for (let i = 0; i < len; i++) {
33             let new_val = img[i] + k;
34             // 将新像素值截取到[0,255]范围
35             new_val = Math.max(0, Math.min(new_val, 255));
36             sum += new_val;
37         }
38
39         // 计算新图的所有像素点的平均值与中位值128的差值
40         const diff = Math.abs(sum / len - 128);
41
42         // 更新最小差值和答案k
43         if (diff < min_diff) {
44             min_diff = diff;
45             k_ans = k;
46         }
47     }
48
49     // 打印结果
50     console.log(`最小差值: ${min_diff}, 答案k: ${k_ans}`);
51
52     // 退出程序
53     process.exit();
54 }
55 ).on('close', () => {
56     process.exit();
57 })
```

```
46     } else if (diff === min_diff && k_ans !== 0) {
47         k_ans = Math.min(k_ans, k);
48     }
49 }
50
51 // 输出答案k
52 console.log(k_ans);
53});
```

C++

```

1 #include <iostream> // 标准输入输出流
2 #include <algorithm>
3 #include <string>
4 #include <sstream>
5 #include <cmath>
6 #include <climits>
7
8 using namespace std;
9
10 int main() {
11     string input_str;
12     getline(cin, input_str);
13     stringstream ss(input_str);
14     int val, len = 0;
15     int img[110];
16     while (ss >> val) {
17         img[len++] = val;
18     }
19     double min_diff = INT_MAX; // 定义双精度浮点型变量min_diff，并初始化为整型最大值
20     int k_ans = 0; // 定义整型变量k_ans，并初始化为0
21
22     for (int k = -127; k <= 128; k++) { // 循环k的值从-127到128
23         double sum = 0; // 定义双精度浮点型变量sum，并初始化为0
24         for (int i = 0; i < len; i++) { // 循环i的值从0到len-1
25             int new_val = img[i] + k; // 定义整型变量new_val，值为img[i]+k
26             new_val = max(0, min(new_val, 255)); // 将new_val的值限制在0到255之间
27             sum += new_val; // 将new_val的值加入sum中
28         }
29
30         double diff = abs(sum / len - 128); // 计算sum/len-128的绝对值
31
32         if (diff < min_diff) { // 如果diff小于min_diff
33             min_diff = diff; // 将min_diff的值更新为diff
34             k_ans = k; // 将k_ans的值更新为k
35         }
36         else if (diff == min_diff && k_ans != 0) { // 如果diff等于min_diff且k_ans不等于0
37             k_ans = min(k_ans, k); // 将k_ans的值更新为k和k_ans中的最小值
38         }
39     }
40
41     cout << k_ans << endl; // 输出k_ans的值，并换行
42 }
```

```
43     return 0; // 返回0，表示程序正常结束
44 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <limits.h>
4 #include <math.h>
5
6 int main() {
7     char input_str[1000]; // 定义字符数组用于存储输入的字符串
8     fgets(input_str, 1000, stdin); // 读取一行输入并存储在input_str中
9
10    int val, len = 0; // 定义整型变量val和len, len用于记录数组长度
11    int img[110]; // 定义整型数组img, 用于存储输入的像素值
12
13    // 用于解析输入字符串中的整数
14    char *token = strtok(input_str, " ");
15    while (token != NULL) { // 当token不为NULL时继续循环
16        val = atoi(token); // 将token转换为整数
17        img[len++] = val; // 将整数存入数组img中, 并增加数组长度len
18        token = strtok(NULL, " "); // 获取下一个token
19    }
20
21    double min_diff = INT_MAX; // 定义双精度浮点型变量min_diff, 并初始化为整型最大值
22    int k_ans = 0; // 定义整型变量k_ans, 并初始化为0
23
24    // 枚举k值从-127到128
25    for (int k = -127; k <= 128; k++) {
26        double sum = 0; // 定义双精度浮点型变量sum, 并初始化为0
27
28        // 遍历img数组中的每一个像素值
29        for (int i = 0; i < len; i++) {
30            int new_val = img[i] + k; // 定义整型变量new_val, 值为img[i]+k
31
32            // 将new_val的值限制在0到255之间
33            if (new_val < 0) {
34                new_val = 0; // 如果new_val小于0, 则将new_val设为0
35            } else if (new_val > 255) {
36                new_val = 255; // 如果new_val大于255, 则将new_val设为255
37            }
38
39            sum += new_val; // 将new_val的值加入sum中
40        }
41
42        // 计算新图像的平均值与中位值128的差的绝对值
43        double diff = fabs(sum / len - 128);
44    }
```

```
45     // 如果当前diff小于min_diff, 更新min_diff和k_ans
46     if (diff < min_diff) {
47         min_diff = diff;
48         k_ans = k;
49     }
50     // 如果当前diff等于min_diff且k_ans不等于0, 更新k_ans为k和k_ans中的较小值
51     else if (diff == min_diff && k_ans != 0) {
52         k_ans = (k_ans < k) ? k_ans : k;
53     }
54 }
55
56     // 输出最终的k_ans值
57     printf("%d\n", k_ans);
58
59     return 0;
60 }
```

完整用例

用例1

▼ Plain Text |
1 120 130 140 150

用例2

▼ Plain Text |
1 50 50 50 50

用例3

▼ Plain Text |
1 255 255 255 255

用例4

▼ Plain Text |
1 100 100 100 100

用例5

```
▼ Plain Text |  
1 128 128 128 128 128
```

用例6

```
▼ Plain Text |  
1 10 20 30 40 50 60 70 80 90 100
```

用例7

```
▼ Plain Text |  
1 1 2 3 4 5 6 7 8 9 10
```

用例8

```
▼ Plain Text |  
1 60 70 80 90
```

用例9

```
▼ Plain Text |  
1 250 240 230 220
```

用例10

```
▼ Plain Text |  
1 133 150 120 140
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)

- 备注
- 输出描述
- 示例1
- 示例2
- 解题思路
 - 代码思路
 - 注意事项
- Java
- Python
- JavaScript
- C++
- C语言
 - 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10



来自: 华为OD机考2025B卷 – 简单的自动曝光/平均像素 (Java & Python& JS & C++ & C) -CSDN
博客

华为OD机考2025B卷 - 用户调度问题（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

题目描述

在[通信系统](#)中，一个常见的问题是用户进行不同策略的调度，会得到不同的系统消耗和性能。

假设当前有n个待串行调度用户，每个用户可以使用A/B/C三种[不同的](#)调度策略，不同的策略会消耗不同的系统资源。请你根据如下规则进行用户调度，并返回总的消耗资源数。

规则：

1. 相邻的用户不能使用相同的调度策略，例如，第1个用户使用了A策略，则第2个用户只能使用B或者C策略。
2. 对单个用户而言，不同的调度策略对系统资源的消耗可以[归一化](#)后抽象为数值。例如，某用户分别使用A/B/C策略的系统消耗分别为15/8/17。
3. 每个用户依次选择当前所能选择的对系统资源消耗最少的策略（局部最优），如果有多个满足要求的策略，选最后一个。

输入描述

第一行表示用户个数n

接下来每一行表示一个用户分别使用三个策略的系统消耗resA resB resC

输出描述

最优策略组合下的总的系统资源消耗数

示例1

输入

```
1 3
2 15 8 17
3 12 20 9
4 11 7 5
```

输出

▼

Plain Text |

```
1 24
```

说明

| 1号用户使用B策略， 2号用户使用C策略， 3号用户使用B策略。系统资源消耗: $8 + 9 + 7 = 24$ 。

解题思路

本题求的是局部最优解，而不是全局最优解。

为什么是局部最优解？

题目要求每个用户依次选择当前所能选择的最小资源消耗策略，并且若有多个策略的资源消耗相同，则选择最后一个策略。这种选择策略本身就是一个局部最优选择的过程：

1. 每个用户的选择是独立的，只考虑该用户的策略和相邻用户的约束，不考虑整个调度序列的全局资源消耗。
2. 每个用户根据自己的当前状态（即自己的三种选择的资源消耗）做出局部最优选择：在当前可选的策略中，选择消耗最小的那个。如果有多个最小值，则选择最后一个（根据题意）。这一选择是局部的最优，即在当下最有利的选择，但并不保证最终的整体系统资源消耗是最小的。

举个例子：

考虑以下输入：

▼

Plain Text |

```
1 3
2 15 8 17
3 12 20 9
4 11 7 5
```

第一位用户：

- 资源消耗分别是：A=15, B=8, C=17。
- 用户选择B，因为B的资源消耗是最小的（8）。此时做出了局部最优选择。

第二位用户：

- 资源消耗分别是：A=12, B=20, C=9。
- 用户不能选择B（因为相邻用户已经选择了B），所以只能在A和C之间选择。此时选择C（因为9是最小的），这也是局部最优选择。

第三位用户：

- 资源消耗分别是：A=11, B=7, C=5。
- 用户不能选择C（因为相邻用户已经选择了C），只能选择A和B。此时选择B（因为7比A的11小），做

出了局部最优选择。

最终的选择是：

- 用户1选择B (消耗8)
- 用户2选择C (消耗9)
- 用户3选择B (消耗7)

总消耗 = 8 + 9 + 7 = 24。

全局最优和局部最优的差异：

- **全局最优解**会考虑整个序列的资源消耗，通过全局最优化的方法（例如动态规划、回溯等），寻找一个能最小化整个系统资源消耗的策略组合。这通常需要在选择每个用户策略时考虑后续的用户选择，以便找到最合适的全局策略。
- **局部最优解**仅考虑当前用户的选择，忽略后续用户可能会带来的影响。因此，它可能导致在某些情况下总资源消耗不一定是最小的，只能保证每一步选择是局部最优的。

```

1 import java.util.Scanner;
2
3 public class Main{
4     public static void main(String[] args) {
5         // 创建Scanner对象
6         Scanner scanner = new Scanner(System.in);
7         // 读取用户个数n
8         int n = scanner.nextInt();
9         // 创建n行3列的二维数组cost，用于存储每个用户使用三个策略的系统消耗resA res
10        B resC
11        int[][] cost = new int[n][3];
12        // 读取每个用户使用三个策略的系统消耗resA resB resC
13        for (int i = 0; i < n; i++) {
14            cost[i][0] = scanner.nextInt();
15            cost[i][1] = scanner.nextInt();
16            cost[i][2] = scanner.nextInt();
17        }
18        // 调用findMinTotalCost方法，计算最优策略组合下的总的系统资源消耗数
19        int result = findMinTotalCost(cost);
20        // 输出最优策略组合下的总的系统资源消耗数
21        System.out.println(result);
22    }
23    // 计算最优策略组合下的总的系统资源消耗数
24    public static int findMinTotalCost(int[][] cost) {
25        // 初始化最小总消耗为整数最大值
26        int minTotalCost = Integer.MAX_VALUE;
27        // 枚举第一个用户使用的调度策略
28        for (int i = 0; i < 3; i++) {
29            // 从第一个用户开始递归调用dfs方法，计算从第一个用户开始到最后一个用户结
30            束的最小总消耗
31            minTotalCost = Math.min(minTotalCost, dfs(cost, 0, i, 0));
32        }
33        // 返回最小总消耗
34        return minTotalCost;
35    }
36    // 从第level个用户开始，枚举第level个用户使用的调度策略index，计算从第level个用
37    户开始到最后一个用户结束的最小总消耗
38    public static int dfs(int[][] cost, int level, int index, int totalCos
39    t) {
40        // 如果已经遍历到最后一个用户，则返回当前总消耗
41        if (level == cost.length) {
42            return totalCost;
43        }

```

```
42 // 获取第level个用户使用三个策略的系统消耗resA resB resC
43 int[] r = cost[level];
44 // 初始化最小消耗为整数最大值
45 int minCost = Integer.MAX_VALUE;
46 // 枚举第level+1个用户使用的调度策略，计算从第level+1个用户开始到最后一个用
47 // 户结束的最小总消耗
48 for (int i = 0; i < r.length; i++) {
49     // 如果第level+1个用户使用的调度策略不等于第level个用户使用的调度策略,
50     // 则递归调用dfs方法，计算从第level+1个用户开始到最后一个用户结束的最小总消耗
51     if (i != index) {
52         minCost = Math.min(minCost, dfs(cost, level + 1, i, totalC
53         ost + r[i]));
54     }
55 }
56 }
```

Python

```

1 # 计算最优策略组合下的总的系统资源消耗数
2 def findMinTotalCost(cost):
3     # 初始化最小总消耗为整数最大值
4     minTotalCost = float("inf")
5     # 枚举第一个用户使用的调度策略
6     for i in range(3):
7         # 从第一个用户开始递归调用dfs方法，计算从第一个用户开始到最后一个用户结束的最
8         # 小总消耗
9         minTotalCost = min(minTotalCost, dfs(cost, 0, i, 0))
10    # 返回最小总消耗
11    return minTotalCost
12
13 # 从第level个用户开始，枚举第level个用户使用的调度策略index，计算从第level个用户开始
14 # 到最后一个用户结束的最小总消耗
15 def dfs(cost, level, index, totalCost):
16     # 如果已经遍历到最后一个用户，则返回当前总消耗
17     if level == len(cost):
18         return totalCost
19     # 获取第level个用户使用三个策略的系统消耗resA resB resC
20     r = cost[level]
21     # 初始化最小消耗为整数最大值
22     minCost = float("inf")
23     # 枚举第level+1个用户使用的调度策略，计算从第level+1个用户开始到最后一个用户结束
24     # 的最小总消耗
25     for i in range(len(r)):
26         # 如果第level+1个用户使用的调度策略不等于第level个用户使用的调度策略，则递归
27         # 调用dfs方法，计算从第level+1个用户开始到最后一个用户结束的最小总消耗
28         if i != index:
29             minCost = min(minCost, dfs(cost, level + 1, i, totalCost + r
30 [i]))
31     # 返回从第level个用户开始到最后一个用户结束的最小总消耗
32     return minCost
33 n = int( input())
34 # 创建n行3列的二维数组cost，用于存储每个用户使用三个策略的系统消耗resA resB resC
35 cost = []
36 for i in range(n):
37     cost.append(list(map(int, input().split())))
38 # 调用findMinTotalCost方法，计算最优策略组合下的总的系统资源消耗数
39 result = findMinTotalCost(cost)
40 # 输出最优策略组合下的总的系统资源消耗数
41 print(result)

```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let n = 0;
9 let cost = [];
10
11 rl.on('line', (input) => {
12     if (n === 0) {
13         n = parseInt(input);
14     } else {
15         const arr = input.split(' ').map(Number);
16         cost.push(arr);
17         if (cost.length === n) {
18             const result = findMinTotalCost(cost);
19             console.log(result);
20             rl.close();
21         }
22     }
23 });
24
25 // 计算最优策略组合下的总的系统资源消耗数
26 function findMinTotalCost(cost) {
27     // 初始化最小总消耗为整数最大值
28     let minTotalCost = Number.MAX_SAFE_INTEGER;
29     // 枚举第一个用户使用的调度策略
30     for (let i = 0; i < 3; i++) {
31         // 从第一个用户开始递归调用dfs方法，计算从第一个用户开始到最后一个用户结束的最小总消耗
32         minTotalCost = Math.min(minTotalCost, dfs(cost, 0, i, 0));
33     }
34     // 返回最小总消耗
35     return minTotalCost;
36 }
37
38 // 从第level个用户开始，枚举第level个用户使用的调度策略index，计算从第level个用户开始到最后一个用户结束的最小总消耗
39 function dfs(cost, level, index, totalCost) {
40     // 如果已经遍历到最后一个用户，则返回当前总消耗
41     if (level === cost.length) {
42         return totalCost;
43     }
```

```
44 // 获取第level个用户使用三个策略的系统消耗resA resB resC
45 const r = cost[level];
46 // 初始化最小消耗为整数最大值
47 let minCost = Number.MAX_SAFE_INTEGER;
48 // 枚举第level+1个用户使用的调度策略，计算从第level+1个用户开始到最后一个用户结束的
49 最小总消耗
50 for (let i = 0; i < r.length; i++) {
51     // 如果第level+1个用户使用的调度策略不等于第level个用户使用的调度策略，则递归调用
52     // dfs方法，计算从第level+1个用户开始到最后一个用户结束的最小总消耗
53     if (i !== index) {
54         minCost = Math.min(minCost, dfs(cost, level + 1, i, totalCost + r
55 [i]));
56     }
57 }
58 // 返回从第level个用户开始到最后一个用户结束的最小总消耗
59 return minCost;
60 }
```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <climits>
4 #include <algorithm>
5 using namespace std;
6
7 int findMinTotalCost(vector<vector<int>>& cost);
8 int dfs(vector<vector<int>>& cost, int level, int index, int totalCost);
9
10 int main() {
11     // 创建输入流对象
12     istream& input = cin;
13     // 读取用户个数n
14     int n;
15     input >> n;
16     // 创建n行3列的二维数组cost, 用于存储每个用户使用三个策略的系统消耗resA resB res
17     C
18     vector<vector<int>> cost(n, vector<int>(3));
19     // 读取每个用户使用三个策略的系统消耗resA resB resC
20     for (int i = 0; i < n; i++) {
21         input >> cost[i][0] >> cost[i][1] >> cost[i][2];
22     }
23     // 调用findMinTotalCost方法, 计算最优策略组合下的总的系统资源消耗数
24     int result = findMinTotalCost(cost);
25     // 输出最优策略组合下的总的系统资源消耗数
26     cout << result << endl;
27     return 0;
28 }
29 // 计算最优策略组合下的总的系统资源消耗数
30 int findMinTotalCost(vector<vector<int>>& cost) {
31     // 初始化最小总消耗为整数最大值
32     int minTotalCost = INT_MAX;
33     // 枚举第一个用户使用的调度策略
34     for (int i = 0; i < 3; i++) {
35         // 从第一个用户开始递归调用dfs方法, 计算从第一个用户开始到最后一个用户结束的最
36         小总消耗
37         minTotalCost = min(minTotalCost, dfs(cost, 0, i, 0));
38     }
39     // 返回最小总消耗
40     return minTotalCost;
41 }
42 // 从第level个用户开始, 枚举第level个用户使用的调度策略index, 计算从第level个用户开
43 始到最后一个用户结束的最小总消耗
```

```
43 int dfs(vector<vector<int>>& cost, int level, int index, int totalCost) {
44     // 如果已经遍历到最后一个用户，则返回当前总消耗
45     if (level == cost.size()) {
46         return totalCost;
47     }
48     // 获取第level个用户使用三个策略的系统消耗resA resB resC
49     vector<int>& r = cost[level];
50     // 初始化最小消耗为整数最大值
51     int minCost = INT_MAX;
52     // 枚举第level+1个用户使用的调度策略，计算从第level+1个用户开始到最后一个用户结束
53     // 的最小总消耗
54     for (int i = 0; i < r.size(); i++) {
55         // 如果第level+1个用户使用的调度策略不等于第level个用户使用的调度策略，则递
56         // 归调用dfs方法，计算从第level+1个用户开始到最后一个用户结束的最小总消耗
57         if (i != index) {
58             minCost = min(minCost, dfs(cost, level + 1, i, totalCost + r
59             [i]));
60         }
61     }
62     // 返回从第level个用户开始到最后一个用户结束的最小总消耗
63     return minCost;
64 }
```

C语言

```
1 #include <stdio.h>
2 #include <limits.h>
3
4 #define MAX_USERS 1000 // 假设最多有 1000 个用户
5
6 // 函数声明
7 int findMinTotalCost(int cost[MAX_USERS][3], int n);
8 int dfs(int cost[MAX_USERS][3], int level, int index, int totalCost, int
n);
9
10 int main() {
11     int n;
12     // 读取用户个数n
13     scanf("%d", &n);
14
15     // 创建二维数组cost, 用于存储每个用户使用三个策略的系统消耗resA, resB, resC
16     int cost[MAX_USERS][3];
17
18     // 读取每个用户使用三个策略的系统消耗resA, resB, resC
19     for (int i = 0; i < n; i++) {
20         scanf("%d %d %d", &cost[i][0], &cost[i][1], &cost[i][2]);
21     }
22
23     // 调用findMinTotalCost方法, 计算最优策略组合下的总的系统资源消耗数
24     int result = findMinTotalCost(cost, n);
25
26     // 输出最优策略组合下的总的系统资源消耗数
27     printf("%d\n", result);
28
29     return 0;
30 }
31
32 // 计算最优策略组合下的总的系统资源消耗数
33 int findMinTotalCost(int cost[MAX_USERS][3], int n) {
34     // 初始化最小总消耗为整数最大值
35     int minTotalCost = INT_MAX;
36
37     // 枚举第一个用户使用的调度策略
38     for (int i = 0; i < 3; i++) {
39         // 从第一个用户开始递归调用dfs方法, 计算从第一个用户开始到最后一个用户结束的最
小总消耗
40         minTotalCost = (minTotalCost < dfs(cost, 0, i, 0, n)) ? minTotalCo
st : dfs(cost, 0, i, 0, n);
41     }
42 }
```

```

43     // 返回最小总消耗
44     return minTotalCost;
45 }
46
47 // 从第level个用户开始，枚举第level个用户使用的调度策略index，计算从第level个用户开
48 // 始到最后一个用户结束的最小总消耗
49 int dfs(int cost[MAX_USERS][3], int level, int index, int totalCost, int
50 n) {
51     // 如果已经遍历到最后一个用户，则返回当前总消耗
52     if (level == n) {
53         return totalCost;
54     }
55
56     // 获取第level个用户使用三个策略的系统消耗resA, resB, resC
57     int* r = cost[level];
58
59     // 初始化最小消耗为整数最大值
60     int minCost = INT_MAX;
61
62     // 枚举第level+1个用户使用的调度策略，计算从第level+1个用户开始到最后一个用户结束
63     // 的最小总消耗
64     for (int i = 0; i < 3; i++) {
65         // 如果第level+1个用户使用的调度策略不等于第level个用户使用的调度策略，则递
66         // 归调用dfs方法，计算从第level+1个用户开始到最后一个用户结束的最小总消耗
67         if (i != index) {
68             minCost = (minCost < dfs(cost, level + 1, i, totalCost + r
69 [i], n)) ? minCost : dfs(cost, level + 1, i, totalCost + r[i], n);
70         }
71     }
72
73     // 返回从第level个用户开始到最后一个用户结束的最小总消耗
74     return minCost;
75 }

```

完整用例

用例1

```

1 3
2 15 8 17
3 12 20 9
4 11 7 5

```

Plain Text

用例2

```
▼ Plain Text |  
1 4  
2 10 15 20  
3 8 12 10  
4 9 7 6  
5 5 6 7
```

用例3

```
▼ Plain Text |  
1 2  
2 7 8 9  
3 6 10 5
```

用例4

```
▼ Plain Text |  
1 5  
2 5 10 15  
3 8 9 6  
4 7 12 11  
5 10 6 9  
6 9 7 10
```

用例5

```
▼ Plain Text |  
1 3  
2 20 15 10  
3 8 9 6  
4 12 11 10
```

用例6

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

Plain Text |

用例7

```
1   4  
2   15 10 20  
3   9 11 7  
4   6 8 5  
5   10 12 11
```

Plain Text |

用例8

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

Plain Text |

用例9

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

Plain Text |

用例10

▼

Plain Text |

```
1 3
2 20 15 10
3 9 11 12
4 10 8 6
```

文章目录

- 最新华为OD机试
- 题目描述
- 输入描述
- 输出描述
- 示例1
- 解题思路
 - 为什么是局部最优解?
 - 举个例子:
 - 第一位用户:
 - 第二位用户:
 - 第三位用户:
 - 全局最优和局部最优的差异:
- Java
- Python
- JavaScript
- C++
- C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10

机考真题 华为OD

No No No



CSDN @算法大师

| 来自: 华为OD机考2025B卷 – 用户调度问题 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 符号运算（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

题目描述

给定一个[表达式](#)，求其分数计算结果。

表达式的限制如下：

1. 所有的输入数字皆为正整数（包括0）
2. 仅支持四则运算 ($\pm*/$) 和括号
3. 结果为整数或分数，分数必须化为最简格式（比如6, $3/4$, $7/8$, $90/7$ ）
4. 除数可能为0，如果遇到这种情况，直接输出"ERROR"
5. 输入和最终计算结果中的数字都不会超出整型范围

用例输入一定合法，不会出现[括号匹配](#)的情况

输入描述

字符串格式的表达式，仅支持 $\pm*/$ ，数字可能超过两位，可能带有空格，没有负数
长度小于200个字符

输出描述

表达式结果，以最简格式表达

- 如果结果为整数，那么直接输出整数
- 如果结果为负数，那么分子分母不可再约分，可以为假分数，不可表达为带分数
- 结果可能是负数，符号放在前面

示例1

输入

```
Plain Text |  
▼  
1 1 + 5 * 7 / 8
```

输出

▼ Plain Text |

```
1 43/8
```

说明

示例2

输入

▼ Plain Text |

```
1 1 / (0 - 5)
```

输出

▼ Plain Text |

```
1 -1/5
```

说明

| 符号需要提到最前面

示例3

输入

▼ Plain Text |

```
1 1 * (3*4/(8-(7+0)))
```

输出

▼ Plain Text |

```
1 12
```

说明

| 注意括号可以多重嵌套

解题思路

```
1 #include <iostream>
2 #include <stack>
3 #include <string>
4 #include <cctype>
5 #include <stdexcept>
6 using namespace std;
7 class Fraction {
8 private:
9     int numerator; // 分子
10    int denominator; // 分母
11
12    // 计算最大公约数的函数
13    int gcd(int a, int b) {
14        return b == 0 ? a : gcd(b, a % b);
15    }
16
17 public:
18    // 构造函数
19    Fraction(int numerator, int denominator) {
20        if (denominator == 0) {
21            throw invalid_argument("Denominator cannot be zero."); // 分
22            母不能为0
23        }
24        int gcdValue = gcd(abs(numerator), abs(denominator)); // 计算最大
25        公约数
26        this->numerator = numerator / gcdValue; // 约分
27        this->denominator = denominator / gcdValue; // 约分
28        if (this->denominator < 0) { // 确保分母为正
29            this->numerator = -this->numerator;
30            this->denominator = -this->denominator;
31        }
32
33    // 加法运算
34    Fraction add(const Fraction& other) const {
35        return Fraction(numerator * other.denominator + other.numerator
36        * denominator,
37                        denominator * other.denominator);
38    }
39
40    // 减法运算
41    Fraction subtract(const Fraction& other) const {
42        return Fraction(numerator * other.denominator - other.numerator
43        * denominator,
44                        denominator * other.denominator);
```

```

42     }
43
44     // 乘法运算
45     Fraction multiply(const Fraction& other) const {
46         return Fraction(numerator * other.numerator, denominator * other.
denominator);
47     }
48
49     // 除法运算
50     Fraction divide(const Fraction& other) const {
51         if (other.numerator == 0) {
52             throw invalid_argument("Division by zero."); // 防止除以0
53         }
54         return Fraction(numerator * other.denominator, denominator * othe
r.numerator);
55     }
56
57     // 用于输出的友元函数
58     friend ostream& operator<<(ostream& os, const Fraction& f);
59 };
60
61 ostream& operator<<(ostream& os, const Fraction& f) {
62     if (f.denominator == 1) {
63         os << f.numerator;
64     } else {
65         os << f.numerator << "/" << f.denominator;
66     }
67     return os;
68 }
69
70 // 定义运算符优先级的函数
71 int precedence(char op) {
72     switch (op) {
73         case '+':
74         case '-':
75             return 1;
76         case '*':
77         case '/':
78             return 2;
79         default:
80             return 0;
81     }
82 }
83
84 // 计算函数, 使用栈操作
85 void calculate(stack<Fraction>& numbers, stack<char>& operators) {
86     Fraction b = numbers.top(); numbers.pop();
87     Fraction a = numbers.top(); numbers.pop();

```

```

88     char op = operators.top(); operators.pop();
89
90     switch (op) {
91         case '+':
92             numbers.push(a.add(b));
93             break;
94         case '-':
95             numbers.push(a.subtract(b));
96             break;
97         case '*':
98             numbers.push(a.multiply(b));
99             break;
100        case '/':
101            numbers.push(a.divide(b));
102            break;
103    }
104 }
105
106 // 表达式计算函数
107 Fraction calculateExpression(const string& expression) {
108     stack<Fraction> numbers;
109     stack<char> operators;
110
111     for (size_t i = 0; i < expression.length(); ++i) {
112         char c = expression[i];
113
114         if (isdigit(c)) { // 如果字符是数字
115             size_t j = i;
116             while (j < expression.length() && isdigit(expression[j])) {
117                 j++;
118             }
119             numbers.push(Fraction(stoi(expression.substr(i, j - i)), 1));
120             i = j - 1;
121         } else if (c == '(') {
122             operators.push(c);
123         } else if (c == ')') {
124             while (operators.top() != '(') {
125                 calculate(numbers, operators);
126             }
127             operators.pop(); // 弹出'
128         } else if (c == '+' || c == '-' || c == '*' || c == '/') {
129             while (!operators.empty() && precedence(c) <= precedence(operators.top())) {
130                 calculate(numbers, operators);
131             }
132             operators.push(c);
133         }
134     }

```

```
135
136     // 处理剩余的运算
137     while (!operators.empty()) {
138         calculate(numbers, operators);
139     }
140
141     // 返回最终结果
142     return numbers.top();
143 }
144
145 int main() {
146     string expression;
147
148     // 读取一行表达式
149     getline(cin, expression);
150
151     try {
152         // 计算表达式结果
153         Fraction result = calculateExpression(expression);
154         // 输出结果
155         cout << result << endl;
156     } catch (const exception& e) {
157         // 处理异常, 比如除以0
158         cout << "ERROR" << endl;
159     }
160
161     return 0;
162 }
```

Java

```

1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         // 创建扫描器读取输入
6         Scanner scanner = new Scanner(System.in);
7         // 读取一行表达式
8         String expression = scanner.nextLine();
9         // 关闭扫描器
10        scanner.close();
11
12        try {
13            // 尝试计算表达式结果
14            Fraction result = calculate(expression);
15            // 输出计算结果
16            System.out.println(result);
17        } catch (ArithmaticException e) {
18            // 捕获并处理算术异常, 比如除以0
19            System.out.println("ERROR");
20        }
21    }
22
23    private static Fraction calculate(String expression) {
24        // 创建两个栈, 一个用于存储数字, 一个用于存储操作符
25        Stack<Fraction> numbers = new Stack<>();
26        Stack<Character> operators = new Stack<>();
27
28        // 遍历表达式的每个字符
29        for (int i = 0; i < expression.length(); i++) {
30            char c = expression.charAt(i);
31
32            // 如果当前字符是数字
33            if (Character.isDigit(c)) {
34                int j = i;
35                // 继续向后读取直到非数字字符
36                while (j < expression.length() && Character.isDigit(expression.charAt(j))) {
37                    j++;
38                }
39                // 将读取到的数字字符串转换为Fraction对象并入栈
40                Fraction number = new Fraction(Integer.parseInt(expression.substring(i, j)), 1);
41                numbers.push(number);
42                i = j - 1;
43            } else if (c == '(') {

```

```

44                         // 如果是左括号，直接入操作符栈
45                         operators.push(c);
46             } else if (c == ')') {
47                 // 如果是右括号，计算到最近一个左括号为止
48                 while (operators.peek() != '(') {
49                     calculate(numbers, operators);
50                 }
51                 // 弹出左括号
52                 operators.pop();
53             } else if (c == '+' || c == '-' || c == '*' || c == '/') {
54                 // 如果是运算符，处理优先级
55                 while (!operators.isEmpty() && precedence(c) <= precedence(operators.peek())) {
56                     calculate(numbers, operators);
57                 }
58                 // 当前运算符入栈
59                 operators.push(c);
60             }
61         }
62
63         // 处理剩余的运算
64         while (!operators.isEmpty()) {
65             calculate(numbers, operators);
66         }
67
68         // 返回计算结果
69         return numbers.pop();
70     }
71
72     private static void calculate(Stack<Fraction> numbers, Stack<Character> operators) {
73         // 从数字栈中弹出两个数字
74         Fraction b = numbers.pop();
75         Fraction a = numbers.pop();
76         // 从操作符栈中弹出操作符
77         char operator = operators.pop();
78
79         // 根据操作符计算结果并入数字栈
80         switch (operator) {
81             case '+':
82                 numbers.push(a.add(b));
83                 break;
84             case '-':
85                 numbers.push(a.subtract(b));
86                 break;
87             case '*':
88                 numbers.push(a.multiply(b));
89                 break;

```

```

90             case '/':
91                 // 注意这里可能会抛出除以0的异常
92                 numbers.push(a.divide(b));
93                 break;
94         }
95     }
96 }
97 
98     private static int precedence(char operator) {
99         // 定义运算符的优先级
100        switch (operator) {
101            case '+':
102            case '-':
103                return 1;
104            case '*':
105            case '/':
106                return 2;
107            default:
108                return 0;
109        }
110    }
111 }

112     class Fraction {
113         private int numerator; // 分子
114         private int denominator; // 分母
115 
116         public Fraction(int numerator, int denominator) {
117             // 分母不能为0
118             if (denominator == 0) {
119                 throw new ArithmeticException("ERROR");
120             }
121             // 计算最大公约数
122             int gcd = gcd(Math.abs(numerator), Math.abs(denominator));
123             // 约分
124             this.numerator = numerator / gcd;
125             this.denominator = denominator / gcd;
126             // 确保分母为正
127             if (this.denominator < 0) {
128                 this.numerator = -this.numerator;
129                 this.denominator = -this.denominator;
130             }
131         }
132 
133         // 加法运算
134         public Fraction add(Fraction other) {
135             return new Fraction(numerator * other.denominator + other.numerat
136             or * denominator, denominator * other.denominator);
137         }

```

```
137
138      // 减法运算
139      public Fraction subtract(Fraction other) {
140          return new Fraction(numerator * other.denominator - other.numerat
141          or * denominator, denominator * other.denominator);
142      }
143
144      // 乘法运算
145      public Fraction multiply(Fraction other) {
146          return new Fraction(numerator * other.numerator, denominator * ot
147          her.denominator);
148      }
149
150      // 除法运算
151      public Fraction divide(Fraction other) {
152          return new Fraction(numerator * other.denominator, denominator *
153          other.numerator);
154      }
155
156      // 计算最大公约数
157      private int gcd(int a, int b) {
158          return b == 0 ? a : gcd(b, a % b);
159      }
160
161      // 重写toString方法, 用于输出
162      @Override
163      public String toString() {
164          if (denominator == 1) {
165              return String.valueOf(numerator);
166          } else {
167              return numerator + "/" + denominator;
168          }
169      }
170  }
```

javaScript


```

45         } else if ('+-*/'.includes(c)) {
46             while (operators.length > 0 && precedence(c) <= precedence(operators[operators.length - 1])) {
47                 performCalculation(numbers, operators);
48             }
49             operators.push(c);
50             i++;
51         } else {
52             i++;
53         }
54     }
55
56     while (operators.length > 0) {
57         performCalculation(numbers, operators);
58     }
59
60     return numbers.pop();
61 }
62
63 function performCalculation(numbers, operators) {
64     let b = numbers.pop();
65     let a = numbers.pop();
66     let operator = operators.pop();
67
68     switch (operator) {
69         case '+':
70             numbers.push(a.add(b));
71             break;
72         case '-':
73             numbers.push(a.subtract(b));
74             break;
75         case '*':
76             numbers.push(a.multiply(b));
77             break;
78         case '/':
79             numbers.push(a.divide(b));
80             break;
81     }
82 }
83
84 function precedence(operator) {
85     switch (operator) {
86         case '+':
87         case '-':
88             return 1;
89         case '*':
90         case '/':
91             return 2;

```

```

92         default:
93             return 0;
94     }
95 }
96
97 class Fraction {
98     constructor(numerator, denominator) {
99         if (denominator === 0) {
100             throw new Error("Division by zero");
101         }
102         const gcd = (a, b) => b ? gcd(b, a % b) : a;
103         let g = gcd(Math.abs(numerator), Math.abs(denominator));
104         this.numerator = numerator / g;
105         this.denominator = denominator / g;
106         if (this.denominator < 0) {
107             this.numerator = -this.numerator;
108             this.denominator = -this.denominator;
109         }
110     }
111
112     add(other) {
113         return new Fraction(this.numerator * other.denominator + other.nu
merator * this.denominator, this.denominator * other.denominator);
114     }
115
116     subtract(other) {
117         return new Fraction(this.numerator * other.denominator - other.nu
merator * this.denominator, this.denominator * other.denominator);
118     }
119
120     multiply(other) {
121         return new Fraction(this.numerator * other.numerator, this.denomi
nator * other.denominator);
122     }
123
124     divide(other) {
125         return new Fraction(this.numerator * other.denominator, this.deno
minator * other.numerator);
126     }
127
128     toString() {
129         return this.denominator === 1 ? `${this.numerator}` : `${this.num
erator}/${this.denominator}`;
130     }
131 }

```

Python

```
1 import fractions
2
3 def main():
4     # 输入一行表达式
5     expression = input(" ")
6
7     try:
8         # 计算表达式结果
9         result = calculate(expression)
10        # 输出计算结果
11        print(result)
12    except ArithmeticException:
13        # 捕获并处理算术异常, 比如除以0
14        print("ERROR")
15
16 def calculate(expression):
17     # 创建两个栈, 一个用于存储数字, 一个用于存储操作符
18     numbers = []
19     operators = []
20
21     i = 0
22     while i < len(expression):
23         c = expression[i]
24
25         # 如果当前字符是数字
26         if c.isdigit():
27             j = i
28             # 继续向后读取直到非数字字符
29             while j < len(expression) and expression[j].isdigit():
30                 j += 1
31             # 将读取到的数字字符串转换为Fraction对象并入栈
32             number = fractions.Fraction(int(expression[i:j]))
33             numbers.append(number)
34             i = j
35         elif c == '(':
36             # 如果是左括号, 直接入操作符栈
37             operators.append(c)
38             i += 1
39         elif c == ')':
40             # 如果是右括号, 计算到最近一个左括号为止
41             while operators[-1] != '(':
42                 perform_calculation(numbers, operators)
43             # 弹出左括号
44             operators.pop()
45             i += 1
```

```
46     elif c in '+-*':
47         # 如果是运算符, 处理优先级
48         while operators and precedence(c) <= precedence(operators[-1]):
49             perform_calculation(numbers, operators)
50             # 当前运算符入栈
51             operators.append(c)
52             i += 1
53         else:
54             # 忽略非法字符
55             i += 1
56
57     # 处理剩余的运算
58     while operators:
59         perform_calculation(numbers, operators)
60
61     # 返回计算结果
62     return numbers.pop()
63
64 def perform_calculation(numbers, operators):
65     # 从数字栈中弹出两个数字
66     b = numbers.pop()
67     a = numbers.pop()
68     # 从操作符栈中弹出操作符
69     operator = operators.pop()
70
71     # 根据操作符计算结果并入数字栈
72     if operator == '+':
73         numbers.append(a + b)
74     elif operator == '-':
75         numbers.append(a - b)
76     elif operator == '*':
77         numbers.append(a * b)
78     elif operator == '/':
79         # 注意这里可能会抛出除以0的异常
80         numbers.append(a / b)
81
82 def precedence(operator):
83     # 定义运算符的优先级
84     if operator in '+-':
85         return 1
86     elif operator in '*/':
87         return 2
88     else:
89         return 0
90
91 if __name__ == "__main__":
92     main()
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <ctype.h>
5 #include <stdbool.h>
6
7 // 分数结构体
8 typedef struct {
9     int numerator;    // 分子
10    int denominator; // 分母
11 } Fraction;
12
13 // 分数栈
14 typedef struct {
15     Fraction* data;
16     int capacity;
17     int top;
18 } FractionStack;
19
20 // 字符栈
21 typedef struct {
22     char* data;
23     int capacity;
24     int top;
25 } CharStack;
26
27 // 计算最大公约数的函数
28 int gcd(int a, int b) {
29     a = abs(a);
30     b = abs(b);
31     return b == 0 ? a : gcd(b, a % b);
32 }
33
34 // 创建分数
35 Fraction createFraction(int numerator, int denominator) {
36     Fraction f;
37
38     if (denominator == 0) {
39         // 分母不能为0, 这里用特殊值表示错误
40         f.numerator = 1;
41         f.denominator = 0;
42         return f;
43     }
44
45     int gcdValue = gcd(abs(numerator), abs(denominator)); // 计算最大公约数
```

```

46     f.numerator = numerator / gcdValue; // 约分
47     f.denominator = denominator / gcdValue; // 约分
48
49     if (f.denominator < 0) { // 确保分母为正
50         f.numerator = -f.numerator;
51         f.denominator = -f.denominator;
52     }
53
54     return f;
55 }
56
57 // 加法运算
58 Fraction add(const Fraction* a, const Fraction* b) {
59     return createFraction(a->numerator * b->denominator + b->numerator * a
60     ->denominator,
61                         a->denominator * b->denominator);
62 }
63
64 // 减法运算
65 Fraction subtract(const Fraction* a, const Fraction* b) {
66     return createFraction(a->numerator * b->denominator - b->numerator * a
67     ->denominator,
68                         a->denominator * b->denominator);
69 }
70
71 // 乘法运算
72 Fraction multiply(const Fraction* a, const Fraction* b) {
73     return createFraction(a->numerator * b->numerator, a->denominator * b-
74     >denominator);
75 }
76
77 // 除法运算
78 Fraction divide(const Fraction* a, const Fraction* b) {
79     if (b->numerator == 0) {
80         // 防止除以0, 这里用特殊值表示错误
81         Fraction result = {1, 0};
82         return result;
83     }
84     return createFraction(a->numerator * b->denominator, a->denominator *
85     b->numerator);
86 }
87
88 // 打印分数
89 void printFraction(const Fraction* f) {
90     if (f->denominator == 0) {
91         printf("ERROR");
92         return;
93     }

```

```

90
91     if (f->denominator == 1) {
92         printf("%d", f->numerator);
93     } else {
94         printf("%d/%d", f->numerator, f->denominator);
95     }
96 }
97 }

98 // 初始化分数栈
99 FractionStack* createFractionStack(int capacity) {
100     FractionStack* stack = (FractionStack*)malloc(sizeof(FractionStack));
101     stack->data = (Fraction*)malloc(sizeof(Fraction) * capacity);
102     stack->capacity = capacity;
103     stack->top = -1;
104     return stack;
105 }
106 }

107 // 判断分数栈是否为空
108 bool isFractionStackEmpty(FractionStack* stack) {
109     return stack->top == -1;
110 }
111 }

112 // 分数入栈
113 void pushFraction(FractionStack* stack, Fraction value) {
114     if (stack->top + 1 >= stack->capacity) {
115         // 栈满, 扩容
116         stack->capacity *= 2;
117         stack->data = (Fraction*)realloc(stack->data, sizeof(Fraction) * s
118         tack->capacity);
119     }
120     stack->data[++stack->top] = value;
121 }

122 // 分数出栈
123 Fraction popFraction(FractionStack* stack) {
124     if (isFractionStackEmpty(stack)) {
125         // 栈空, 返回特殊值
126         Fraction error = {0, 0};
127         return error;
128     }
129     return stack->data[stack->top--];
130 }

132 // 获取分数栈顶元素
133 Fraction topFraction(FractionStack* stack) {
134     if (isFractionStackEmpty(stack)) {
135         // 栈空, 返回特殊值
136         Fraction error = {0, 0};

```

```

137         return error;
138     }
139     return stack->data[stack->top];
140 }
141
142 // 释放分数栈
143 void freeFractionStack(FractionStack* stack) {
144     free(stack->data);
145     free(stack);
146 }
147
148 // 初始化字符栈
149 CharStack* createCharStack(int capacity) {
150     CharStack* stack = (CharStack*)malloc(sizeof(CharStack));
151     stack->data = (char*)malloc(sizeof(char) * capacity);
152     stack->capacity = capacity;
153     stack->top = -1;
154     return stack;
155 }
156
157 // 判断字符栈是否为空
158 bool isCharStackEmpty(CharStack* stack) {
159     return stack->top == -1;
160 }
161
162 // 字符入栈
163 void pushChar(CharStack* stack, char value) {
164     if (stack->top + 1 >= stack->capacity) {
165         // 栈满, 扩容
166         stack->capacity *= 2;
167         stack->data = (char*)realloc(stack->data, sizeof(char) * stack->
168                                     capacity);
169     }
170     stack->data[++stack->top] = value;
171 }
172
173 // 字符出栈
174 char popChar(CharStack* stack) {
175     if (isCharStackEmpty(stack)) {
176         // 栈空, 返回特殊值
177         return '\0';
178     }
179     return stack->data[stack->top--];
180 }
181
182 // 获取字符栈顶元素
183 char topChar(CharStack* stack) {
184     if (isCharStackEmpty(stack)) {

```

```

184         // 栈空, 返回特殊值
185         return '\0';
186     }
187     return stack->data[stack->top];
188 }
189
190 // 释放字符栈
191 void freeCharStack(CharStack* stack) {
192     free(stack->data);
193     free(stack);
194 }
195
196 // 定义运算符优先级的函数
197 int precedence(char op) {
198     switch (op) {
199         case '+':
200         case '-':
201             return 1;
202         case '*':
203         case '/':
204             return 2;
205         default:
206             return 0;
207     }
208 }
209
210 // 计算函数, 使用栈操作
211 bool calculate(FractionStack* numbers, CharStack* operators) {
212     Fraction b = popFraction(numbers);
213     Fraction a = popFraction(numbers);
214     char op = popChar(operators);
215
216     switch (op) {
217         case '+':
218             pushFraction(numbers, add(&a, &b));
219             break;
220         case '-':
221             pushFraction(numbers, subtract(&a, &b));
222             break;
223         case '*':
224             pushFraction(numbers, multiply(&a, &b));
225             break;
226         case '/':
227         {
228             Fraction result = divide(&a, &b);
229             if (result.denominator == 0) {
230                 return false; // 除以零错误
231             }

```

```

232                     pushFraction(numbers, result);
233                 }
234             break;
235         }
236     return true;
237 }
238
239 // 表达式计算函数
240 Fraction calculateExpression(const char* expression, bool* success) {
241     FractionStack* numbers = createFractionStack(10);
242     CharStack* operators = createCharStack(10);
243     int len = strlen(expression);
244     *success = true;
245
246     for (int i = 0; i < len; ++i) {
247         char c = expression[i];
248
249         if (isdigit(c)) { // 如果字符是数字
250             int j = i;
251             while (j < len && isdigit(expression[j])) {
252                 j++;
253             }
254
255             // 提取数字并转换为分数
256             char* numStr = (char*)malloc(j - i + 1);
257             strncpy(numStr, expression + i, j - i);
258             numStr[j - i] = '\0';
259             int num = atoi(numStr);
260             free(numStr);
261
262             pushFraction(numbers, createFraction(num, 1));
263             i = j - 1;
264         } else if (c == '(') {
265             pushChar(operators, c);
266         } else if (c == ')') {
267             while (!isCharStackEmpty(operators) && topChar(operators) !=
268             '(') {
269                 if (!calculate(numbers, operators)) {
270                     *success = false;
271                     freeFractionStack(numbers);
272                     freeCharStack(operators);
273                     Fraction error = {1, 0};
274                     return error;
275                 }
276             popChar(operators); // 弹出'(
277         } else if (c == '+' || c == '-' || c == '*' || c == '/') {
278

```

```

279         while (!isCharStackEmpty(operators) && precedence(c) <= preced
280 ence(topChar(operators))) {
281             if (!calculate(numbers, operators)) {
282                 *success = false;
283                 freeFractionStack(numbers);
284                 freeCharStack(operators);
285                 Fraction error = {1, 0};
286                 return error;
287             }
288         }
289     }
290 }
291
292 // 处理剩余的运算
293 while (!isCharStackEmpty(operators)) {
294     if (!calculate(numbers, operators)) {
295         *success = false;
296         freeFractionStack(numbers);
297         freeCharStack(operators);
298         Fraction error = {1, 0};
299         return error;
300     }
301 }
302
303 // 返回最终结果
304 Fraction result = topFraction(numbers);
305
306 freeFractionStack(numbers);
307 freeCharStack(operators);
308
309 return result;
310 }
311
312 int main() {
313     char expression[1000];
314
315     // 读取一行表达式
316     if (fgets(expression, sizeof(expression), stdin) != NULL) {
317         // 移除结尾的换行符
318         size_t len = strlen(expression);
319         if (len > 0 && expression[len-1] == '\n') {
320             expression[len-1] = '\0';
321         }
322
323         bool success;
324         // 计算表达式结果
325         Fraction result = calculateExpression(expression, &success);

```

```
326  
327     // 输出结果  
328     if (success && result.denominator != 0) {  
329         printFraction(&result);  
330         printf("\n");  
331     } else {  
332         printf("ERROR\n");  
333     }  
334 }  
335  
336     return 0;  
}
```

完整用例

用例1

```
1 1 + 2 * 3 - 4 / 2
```

Plain Text |

用例2

```
1 1 + (2 * (3 + 4))
```

Plain Text |

用例3

```
1 (3 * (2 - 5)) / 2
```

Plain Text |

用例4

```
1 123456 + 654321
```

Plain Text |

用例5

▼

Plain Text |

```
1 1 * (2 + 3) / 5
```

用例6

▼

Plain Text |

```
1 1 + 2 * 3
```

用例7

▼

Plain Text |

```
1 1 * (2 + 3) / 5
```

用例8

▼

Plain Text |

```
1 20 / 4
```

用例9

▼

Plain Text |

```
1 6 * 7
```

用例10

▼

Plain Text |

```
1 5 / (2 - 2)
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)

- 示例1
- 示例2
- 示例3
- 解题思路
- C++
- Java
- javaScript
- Python
- C语言
 - 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10



| 来自: 华为OD机考2025B卷 – 符号运算 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 机房布局（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

题目描述

小明正在规划一个大型[数据中心](#)机房，为了使得机柜上的机器都能正常满负荷工作，需要确保在每个机柜边上至少要有一个电箱。

为了简化题目，假设这个机房是一整排，M表示机柜，I表示间隔，请你返回这整排机柜，至少需要多少个电箱。如果无解请返回 -1。

输入描述

```
cabinets = "MIIM"
```

备注：

其中 M 表示机柜，I 表示间隔

$1 \leq \text{strlen}(\text{cabinets}) \leq 10000$

其中 $\text{cabinets}[i] = 'M'$ 或者 ' I '

输出描述

```
2
```

表示至少需要2个电箱

示例1

输入

```
▼ Plain Text |  
1 MIIM
```

输出

Plain Text |

1 2

示例2

输入

Plain Text |

1 MIM

输出

Plain Text |

1 1

示例3

输入

Plain Text |

1 M

输出

Plain Text |

1 -1

示例4

输入

Plain Text |

1 MMM

输出

Plain Text |

1 -1

示例5

输入

Plain Text |

1 I

输出

Plain Text |

1 0

解题思路

这个题目的主要意图是计算在给定的机柜排列中，至少需要多少个电箱来保证所有机柜都能正常工作。

示例分析

- **示例 1:**

- 输入: M I I M
 - 布局: M (电箱放在这) | M (电箱放在这) | M
 - 可以放置 2 个电箱，分别在第一个和第三个机柜旁边。
 - 输出: 2

- **示例 2:**

- 输入: M I M
 - 布局: M (电箱放在这) | M
 - 可以放置 1 个电箱，在第一个机柜旁边即可。
 - 输出: 1

- **示例 3:**

- 输入: M
 - 只有一个机柜，没有间隔可供放置电箱。
 - 输出: -1 (因为无法满足条件)。

好的，以下是结合稍微复杂用例的优化表达：

代码思路

本题的关键在于合理放置电箱，以确保每个机柜旁边至少有一个电箱。我们采用从左到右的遍历方式，并优先将电箱放在机柜的右侧。

为什么优先放在右侧？

考虑以下示例：

```
▼ Plain Text |  
1 I M I M I
```

在这种情况下，如果将电箱放在第一个红色 `I` 的位置，只需一个电箱；而如果放在绿色 `I` 的位置，则需要两个电箱，显然不如前者经济。

再看一个稍复杂的例子：

```
▼ Plain Text |  
1 M I M I I M
```

从左到右遍历，遇到第一个 `M` 时，我们优先在其右侧（即第 `i + 1` 位置）放置电箱。接着，再遇到下一个 `M`，右侧同样可以放电箱。对于第三个 `M`，其右侧为 `I`，也可以放置电箱。

特殊情况处理

在如下情况下：

```
▼ Plain Text |  
1 I M M I I
```

当遇到第一个 `M` 时，如果其右侧无法放电箱，则需要检查左侧，发现可以放电箱。之后再遇到第二个 `M`，其右侧为 `I`，可以继续放置电箱。

然而，如果出现以下情况：

```
▼ Plain Text |  
1 M M M
```

则无论从左侧还是右侧都无法放置电箱，因此应直接返回 `-1`。

重要考虑

当机柜的右侧可以放电箱时，例如在第 `i` 个位置是机柜，第 `i + 1` 个位置是间隔，此时放置电箱后，我们是否还需考虑第 `i + 2` 个位置？

例如：

▼

Plain Text |

```
1 M I M I
```

对于红色的 `M`，由于必然会有一个电箱，因此可以直接跳过 `i + 2` 和 `i + 3` 的位置，重新开始判断。

这种策略能有效减少不必要的判断，提升算法效率。通过优先放电箱并合理处理特殊情况，我们可以确保最小化所需电箱的数量。

▼

Java |

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         String layout = scanner.nextLine(); // 读取机柜布局
7         int len = layout.length(); // 获取机柜布局的长度
8         int count = 0; // 初始化电箱数量为0
9
10        // 遍历机柜布局的每一个字符
11        for (int i = 0; i < len; i++) {
12            // 如果当前字符是 'M'，表示这是一个机柜
13            if (layout.charAt(i) == 'M') {
14                // 检查当前机柜的右侧是否有间隔 'I'
15                if (i + 1 < len && layout.charAt(i + 1) == 'I') {
16                    count++; // 在右侧放置一个电箱
17                    i += 2; // 跳过下一个间隔，继续检查后面的机柜
18                }
19                // 检查当前机柜的左侧是否有间隔 'I'
20                else if (i - 1 >= 0 && layout.charAt(i - 1) == 'I') {
21                    count++; // 在左侧放置一个电箱
22                }
23                // 如果左右都没有间隔，无法放电箱
24                else {
25                    count = -1; // 设置 count 为 -1 表示无解
26                    break; // 退出循环
27                }
28            }
29        }
30        // 输出结果：电箱数量或 -1 表示无解
31        System.out.println(count); // 打印最终的电箱数量
32    }
33 }
```

Python

```
Plain Text | ▾  
1  def main():  
2      # 读取机柜布局  
3      layout = input()  
4      length = len(layout) # 获取机柜布局的长度  
5      count = 0 # 初始化电箱数量为0  
6  
7      # 遍历机柜布局的每一个字符  
8      i = 0  
9      while i < length:  
10          # 如果当前字符是 'M'，表示这是一个机柜  
11          if layout[i] == 'M':  
12              # 检查当前机柜的右侧是否有间隔 'I'  
13              if i + 1 < length and layout[i + 1] == 'I':  
14                  count += 1 # 在右侧放置一个电箱  
15                  i += 2 # 跳过下一个间隔，继续检查后面的机柜  
16              # 检查当前机柜的左侧是否有间隔 'I'  
17              elif i - 1 >= 0 and layout[i - 1] == 'I':  
18                  count += 1 # 在左侧放置一个电箱  
19              # 如果左右都没有间隔，无法放电箱  
20              else:  
21                  count = -1 # 设置 count 为 -1 表示无解  
22                  break # 退出循环  
23          i += 1  
24  
25      # 输出结果：电箱数量或 -1 表示无解  
26      print(count)  
27  
28  if __name__ == "__main__":  
29      main()
```

JavaScript

```
1 const readline = require('readline');
2
3 // 创建接口以读取输入
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout
7 });
8
9 // 读取机柜布局
10 rl.on('line', (layout) => {
11     let len = layout.length; // 获取机柜布局的长度
12     let count = 0; // 初始化电箱数量为0
13
14     // 遍历机柜布局的每一个字符
15     for (let i = 0; i < len; i++) {
16         // 如果当前字符是 'M', 表示这是一个机柜
17         if (layout[i] === 'M') {
18             // 检查当前机柜的右侧是否有间隔 'I'
19             if (i + 1 < len && layout[i + 1] === 'I') {
20                 count++; // 在右侧放置一个电箱
21                 i += 2; // 跳过下一个间隔, 继续检查后面的机柜
22             }
23             // 检查当前机柜的左侧是否有间隔 'I'
24             else if (i - 1 >= 0 && layout[i - 1] === 'I') {
25                 count++; // 在左侧放置一个电箱
26             }
27             // 如果左右都没有间隔, 无法放电箱
28             else {
29                 count = -1; // 设置 count 为 -1 表示无解
30                 break; // 退出循环
31             }
32         }
33     }
34     // 输出结果: 电箱数量或 -1 表示无解
35     console.log(count); // 打印最终的电箱数量
36 });
});
```

C++

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 int main() {
7     string layout; // 存储机柜布局的字符串
8     getline(cin, layout); // 读取机柜布局
9     int len = layout.length(); // 获取机柜布局的长度
10    int count = 0; // 初始化电箱数量为0
11
12    // 遍历机柜布局的每一个字符
13    for (int i = 0; i < len; i++) {
14        // 如果当前字符是 'M', 表示这是一个机柜
15        if (layout[i] == 'M') {
16            // 检查当前机柜的右侧是否有间隔 'I'
17            if (i + 1 < len && layout[i + 1] == 'I') {
18                count++; // 在右侧放置一个电箱
19                i += 2; // 跳过下一个间隔, 继续检查后面的机柜
20            }
21            // 检查当前机柜的左侧是否有间隔 'I'
22            else if (i - 1 >= 0 && layout[i - 1] == 'I') {
23                count++; // 在左侧放置一个电箱
24            }
25            // 如果左右都没有间隔, 无法放电箱
26            else {
27                count = -1; // 设置 count 为 -1 表示无解
28                break; // 退出循环
29            }
30        }
31    }
32    // 输出结果: 电箱数量或 -1 表示无解
33    cout << count << endl; // 打印最终的电箱数量
34
35 }
```

C语言

```

1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     char layout[10000]; // 存储机柜布局的字符串
6     fgets(layout, sizeof(layout), stdin); // 读取机柜布局
7     int len = strlen(layout) - 1; // 获取机柜布局的长度，减去换行符
8     layout[len] = '\0'; // 去除换行符
9     int count = 0; // 初始化电箱数量为0
10
11    // 遍历机柜布局的每一个字符
12    for (int i = 0; i < len; i++) {
13        // 如果当前字符是 'M'，表示这是一个机柜
14        if (layout[i] == 'M') {
15            // 检查当前机柜的右侧是否有间隔 'I'
16            if (i + 1 < len && layout[i + 1] == 'I') {
17                count++; // 在右侧放置一个电箱
18                i += 2; // 跳过下一个间隔，继续检查后面的机柜
19            }
20            // 检查当前机柜的左侧是否有间隔 'I'
21            else if (i - 1 >= 0 && layout[i - 1] == 'I') {
22                count++; // 在左侧放置一个电箱
23            }
24            // 如果左右都没有间隔，无法放电箱
25            else {
26                count = -1; // 设置 count 为 -1 表示无解
27                break; // 退出循环
28            }
29        }
30    }
31    // 输出结果：电箱数量或 -1 表示无解
32    printf("%d\n", count); // 打印最终的电箱数量
33    return 0;
34 }
```

完整用例

用例1

```
1 MIIM
```

用例2

▼ Plain Text |
1 MIM

用例3

▼ Plain Text |
1 M

用例4

▼ Plain Text |
1 MMM

用例5

▼ Plain Text |
1 I

用例6

▼ Plain Text |
1 IMIMI

用例7

▼ Plain Text |
1 IIIMMMIII

用例8

Plain Text |

1 MMIMIMIM

用例9

Plain Text |

1 MIMIMMMIMIMIMIMI

用例10

Plain Text |

1 IMIMIMIMIMIIIIIM

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
 - [备注:](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [示例3](#)
- [示例4](#)
- [示例5](#)
- [解题思路](#)
 - [示例分析](#)
- [代码思路](#)
 - [为什么优先放在右侧?](#)
 - [特殊情况处理](#)
 - [重要考虑](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)
- [完整用例](#)

- 用例1
- 用例2
- 用例3
- 用例4
- 用例5
- 用例6
- 用例7
- 用例8
- 用例9
- 用例10



| 来自: 华为OD机考2025B卷 – 机房布局 (Java & Python& JS & C++ & C) -CSDN博客

华为OD机考2025B卷 - 字符串重新排序（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

给定一个字符串s，s包括以空格分隔的若干个单词，请对s进行如下处理后输出：

1、单词内部调整：对每个单词字母重新按字典序排序

2、单词间顺序调整：

- 统计每个单词出现的次数，并按次数降序排列
- 次数相同，按单词长度升序排列
- 次数和单词长度均相同，按字典升序排列

请输出处理后的字符串，每个单词以一个空格分隔

输入描述

一行字符串，每个字符取值范围：[a-zA-Z0-9]以及空格，[字符串长度](#)范围：[1, 1000]

输出描述

输出处理后的字符串，每个单词以一个空格分隔。

示例1

输入

```
Plain Text |  
▼  
1 This is an apple
```

输出

```
Plain Text |  
▼  
1 an is This aelpp
```

说明

示例2

输入

```
Plain Text |  
1 My sister is in the house not in the yard
```

输出

```
Plain Text |  
1 in in eht eht My is not adry ehosu eirsst
```

说明

解题思路

考察的是排序

1. 对每个单词内部进行字典序排序
2. 按照单词出现次数、单词长度、字典序升序排列

具体实现思路如下：

1. 使用分割字符串，得到每个单词
2. 对每个单词进行字典序排序，并存储到一个 List 中
3. 统计每个单词出现的次数，并存储到一个 Map 中
4. 使用对 List 进行排序，排序规则如下：
 - 次数不同，按照次数降序排列
 - 次数相同，长度不同，按照长度升序排列
 - 次数和长度都相同，按照字典序升序排列
5. 输出处理后的字符串，每个单词以一个空格分隔

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         String input = scanner.nextLine();
7         scanner.close();
8
9         // 使用空格分割字符串
10        String[] words = input.split(" "); // 分割字符串为单词
11
12        // 对每个单词内部进行字典序排序
13        for (int i = 0; i < words.length; i++) {
14            char[] chars = words[i].toCharArray();
15            Arrays.sort(chars);
16            words[i] = new String(chars);
17        }
18
19        // 统计每个单词出现的次数
20        Map<String, Integer> count = new HashMap<>();
21        for (String word : words) {
22            count.put(word, count.getOrDefault(word, 0) + 1);
23        }
24
25        // 进行排序 (使用简单的冒泡排序代替 Collections.sort)
26        for (int i = 0; i < words.length - 1; i++) {
27            for (int j = 0; j < words.length - 1 - i; j++) {
28                // 进行比较
29                if (shouldSwap(words[j], words[j + 1], count)) {
30                    // 交换
31                    String temp = words[j];
32                    words[j] = words[j + 1];
33                    words[j + 1] = temp;
34                }
35            }
36        }
37
38        // 输出处理后的字符串
39        StringBuilder sb = new StringBuilder();
40        for (String word : words) {
41            sb.append(word).append(" ");
42        }
43        System.out.println(sb.toString().trim());
44    }
45}
```

```

46     // 判断是否需要交换两个单词
47     private static boolean shouldSwap(String a, String b, Map<String, Integer> count) {
48         if (!count.get(a).equals(count.get(b))) {
49             // 次数不同, 按照次数降序排列
50             return count.get(a) < count.get(b);
51         } else if (a.length() != b.length()) {
52             // 次数相同, 长度不同, 按照长度升序排列
53             return a.length() > b.length();
54         } else {
55             // 次数和长度都相同, 按照字典序升序排列
56             return a.compareTo(b) > 0;
57         }
58     }
59 }
```

Python

```

Plain Text | ▾

1 import collections
2 import functools
3
4 input = input()
5 words = input.split(" ")
6
7 # 对每个单词内部进行字典序排序
8 words = ["".join(sorted(word)) for word in words]
9
10 # 统计每个单词出现的次数
11 count = collections.Counter(words)
12
13 # 按照要求排序
14 words = sorted(words, key=functools.cmp_to_key(lambda a, b: count[b] - count[a] if count[a] != count[b] else len(a) - len(b) if len(a) != len(b) else -1 if a < b else 1))
15
16 # 输出处理后的字符串
17 output = " ".join(words)
18 print(output)
```

JavaScript

```
1 const readline = require("readline");
2 const rl = readline.createInterface({
3     input: process.stdin,
4     output: process.stdout,
5 });
6
7 rl.on("line", (input) => {
8     const words = input.split(" ");
9
10    // 对每个单词内部进行字典序排序
11    const sortedWords = words.map((word) => word.split("").sort().join(""));
12
13    // 统计每个单词出现的次数
14    const count = sortedWords.reduce((acc, word) => {
15        acc[word] = (acc[word] || 0) + 1;
16        return acc;
17    }, {});
18
19    // 按照要求排序
20    sortedWords.sort((a, b) => {
21        if (count[b] !== count[a]) {
22            return count[b] - count[a]; // 按出现次数降序
23        }
24        if (a.length !== b.length) {
25            return a.length - b.length; // 按长度升序
26        }
27        return a < b ? -1 : 1; // 字典序升序
28    });
29
30    // 输出处理后的字符串
31    console.log(sortedWords.join(" "));
32    rl.close();
33});
```

C++

```
1 #include <iostream>
2 #include <algorithm>
3 #include <unordered_map>
4 #include <sstream>
5 #include <vector>
6
7 using namespace std;
8
9 int main() {
10     // 读入字符串
11     string input;
12     getline(cin, input);
13     // 使用 istringstream 分割字符串
14     istringstream iss(input);
15     string token;
16     vector<string> words;
17     while (getline(iss, token, ' ')) {
18         // 对每个单词内部进行字典序排序
19         sort(token.begin(), token.end());
20         words.push_back(token);
21     }
22
23     // 统计每个单词出现的次数
24     unordered_map<string, int> count;
25     for (string word : words) {
26         count[word]++;
27     }
28
29     // 按照要求排序
30     sort(words.begin(), words.end(), [&](const string& a, const string&
31 b) {
32         if (count[a] != count[b]) {
33             // 次数不同, 按照次数降序排列
34             return count[b] < count[a];
35         } else if (a.length() != b.length()) {
36             // 次数相同, 长度不同, 按照长度升序排列
37             return a.length() < b.length();
38         } else {
39             // 次数和长度都相同, 按照字典序升序排列
40             return a < b;
41         }
42     });
43
44     // 输出处理后的字符串
45     ostringstream oss;
```

```
45     for (string word : words) {
46         oss << word << " ";
47     }
48     cout << oss.str() << endl;
49
50     return 0;
51 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_WORDS 1000      // 定义最大单词数量
6 #define MAX_LENGTH 100      // 定义单词的最大长度
7
8 // 定义一个结构体，用于存储单词和其出现的次数
9 typedef struct {
10     char word[MAX_LENGTH]; // 存储单词
11     int count;             // 存储单词出现的次数
12 } WordCount;
13
14 // 字母排序函数
15 void sortString(char *str) {
16     int length = strlen(str); // 获取字符串的长度
17     // 使用 qsort 对字符串的字母进行排序
18     qsort(str, length, sizeof(char), (int (*)(const void *, const void
19 *)) strcmp);
20 }
21
22 // 比较函数，用于对 WordCount 结构体数组进行排序
23 int compare(const void *a, const void *b) {
24     WordCount *wordCountA = (WordCount *)a; // 将 void 指针转换为 WordCount
25     WordCount *wordCountB = (WordCount *)b; // 将 void 指针转换为 WordCount
26
27     // 先比较出现次数
28     if (wordCountA->count != wordCountB->count) {
29         return wordCountB->count - wordCountA->count; // 次数降序排列
30     } else {
31         // 如果出现次数相同，则比较单词长度
32         int lenDiff = strlen(wordCountA->word) - strlen(wordCountB->word);
33         if (lenDiff != 0) {
34             return lenDiff; // 长度升序排列
35         } else {
36             return strcmp(wordCountA->word, wordCountB->word); // 字典序排
37         }
38     }
39
40     int main() {
41         char input[1000]; // 定义一个字符数组用于存储输入的字符串
```

```
42 fgets(input, sizeof(input), stdin); // 从标准输入读取一行字符串
43
44 // 处理单词的数组
45 char *words[MAX_WORDS]; // 存储分割后的单词
46 int wordCount = 0; // 记录总共的单词数量
47
48 // 使用 strtok 函数分割单词，以空格和换行符为分隔符
49 char *token = strtok(input, " \n");
50 while (token != NULL) {
51     char *sortedWord = malloc(MAX_LENGTH); // 动态分配内存存储排序后的单词
52     strcpy(sortedWord, token); // 复制当前分割出的单词
53     sortString(sortedWord); // 对单词内部进行字典序排序
54     words[wordCount++] = sortedWord; // 将排序后的单词存入数组
55     token = strtok(NULL, " \n"); // 继续分割下一个单词
56 }
57
58 // 统计每个单词出现的次数
59 WordCount wordCounts[MAX_WORDS]; // 存储每个单词及其出现次数的数组
60 int uniqueCount = 0; // 记录不同单词的数量
61
62 // 遍历所有分割出的单词
63 for (int i = 0; i < wordCount; i++) {
64     int found = 0; // 标记是否找到了重复的单词
65     // 检查当前单词是否已经存在于 wordCounts 数组中
66     for (int j = 0; j < uniqueCount; j++) {
67         if (strcmp(wordCounts[j].word, words[i]) == 0) {
68             wordCounts[j].count++; // 如果找到，增加计数
69             found = 1; // 标记已找到
70             break;
71         }
72     }
73     // 如果没有找到该单词，则将其添加到 wordCounts 中
74     if (!found) {
75         strcpy(wordCounts[uniqueCount].word, words[i]); // 复制单词
76         wordCounts[uniqueCount].count = 1; // 初始化计数为 1
77         uniqueCount++; // 增加不同单词的计数
78     }
79 }
80
81 // 按出现次数对单词进行排序
82 qsort(wordCounts, uniqueCount, sizeof(WordCount), compare);
83
84 // 输出结果
85 for (int i = 0; i < uniqueCount; i++) {
86     // 根据单词的出现次数输出每个单词
87     for (int j = 0; j < wordCounts[i].count; j++) {
88         printf("%s ", wordCounts[i].word);
89     }
}
```

```
90     }
91     printf("\n"); // 输出换行
92
93     // 释放动态分配的内存
94     for (int i = 0; i < wordCount; i++) {
95         free(words[i]); // 释放每个排序后单词的内存
96     }
97
98     return 0; // 程序结束
99 }
```

完整用例

用例1

Plain Text |
1 This is an apple

用例2

Plain Text |
1 My sister is in the house not in the yard

用例3

Plain Text |
1 apple apple orange apple banana banana orange

用例4

Plain Text |
1 hello world hello code

用例5

Plain Text |
1 test test test example example code

用例6

```
Plain Text |  
1 car bike plane car car bike
```

用例7

```
Plain Text |  
1 apple banana cherry apple banana
```

用例8

```
Plain Text |  
1 abcd efg h ijk l mnop qrst uvwx yz
```

用例9

```
Plain Text |  
1 one two three four five six seven
```

用例10

```
Plain Text |  
1 zoo zoo zebra zebra apple apple apple
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [解题思路](#)
- [Java](#)
- [Python](#)

- JavaScript
- C++
- C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10



最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

给定一个字符串s，s包括以空格分隔的若干个单词，请对s进行如下处理后输出：

1、单词内部调整：对每个单词字母重新按字典序排序

2、单词间顺序调整：

- 统计每个单词出现的次数，并按次数降序排列
- 次数相同，按单词长度升序排列

- 次数和单词长度均相同，按字典升序排列

请输出处理后的字符串，每个单词以一个空格分隔

输入描述

一行字符串，每个字符取值范围：[a-zA-Z0-9]以及空格，[字符串长度](#)范围：[1, 1000]

输出描述

输出处理后的字符串，每个单词以一个空格分隔。

示例1

输入

```
▼ Plain Text |  
1 This is an apple
```

输出

```
▼ Plain Text |  
1 an is This aelpp
```

说明

示例2

输入

```
▼ Plain Text |  
1 My sister is in the house not in the yard
```

输出

```
▼ Plain Text |  
1 in in eht eht My is not adry ehosu eirsst
```

说明

解题思路

考察的是排序

1. 对每个单词内部进行字典序排序
2. 按照单词出现次数、单词长度、字典序升序排列

具体实现思路如下：

1. 使用分割字符串，得到每个单词
2. 对每个单词进行字典序排序，并存储到一个 List 中
3. 统计每个单词出现的次数，并存储到一个 Map 中
4. 使用对 List 进行排序，排序规则如下：
 - 次数不同，按照次数降序排列
 - 次数相同，长度不同，按照长度升序排列
 - 次数和长度都相同，按照字典序升序排列
5. 输出处理后的字符串，每个单词以一个空格分隔

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         String input = scanner.nextLine();
7         scanner.close();
8
9         // 使用空格分割字符串
10        String[] words = input.split(" "); // 分割字符串为单词
11
12        // 对每个单词内部进行字典序排序
13        for (int i = 0; i < words.length; i++) {
14            char[] chars = words[i].toCharArray();
15            Arrays.sort(chars);
16            words[i] = new String(chars);
17        }
18
19        // 统计每个单词出现的次数
20        Map<String, Integer> count = new HashMap<>();
21        for (String word : words) {
22            count.put(word, count.getOrDefault(word, 0) + 1);
23        }
24
25        // 进行排序 (使用简单的冒泡排序代替 Collections.sort)
26        for (int i = 0; i < words.length - 1; i++) {
27            for (int j = 0; j < words.length - 1 - i; j++) {
28                // 进行比较
29                if (shouldSwap(words[j], words[j + 1], count)) {
30                    // 交换
31                    String temp = words[j];
32                    words[j] = words[j + 1];
33                    words[j + 1] = temp;
34                }
35            }
36        }
37
38        // 输出处理后的字符串
39        StringBuilder sb = new StringBuilder();
40        for (String word : words) {
41            sb.append(word).append(" ");
42        }
43        System.out.println(sb.toString().trim());
44    }
45}
```

```
46     // 判断是否需要交换两个单词
47     private static boolean shouldSwap(String a, String b, Map<String, Integer> count) {
48         if (!count.get(a).equals(count.get(b))) {
49             // 次数不同, 按照次数降序排列
50             return count.get(a) < count.get(b);
51         } else if (a.length() != b.length()) {
52             // 次数相同, 长度不同, 按照长度升序排列
53             return a.length() > b.length();
54         } else {
55             // 次数和长度都相同, 按照字典序升序排列
56             return a.compareTo(b) > 0;
57         }
58     }
59 }
```

Python

```
1 import collections
2 import functools
3
4 input = input()
5 words = input.split(" ")
6
7 # 对每个单词内部进行字典序排序
8 words = ["".join(sorted(word)) for word in words]
9
10 # 统计每个单词出现的次数
11 count = collections.Counter(words)
12
13 # 按照要求排序
14 words = sorted(words, key=functools.cmp_to_key(lambda a, b: count[b] - count[a] if count[a] != count[b] else len(a) - len(b) if len(a) != len(b) else -1 if a < b else 1))
15
16 # 输出处理后的字符串
17 output = " ".join(words)
18 print(output)
```

JavaScript

```
1 const readline = require("readline");
2 const rl = readline.createInterface({
3     input: process.stdin,
4     output: process.stdout,
5 });
6
7 rl.on("line", (input) => {
8     const words = input.split(" ");
9
10    // 对每个单词内部进行字典序排序
11    const sortedWords = words.map((word) => word.split("").sort().join(""));
12
13    // 统计每个单词出现的次数
14    const count = sortedWords.reduce((acc, word) => {
15        acc[word] = (acc[word] || 0) + 1;
16        return acc;
17    }, {});
18
19    // 按照要求排序
20    sortedWords.sort((a, b) => {
21        if (count[b] !== count[a]) {
22            return count[b] - count[a]; // 按出现次数降序
23        }
24        if (a.length !== b.length) {
25            return a.length - b.length; // 按长度升序
26        }
27        return a < b ? -1 : 1; // 字典序升序
28    });
29
30    // 输出处理后的字符串
31    console.log(sortedWords.join(" "));
32    rl.close();
33});
```

C++

```
1 #include <iostream>
2 #include <algorithm>
3 #include <unordered_map>
4 #include <sstream>
5 #include <vector>
6
7 using namespace std;
8
9 int main() {
10     // 读入字符串
11     string input;
12     getline(cin, input);
13     // 使用 istringstream 分割字符串
14     istringstream iss(input);
15     string token;
16     vector<string> words;
17     while (getline(iss, token, ' ')) {
18         // 对每个单词内部进行字典序排序
19         sort(token.begin(), token.end());
20         words.push_back(token);
21     }
22
23     // 统计每个单词出现的次数
24     unordered_map<string, int> count;
25     for (string word : words) {
26         count[word]++;
27     }
28
29     // 按照要求排序
30     sort(words.begin(), words.end(), [&](const string& a, const string&
31 b) {
32         if (count[a] != count[b]) {
33             // 次数不同, 按照次数降序排列
34             return count[b] < count[a];
35         } else if (a.length() != b.length()) {
36             // 次数相同, 长度不同, 按照长度升序排列
37             return a.length() < b.length();
38         } else {
39             // 次数和长度都相同, 按照字典序升序排列
40             return a < b;
41         }
42     });
43
44     // 输出处理后的字符串
45     ostringstream oss;
```

```
45     for (string word : words) {
46         oss << word << " ";
47     }
48     cout << oss.str() << endl;
49
50     return 0;
51 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_WORDS 1000      // 定义最大单词数量
6 #define MAX_LENGTH 100      // 定义单词的最大长度
7
8 // 定义一个结构体，用于存储单词和其出现的次数
9 typedef struct {
10     char word[MAX_LENGTH]; // 存储单词
11     int count;             // 存储单词出现的次数
12 } WordCount;
13
14 // 字母排序函数
15 void sortString(char *str) {
16     int length = strlen(str); // 获取字符串的长度
17     // 使用 qsort 对字符串的字母进行排序
18     qsort(str, length, sizeof(char), (int (*)(const void *, const void
19 *)) strcmp);
20 }
21
22 // 比较函数，用于对 WordCount 结构体数组进行排序
23 int compare(const void *a, const void *b) {
24     WordCount *wordCountA = (WordCount *)a; // 将 void 指针转换为 WordCount
25     WordCount *wordCountB = (WordCount *)b; // 将 void 指针转换为 WordCount
26
27     // 先比较出现次数
28     if (wordCountA->count != wordCountB->count) {
29         return wordCountB->count - wordCountA->count; // 次数降序排列
30     } else {
31         // 如果出现次数相同，则比较单词长度
32         int lenDiff = strlen(wordCountA->word) - strlen(wordCountB->word);
33         if (lenDiff != 0) {
34             return lenDiff; // 长度升序排列
35         } else {
36             return strcmp(wordCountA->word, wordCountB->word); // 字典序排
37         }
38     }
39
40     int main() {
41         char input[1000]; // 定义一个字符数组用于存储输入的字符串
```

```

42 fgets(input, sizeof(input), stdin); // 从标准输入读取一行字符串
43
44 // 处理单词的数组
45 char *words[MAX_WORDS]; // 存储分割后的单词
46 int wordCount = 0; // 记录总共的单词数量
47
48 // 使用 strtok 函数分割单词，以空格和换行符为分隔符
49 char *token = strtok(input, " \n");
50 while (token != NULL) {
51     char *sortedWord = malloc(MAX_LENGTH); // 动态分配内存存储排序后的单词
52     strcpy(sortedWord, token); // 复制当前分割出的单词
53     sortString(sortedWord); // 对单词内部进行字典序排序
54     words[wordCount++] = sortedWord; // 将排序后的单词存入数组
55     token = strtok(NULL, " \n"); // 继续分割下一个单词
56 }
57
58 // 统计每个单词出现的次数
59 WordCount wordCounts[MAX_WORDS]; // 存储每个单词及其出现次数的数组
60 int uniqueCount = 0; // 记录不同单词的数量
61
62 // 遍历所有分割出的单词
63 for (int i = 0; i < wordCount; i++) {
64     int found = 0; // 标记是否找到了重复的单词
65     // 检查当前单词是否已经存在于 wordCounts 数组中
66     for (int j = 0; j < uniqueCount; j++) {
67         if (strcmp(wordCounts[j].word, words[i]) == 0) {
68             wordCounts[j].count++; // 如果找到，增加计数
69             found = 1; // 标记已找到
70             break;
71         }
72     }
73     // 如果没有找到该单词，则将其添加到 wordCounts 中
74     if (!found) {
75         strcpy(wordCounts[uniqueCount].word, words[i]); // 复制单词
76         wordCounts[uniqueCount].count = 1; // 初始化计数为 1
77         uniqueCount++; // 增加不同单词的计数
78     }
79 }
80
81 // 按出现次数对单词进行排序
82 qsort(wordCounts, uniqueCount, sizeof(WordCount), compare);
83
84 // 输出结果
85 for (int i = 0; i < uniqueCount; i++) {
86     // 根据单词的出现次数输出每个单词
87     for (int j = 0; j < wordCounts[i].count; j++) {
88         printf("%s ", wordCounts[i].word);
89     }

```

```
90     }
91     printf("\n"); // 输出换行
92
93     // 释放动态分配的内存
94     for (int i = 0; i < wordCount; i++) {
95         free(words[i]); // 释放每个排序后单词的内存
96     }
97
98     return 0; // 程序结束
99 }
```

完整用例

用例1

Plain Text |

```
1 This is an apple
```

用例2

Plain Text |

```
1 My sister is in the house not in the yard
```

用例3

Plain Text |

```
1 apple apple orange apple banana banana orange
```

用例4

Plain Text |

```
1 hello world hello code
```

用例5

Plain Text |

```
1 test test test example example code
```

用例6

```
1 car bike plane car car bike
```

Plain Text |

用例7

```
1 apple banana cherry apple banana
```

Plain Text |

用例8

```
1 abcd efg h ijk l mnop qrst uvwx yz
```

Plain Text |

用例9

```
1 one two three four five six seven
```

Plain Text |

用例10

```
1 zoo zebra apple
```

Plain Text |

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [解题思路](#)
- [Java](#)
- [Python](#)

- JavaScript
- C++
- C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10



| 来自: 华为OD机考2025B卷 – 字符串重新排序 (Java & Python& JS & C++ & C) –CSDN博客

| 来自: 华为OD机考2025B卷 – 字符串重新排序 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 荒岛求生（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

一个荒岛上有很多人，岛上只有一条路通往岛屿两端的港口，大家需要逃往两端的港口才可逃生。

假定每个人移动的速度一样，且只可选择向左或向右逃生。

若两个人相遇，则进行决斗，战斗力强的能够活下来，并损失掉与对方相同的战斗力；若战斗力相同，则两人同归于尽。

输入描述

给定一行非 0 整数数组，元素个数不超过30000；

正负表示逃生方向（正表示向右逃生，负表示向左逃生），绝对值表示战斗力，越左边的[数字表示](#)里左边港口越近，逃生方向相同的人永远不会发生决斗。

输出描述

能够逃生的人总数，没有人逃生输出0，输入异常时输出-1。

示例1

输入

```
1 5 10 8 -8 -5
```

输出

```
1 2
```

说明

| 第3个人和第4个人同归于尽，第2个人杀死第5个人并剩余5战斗力，第1个人没有遇到敌人。

解题思路

原题：[735. 行星碰撞](<https://leetcode.cn/problems/asteroid-collision/>)

唯一的区别，就是本题会自减对方的战斗力。

```
1 import java.util.ArrayList;
2 import java.util.List;
3 import java.util.Scanner;
4
5 public class Main {
6     // 定义一个名为asteroidCollision的方法，接受一个整数列表作为参数
7     public static int asteroidCollision(List<Integer> people) {
8         // 创建一个新的ArrayList，用于存储幸存者
9         List<Integer> survivors = new ArrayList<>();
10
11        // 遍历输入的整数列表
12        for (int person : people) {
13            // 如果person等于0，则返回-1
14            if (person == 0) {
15                return -1;
16            }
17
18            // 初始化一个布尔变量alive为true
19            boolean alive = true;
20
21            // 当alive为true且person小于0，且survivors不为空，且survivors列表中
22            // 最后一个元素大于0时，执行循环
23            while (alive && person < 0 && !survivors.isEmpty() && survivors.get(survivors.size() - 1) > 0) {
24                // 更新alive的值，判断当前person是否比survivors列表中最后一个元素
25                // 的相反数大
26                alive = survivors.get(survivors.size() - 1) < -person;
27
28                // 如果survivors列表中最后一个元素小于等于person的相反数，则移除该
29                // 元素
30                if (survivors.get(survivors.size() - 1) <= -person) {
31                    person = person + survivors.get(survivors.size() - 1);
32
33                    survivors.remove(survivors.size() - 1);
34                }
35            }
36
37        }
38
39        // 返回survivors列表的大小
40        return survivors.size();
```

```
41     }
42
43     public static void main(String[] args) {
44         Scanner scanner = new Scanner(System.in);
45
46         // 读取一行输入，并使用空格分隔字符串
47         String[] input = scanner.nextLine().split(" ");
48
49         // 创建一个整数列表，用于存储输入的整数
50         List<Integer> people = new ArrayList<>();
51
52         // 将输入的字符串数组转换为整数，并添加到people列表中
53         for (String s : input) {
54             people.add(Integer.parseInt(s));
55         }
56
57         // 如果people列表的大小大于30000，则输出-1
58         if (people.size() > 30000) {
59             System.out.println(-1);
60         } else {
61             // 调用asteroidCollision方法，并将结果输出
62             int result = asteroidCollision(people);
63             System.out.println(result);
64         }
65     }
66 }
```

Python

```
1 def asteroidCollision(people: list[int]) -> int:
2     survivors = []
3     for person in people:
4         if person == 0:
5             return -1
6         alive = True
7         # 当前人向左逃生，且有人向右逃生时进行决斗
8         while alive and person < 0 and survivors and survivors[-1] > 0:
9             # 决斗结果：当前人战斗力大于对手
10            alive = survivors[-1] < - person
11            # 如果战斗力相等或当前人战斗力更大，移除对手
12            if survivors[-1] <= -person:
13                person = person + survivors[-1]
14
15                survivors.pop()
16            else:
17                survivors[-1] = survivors[-1] + person
18                print(survivors[-1])
19            # 如果当前人仍然存活，将其添加到逃生者列表
20            if alive:
21                survivors.append(person)
22        return len(survivors)
23
24 try:
25     # 从输入获取人员列表
26     people = list(map(int, input().split()))
27
28     # 检查输入是否异常
29     if len(people) > 30000:
30         raise ValueError("输入异常")
31
32     # 调用函数并输出结果
33     result = asteroidCollision(people)
34     print(result)
35 except ValueError as e:
36     print(-1)
```

JavaScript

```
1 function asteroidCollision(people) {
2     // 创建一个空数组，用于存储幸存者
3     const survivors = [];
4
5     // 遍历输入的整数数组
6     for (let person of people) {
7         // 如果person等于0，则返回-1
8         if (person === 0) {
9             return -1;
10        }
11
12         // 初始化一个布尔变量alive为true
13         let alive = true;
14
15         // 当alive为true且person小于0，且survivors长度大于0，且survivors数组中
16         // 最后一个元素大于0时，执行循环
17         while (alive && person < 0 && survivors.length > 0 && survivors[survivors.length - 1] > 0) {
18             // 更新alive的值，判断当前person是否比survivors数组中最后一个元素的相
19             // 反数大
20             alive = survivors[survivors.length - 1] < -person;
21
22             // 如果survivors数组中最后一个元素小于等于person的相反数，则移除该元素
23             if (survivors[survivors.length - 1] <= -person) {
24                 person = person + survivors[survivors.length - 1];
25                 survivors.pop();
26             }
27
28             // 如果alive为true，则将person添加到survivors数组中
29             if (alive) {
30                 survivors.push(person);
31             }
32
33             // 返回survivors数组的长度
34             return survivors.length;
35         }
36
37     // 导入readline模块
38     const readline = require('readline');
39     // 创建一个readline接口实例
40     const rl = readline.createInterface({
41         input: process.stdin,
42         output: process.stdout
43     });
44 }
```

```
43  });
44
45 // 当接收到一行输入时，执行以下操作
46 rl.on('line', (input) => {
47     // 将输入的字符串以空格分隔，并将每个子字符串转换为数字，存储在people数组中
48     const people = input.split(' ').map(Number);
49
50     // 如果people数组的长度大于30000，则输出-1
51     if (people.length > 30000) {
52         console.log(-1);
53     } else {
54         // 调用asteroidCollision函数，并将结果输出
55         const result = asteroidCollision(people);
56         console.log(result);
57     }
58
59     // 关闭readline接口
60     rl.close();
61 });
});
```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4 #include <sstream>
5
6 using namespace std;
7
8 // 定义一个名为asteroidCollision的函数，接受一个整数向量作为参数
9 int asteroidCollision(vector<int>& people) {
10     // 创建一个新的vector，用于存储幸存者
11     vector<int> survivors;
12
13     // 遍历输入的整数向量
14     for (int person : people) {
15         // 如果person等于0，则返回-1
16         if (person == 0) {
17             return -1;
18         }
19
20         // 初始化一个布尔变量alive为true
21         bool alive = true;
22
23         // 当alive为true且person小于0，且survivors不为空，且survivors向量中最后一个元素大于0时，执行循环
24         while (alive && person < 0 && !survivors.empty() && survivors.back() > 0) {
25             // 更新alive的值，判断当前person是否比survivors向量中最后一个元素的相反数大
26             alive = survivors.back() < -person;
27
28             // 如果survivors向量中最后一个元素小于等于person的相反数，则移除该元素
29             if (survivors.back() <= -person) {
30                 person = person + survivors.back();
31                 survivors.pop_back();
32             }
33         }
34
35         // 如果alive为true，则将person添加到survivors向量中
36         if (alive) {
37             survivors.push_back(person);
38         }
39     }
40
41     // 返回survivors向量的大小
42     return survivors.size();
```

```
43 }
44
45 int main() {
46     string input;
47     getline(cin, input);
48     stringstream ss(input);
49
50     // 创建一个整数向量，用于存储输入的整数
51     vector<int> people;
52     int num;
53
54     // 将输入的字符串转换为整数，并添加到people向量中
55     while (ss >> num) {
56         people.push_back(num);
57     }
58
59     // 如果people向量的大小大于30000，则输出-1
60     if (people.size() > 30000) {
61         cout << -1 << endl;
62     } else {
63         // 调用asteroidCollision函数，并将结果输出
64         int result = asteroidCollision(people);
65         cout << result << endl;
66     }
67
68     return 0;
69 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4 #include <string.h>
5
6 #define MAX_PEOPLE 30000
7
8 // 定义一个方法 asteroidCollision, 接受一个整数数组 people 及其大小 n 作为参数
9 int asteroidCollision(int *people, int n) {
10     // 定义一个数组 survivors 来存储幸存者
11     int survivors[MAX_PEOPLE];
12     int survivors_size = 0; // 幸存者列表的大小
13
14     // 遍历输入的整数数组
15     for (int i = 0; i < n; i++) {
16         int person = people[i];
17
18         // 如果 person 等于 0, 返回 -1, 表示输入异常
19         if (person == 0) {
20             return -1;
21         }
22
23         bool alive = true;
24
25         // 当 alive 为 true 且 person 小于 0 且 survivors 不为空且 survivors
26         // 列表中最后一个元素大于 0 时, 执行循环
27         while (alive && person < 0 && survivors_size > 0 && survivors[survivors_size - 1] > 0) {
28             // 更新 alive 的值, 判断当前 person 是否比 survivors 列表中最后一个
29             // 元素的相反数大
30             alive = survivors[survivors_size - 1] < -person;
31
32             // 如果 survivors 列表中最后一个元素小于等于 person 的相反数, 则移除该
33             // 元素
34             if (survivors[survivors_size - 1] <= -person) {
35                 person += survivors[survivors_size - 1]; // 更新 person 的
36                 // 战斗力
37                 survivors_size--; // 移除最后一个幸存者
38             }
39         }
40
41         // 如果 alive 为 true, 则将 person 添加到 survivors 列表中
42         if (alive) {
43             survivors[survivors_size++] = person;
44         }
45     }
46 }
```

```
41     }
42
43     // 返回 survivors 列表的大小
44     return survivors_size;
45 }
46
47 int main() {
48     char input[500000]; // 假设输入的最大长度不超过 500000 个字符
49     int people[MAX_PEOPLE];
50     int count = 0;
51
52     // 读取一行输入
53     if (fgets(input, sizeof(input), stdin) == NULL) {
54         printf("-1\n");
55         return 0;
56     }
57
58     // 将输入字符串分割为整数并存储在 people 数组中
59     char *token = strtok(input, " ");
60     while (token != NULL && count < MAX_PEOPLE) {
61         people[count++] = atoi(token);
62         token = strtok(NULL, " ");
63     }
64
65     // 如果输入的数量超过 30000, 输出 -1
66     if (count > MAX_PEOPLE) {
67         printf("-1\n");
68     } else {
69         // 调用 asteroidCollision 方法, 并将结果输出
70         int result = asteroidCollision(people, count);
71         printf("%d\n", result);
72     }
73
74     return 0;
75 }
```

完整用例

用例1

```
▼ Plain Text |  
1 1 5
```

用例2

Plain Text |

1 1 -5

用例3

Plain Text |

1 2 5 -5

用例4

Plain Text |

1 3 5 10 -5

用例5

Plain Text |

1 3 5 -10 5

用例6

Plain Text |

1 4 5 -10 5 -5

用例7

Plain Text |

1 5 5 -10 5 -5 10

用例8

Plain Text |

1 5 5 -10 5 -5 10 -10

用例9

```
Plain Text |  
1 6 5 -10 5 -5 10 -10 5
```

用例10

```
Plain Text |  
1 0
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [解题思路](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)
- [完整用例](#)
 - [用例1](#)
 - [用例2](#)
 - [用例3](#)
 - [用例4](#)
 - [用例5](#)
 - [用例6](#)
 - [用例7](#)
 - [用例8](#)
 - [用例9](#)
 - [用例10](#)

机考真题 华为OD

No No No



CSDN @算法大师

| 来自: 华为OD机考2025B卷 – 荒岛求生 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机试2025B卷 - 返回矩阵中非1的元素、个数/数值同化（Java & Python& JS & C++ & C） -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

题目描述

存在一个 $m \times n$ 的二维数组，其成员取值范围为0, 1, 2。

其中值为1的元素具备同化特性，每经过1S，将上下左右值为0的元素同化为1。

而值为2的元素，免疫同化。

将数组所有成员随机初始化为0或2，再将矩阵的[0, 0]元素修改成1，在经过足够长的时间后求矩阵中有多少个元素是0或2（即0和2数量之和）。

输入描述

输入的前两个数字是矩阵大小。后面是数字矩阵内容。

输出描述

返回矩阵中非1的元素个数。

输入

```
1 4 4
2 0 0 0 0
3 0 2 2 2
4 0 2 0 0
5 0 2 0 0
```

输出

```

1 4 4
2 0 0 0 0
3 0 2 2 2
4 0 2 0 0
5 0 2 0 0

```

说明

输入数字前两个数字是矩阵大小。后面的数字是矩阵内容。

起始位置(0,0)被修改为1后，最终只能同化矩阵为：

```

1 1 1 1
1 2 2 2
1 2 0 0
1 2 0 0

```

所以矩阵中非1的元素个数为9

解题思路

题目的要求是模拟一个在二维数组中进行的“同化”过程。

1. 二维数组的初始化：

- 给定一个大小为 $m \times n$ 的二维数组，每个元素的取值范围为 0、1 或 2。
- 值为 1 的元素表示同化源，会将相邻的 0 元素同化为 1。
- 值为 2 的元素对同化免疫，无法被同化。

2. 同化过程：

- 从矩阵的左上角元素 [0, 0] 开始，将其设为 1，作为初始同化源。
- 在接下来的每秒钟内，所有为 1 的元素会尝试同化它的上、下、左、右相邻的 0 元素，将其变为 1。
- 2 元素则不会被同化，也不会对周围元素产生影响。

3. 目标：

- 经过足够长时间的同化过程后，矩阵中会有一部分元素被同化成 1，无法被同化的 0 和 2 元素将保持原状。
- 最后，计算矩阵中非 1 元素的个数（即 0 和 2 的数量）。

Java

```
1 import java.util.Scanner;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 输入矩阵大小
8         int rows = scanner.nextInt();
9         int cols = scanner.nextInt();
10
11         // 创建并初始化矩阵
12         int[][] matrix = new int[rows][cols];
13         for (int i = 0; i < rows; i++) {
14             for (int j = 0; j < cols; j++) {
15                 matrix[i][j] = scanner.nextInt();
16             }
17         }
18
19         // 标记数组，避免重复访问
20         boolean[][] visited = new boolean[rows][cols];
21
22         // 使用 DFS 从 (0, 0) 开始同化区域
23         matrix[0][0] = 1; // 将起始点设为 1
24         dfs(matrix, 0, 0, rows, cols, visited);
25
26         // 计算非 1 元素的个数
27         int nonAssimilatedCount = 0;
28         for (int i = 0; i < rows; i++) {
29             for (int j = 0; j < cols; j++) {
30                 if (matrix[i][j] != 1) {
31                     nonAssimilatedCount++;
32                 }
33             }
34         }
35
36         System.out.println(nonAssimilatedCount);
37     }
38
39     // DFS 方法
40     private static void dfs(int[][] matrix, int x, int y, int rows, int cols, boolean[][] visited) {
41         // 定义四个方向偏移量
42         int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
43
44         visited[x][y] = true; // 标记当前点为已访问
```

```
45
46     for (int[] direction : directions) {
47         int newX = x + direction[0];
48         int newY = y + direction[1];
49
50         // 检查新坐标是否在范围内、值为0, 并且未被访问
51         if (isValid(newX, newY, rows, cols, matrix, visited)) {
52             matrix[newX][newY] = 1; // 同化新坐标
53             dfs(matrix, newX, newY, rows, cols, visited); // 递归同化相
54             连区域
55         }
56     }
57
58     // 辅助方法: 检查坐标是否合法、矩阵值为0且未被访问
59     private static boolean isValid(int x, int y, int rows, int cols, int[]
60     [] matrix, boolean[][] visited) {
61         return x >= 0 && x < rows && y >= 0 && y < cols && matrix[x][y] ==
62         0 && !visited[x][y];
63     }
64 }
```

Python

```

1 def dfs(matrix, x, y, rows, cols, visited):
2     # 定义四个方向偏移量: 上、下、左、右
3     directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
4     visited[x][y] = True  # 标记当前点为已访问
5
6     for direction in directions:
7         newX = x + direction[0]
8         newY = y + direction[1]
9
10    # 检查新坐标是否合法且未访问, 且矩阵值为0
11    if is_valid(newX, newY, rows, cols, matrix, visited):
12        matrix[newX][newY] = 1  # 同化新坐标
13        dfs(matrix, newX, newY, rows, cols, visited)  # 递归调用
14
15 def is_valid(x, y, rows, cols, matrix, visited):
16     # 检查坐标范围、矩阵值和访问状态
17     return 0 <= x < rows and 0 <= y < cols and matrix[x][y] == 0 and not v
isited[x][y]
18
19 def main():
20     # 输入矩阵大小
21     rows, cols = map(int, input().split())
22     # 初始化矩阵
23     matrix = [list(map(int, input().split())) for _ in range(rows)]
24
25     # 初始化标记数组, 防止重复访问
26     visited = [[False for _ in range(cols)] for _ in range(rows)]
27
28     # 将起点设为1
29     matrix[0][0] = 1
30     dfs(matrix, 0, 0, rows, cols, visited)
31
32     # 统计非1元素的数量
33     non_assimilated_count = sum(1 for i in range(rows) for j in range(col
s) if matrix[i][j] != 1)
34     print(non_assimilated_count)
35
36 if __name__ == "__main__":
37     main()

```

JavaScript

```
1 const readline = require('readline');
2
3 function dfs(matrix, x, y, rows, cols, visited) {
4     // 定义四个方向偏移量: 上、下、左、右
5     const directions = [[-1, 0], [1, 0], [0, -1], [0, 1]];
6     visited[x][y] = true; // 标记当前点为已访问
7
8     for (const [dx, dy] of directions) {
9         const newX = x + dx;
10        const newY = y + dy;
11
12        // 检查新坐标是否合法且未访问, 且矩阵值为0
13        if (isValid(newX, newY, rows, cols, matrix, visited)) {
14            matrix[newX][newY] = 1; // 同化新坐标
15            dfs(matrix, newX, newY, rows, cols, visited); // 递归调用
16        }
17    }
18 }
19
20 function isValid(x, y, rows, cols, matrix, visited) {
21     // 检查坐标范围、矩阵值和访问状态
22     return x >= 0 && x < rows && y >= 0 && y < cols && matrix[x][y] === 0
&& !visited[x][y];
23 }
24
25 const rl = readline.createInterface({
26     input: process.stdin,
27     output: process.stdout
28 });
29
30 let input = [];
31 rl.on('line', (line) => {
32     input.push(line);
33 }).on('close', () => {
34     const [rows, cols] = input[0].split(' ').map(Number);
35     const matrix = input.slice(1, rows + 1).map(row => row.split(' ').map(
Number));
36     const visited = Array.from({ length: rows }, () => Array(cols).fill(
false));
37
38     // 将起点设为1
39     matrix[0][0] = 1;
40     dfs(matrix, 0, 0, rows, cols, visited);
41
42     // 统计非1元素的数量
```

```
43     let nonAssimilatedCount = 0;
44     for (let i = 0; i < rows; i++) {
45         for (let j = 0; j < cols; j++) {
46             if (matrix[i][j] !== 1) {
47                 nonAssimilatedCount++;
48             }
49         }
50     }
51     console.log(nonAssimilatedCount);
52 });

});
```

C++

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 void dfs(vector<vector<int>> &matrix, int x, int y, int rows, int cols, ve
ctor<vector<bool>> &visited) {
6     // 定义四个方向偏移量: 上、下、左、右
7     int directions[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
8     visited[x][y] = true; // 标记当前点为已访问
9
10    for (int i = 0; i < 4; i++) {
11        int newX = x + directions[i][0];
12        int newY = y + directions[i][1];
13
14        // 检查新坐标是否合法且未访问, 且矩阵值为0
15        if (newX >= 0 && newX < rows && newY >= 0 && newY < cols && matrix
[newX][newY] == 0 && !visited[newX][newY]) {
16            matrix[newX][newY] = 1; // 同化新坐标
17            dfs(matrix, newX, newY, rows, cols, visited); // 递归调用
18        }
19    }
20 }
21
22 int main() {
23     int rows, cols;
24     cin >> rows >> cols;
25
26     // 初始化矩阵
27     vector<vector<int>> matrix(rows, vector<int>(cols));
28     for (int i = 0; i < rows; i++) {
29         for (int j = 0; j < cols; j++) {
30             cin >> matrix[i][j];
31         }
32     }
33
34     // 初始化标记数组, 防止重复访问
35     vector<vector<bool>> visited(rows, vector<bool>(cols, false));
36
37     // 将起点设为1
38     matrix[0][0] = 1;
39     dfs(matrix, 0, 0, rows, cols, visited);
40
41     // 统计非1元素的数量
42     int nonAssimilatedCount = 0;
43     for (int i = 0; i < rows; i++) {
```

```
44     for (int j = 0; j < cols; j++) {
45         if (matrix[i][j] != 1) {
46             nonAssimilatedCount++;
47         }
48     }
49 }
50 cout << nonAssimilatedCount << endl;
51 return 0;
52 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdbool.h>
3
4 void dfs(int matrix[][][100], int x, int y, int rows, int cols, bool visited
[])[100]) {
5     // 定义四个方向偏移量: 上、下、左、右
6     int directions[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
7     visited[x][y] = true; // 标记当前点为已访问
8
9     for (int i = 0; i < 4; i++) {
10         int newX = x + directions[i][0];
11         int newY = y + directions[i][1];
12
13         // 检查新坐标是否合法且未访问, 且矩阵值为0
14         if (newX >= 0 && newX < rows && newY >= 0 && newY < cols && matrix
15 [newX][newY] == 0 && !visited[newX][newY]) {
16             matrix[newX][newY] = 1; // 同化新坐标
17             dfs(matrix, newX, newY, rows, cols, visited); // 递归调用
18         }
19     }
20 }
21
22 int main() {
23     int rows, cols;
24     scanf("%d %d", &rows, &cols);
25
26     // 初始化矩阵
27     int matrix[100][100];
28     for (int i = 0; i < rows; i++) {
29         for (int j = 0; j < cols; j++) {
30             scanf("%d", &matrix[i][j]);
31         }
32     }
33
34     // 初始化标记数组, 防止重复访问
35     bool visited[100][100] = {false};
36
37     // 将起点设为1
38     matrix[0][0] = 1;
39     dfs(matrix, 0, 0, rows, cols, visited);
40
41     // 统计非1元素的数量
42     int nonAssimilatedCount = 0;
43     for (int i = 0; i < rows; i++) {
44         for (int j = 0; j < cols; j++) {
```

```
44         if (matrix[i][j] != 1) {
45             nonAssimilatedCount++;
46         }
47     }
48 }
49
50 printf("%d\n", nonAssimilatedCount);
51 return 0;
52 }
```

完整用例

用例1

```
1 4 4
2 0 0 0 0
3 0 2 2 2
4 0 2 0 0
5 0 2 0 0
```

用例2

```
1 3 3
2 0 2 0
3 0 0 0
4 2 0 2
```

用例3

```
1 5 5
2 0 0 0 0 0
3 0 2 2 2 0
4 0 2 0 0 0
5 0 2 0 0 0
6 0 0 0 0 0
```

用例4

```
▼ Plain Text |  
1 2 2  
2 0 0  
3 0 2
```

用例5

```
▼ Plain Text |  
1 4 4  
2 0 0 0 0  
3 2 2 2 2  
4 0 0 0 0  
5 0 0 0 2
```

用例6

```
▼ Plain Text |  
1 3 4  
2 2 2 2 2  
3 2 0 0 2  
4 2 2 2 2
```

用例7

```
▼ Plain Text |  
1 4 5  
2 0 0 0 0 0  
3 0 2 2 0 0  
4 0 0 0 2 2  
5 0 0 0 0 0
```

用例8

```
1 5 5
2 0 0 0 0 0
3 0 2 2 2 0
4 0 2 0 2 0
5 0 2 2 2 0
6 0 0 0 0 0
```

Plain Text |

用例9

```
1 3 4
2 2 2 2
3 2 0 0 2
4 2 2 2 2
```

用例10

```
1 5 5
2 2 2 2 2
3 2 0 0 0 2
4 2 0 2 0 2
5 2 0 0 0 2
6 2 2 2 2 2
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [解题思路](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)

- 完整用例
- 用例1
- 用例2
- 用例3
- 用例4
- 用例5
- 用例6
- 用例7
- 用例8
- 用例9
- 用例10



来自: 华为OD机试2025B卷 – 返回矩阵中非1的元素、个数/数值同化 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 查找一个有向网络的头节点和尾节点 (Java & Python& JS & C++ & C) - CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

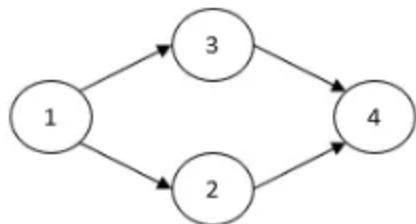
2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

给定一个[有向图](#)，图中可能包含有环，图使用二维矩阵表示，每一行的第一列表示起始节点，第二列表示终止节点，如 [0, 1] 表示从 0 到 1 的路径。

每个节点用正整数表示。求这个数据的首节点与尾节点，题目给的用例会是一个首节点，但可能存在多个尾节点。同时图中可能含有环。如果图中含有环，返回 [-1]。

说明：入度为0是首节点，出度为0是尾节点。



输入描述

第一行为后续输入的键值对数量N ($N \geq 0$)

第二行为 $2N$ 个数字。每两个为一个起点，一个终点。

输出描述

输出一行头节点和尾节点。如果有多个尾节点，按从小到大的顺序输出

备注

- 如果图有环，输出为 -1
- 所有输入均合法，不会出现不配对的数据

用例1

输入

```
1 4  
2 0 1 0 2 1 2 2 3
```

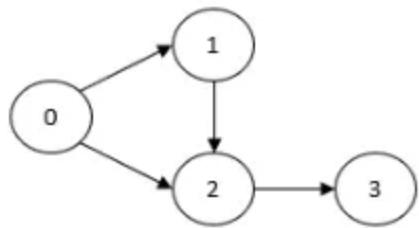
Plain Text |

输出

```
1 0 3
```

Plain Text |

说明



用例2

输入

```
1 2  
2 0 1 0 2
```

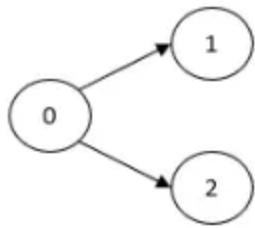
Plain Text |

输出

```
1 0 1 2
```

Plain Text |

说明



解题思路

环的检测

1. DFS实现:

- 为了进行DFS，首先构建了图的邻接表表示。这是通过将边信息转换成每个节点的邻接节点列表来完成的。
- 然后，对每个节点进行DFS遍历。
- 在遍历过程中，维护两个集合：`visited` 和 `recStack`。
 - `visited` 用于记录已经访问过的节点。
 - `recStack` 用于记录当前递归调用栈中的节点。

2. 检测环的逻辑:

- 在DFS遍历中，如果当前节点已在 `recStack` 中，则表示找到了一个环，因为这意味着在遍历过程中回到了正在访问的节点。
- 如果节点在 `visited` 中但不在 `recStack` 中，这意味着节点已被访问且没有形成环。
- 对每个邻接节点重复这个过程。

3. 结束条件:

- 如果在任何节点的DFS遍历中检测到环，立即返回 `true`，表示图中存在环。
- 如果所有节点都被遍历过，且没有检测到环，则返回 `false`。

确定起点和终点

1. 起点和终点的定义:

- 起点是指没有入度的节点，即没有其他节点指向它的节点。
- 终点是指没有出度的节点，即它不指向任何其他节点的节点。

2. 查找方法:

- 遍历所有节点，检查它们的入度和出度。
- 对于每个节点，如果它在 `inDegree` 映射中没有记录，那么它就是起点。
- 如果它在 `outDegree` 映射中没有记录，那么它就是终点。

3. 存储结果:

- 结果数组首先存储起点，然后是所有终点。
- 这个数组最后返回作为方法的输出。

```
1 #include <iostream>
2 #include <string>
3 #include <sstream>
4 #include <vector>
5 #include <map>
6 #include <set>
7 // 递归检测图中是否有环
8 bool detectCycle(const std::map<int, std::vector<int>>& graph, int node,
9     std::set<int>& visited, std::set<int>& recStack) {
10    if (recStack.find(node) != recStack.end()) {
11        return true;
12    }
13    if (visited.find(node) != visited.end()) {
14        return false;
15    }
16    visited.insert(node);
17    recStack.insert(node);
18
19    const std::vector<int>& neighbors = graph.at(node);
20    for (int neighbor : neighbors) {
21        if (detectCycle(graph, neighbor, visited, recStack)) {
22            return true;
23        }
24    }
25
26    recStack.erase(node);
27    return false;
28 }
29 // 检测图中是否有环
30 bool hasCycle(const std::set<int>& nodes, const std::vector<int>& edges)
31 {
32     std::map<int, std::vector<int>> graph;
33     for (int node : nodes) {
34         graph[node] = std::vector<int>();
35     }
36
37     for (size_t i = 0; i < edges.size(); i += 2) {
38         graph[edges[i]].push_back(edges[i + 1]);
39     }
40
41     std::set<int> visited;
42     std::set<int> recStack;
43
44     for (int node : nodes) {
```

```

44         if (detectCycle(graph, node, visited, recStack)) {
45             return true;
46         }
47     }
48
49     return false;
50 }
51 // 寻找起点和终点节点
52 std::vector<int> findStartAndEndNodes(const std::vector<int>& edges) {
53     std::map<int, int> inDegree, outDegree;
54     std::set<int> nodes;
55
56     // 构建入度和出度的映射
57     for (size_t i = 0; i < edges.size(); i += 2) {
58         int start = edges[i];
59         int end = edges[i + 1];
60         nodes.insert(start);
61         nodes.insert(end);
62         inDegree[end]++;
63         outDegree[start]++;
64     }
65
66     // 检测是否有环
67     if (hasCycle(nodes, edges)) {
68         return {-1};
69     }
70
71     // 寻找起点和终点
72     std::vector<int> endNodes;
73     int startNode = -1;
74     for (int node : nodes) {
75         if (inDegree.find(node) == inDegree.end()) {
76             startNode = node;
77         }
78         if (outDegree.find(node) == outDegree.end()) {
79             endNodes.push_back(node);
80         }
81     }
82
83     std::vector<int> result = {startNode};
84     result.insert(result.end(), endNodes.begin(), endNodes.end());
85     return result;
86 }
87
88
89
90
91

```

```
92 // 主函数
93 int main() {
94     std::string line;
95     std::getline(std::cin, line); // 读取N, 但在这个程序中不使用
96     std::getline(std::cin, line); // 读取边的数据
97
98     std::istringstream iss(line);
99     std::vector<int> edges;
100    int edge;
101    while (iss >> edge) {
102        edges.push_back(edge);
103    }
104
105    std::vector<int> result = findStartAndEndNodes(edges);
106    for (int num : result) {
107        std::cout << num << ' ';
108    }
109    std::cout << std::endl;
110
111    return 0;
112 }
```

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         // 读取边的数量
7         int N = scanner.nextInt();
8         // 存储所有边的数组
9         int[] edges = new int[2 * N];
10        for (int i = 0; i < 2 * N; i++) {
11            // 读取每条边的起点和终点
12            edges[i] = scanner.nextInt();
13        }
14
15        // 找出起点和终点
16        int[] result = findStartAndEndNodes(edges);
17        StringBuilder sb = new StringBuilder();
18        for (int i = 0; i < result.length; i++) {
19            // 构建输出字符串
20            sb.append(result[i]);
21            if (i < result.length - 1) {
22                sb.append(" ");
23            }
24        }
25        // 打印结果
26        System.out.println(sb.toString());
27    }
28
29    // 方法: 找出起点和终点
30    private static int[] findStartAndEndNodes(int[] edges) {
31        // 存储每个节点的入度
32        Map<Integer, Integer> inDegree = new HashMap<>();
33        // 存储每个节点的出度
34        Map<Integer, Integer> outDegree = new HashMap<>();
35        // 存储所有节点
36        Set<Integer> nodes = new HashSet<>();
37
38        for (int i = 0; i < edges.length; i += 2) {
39            int start = edges[i];
40            int end = edges[i + 1];
41
42            // 添加节点
43            nodes.add(start);
44            nodes.add(end);
45        }
46
47        // 遍历所有节点，找出起点和终点
48        for (int node : nodes) {
49            int inDegreeCount = 0;
50            int outDegreeCount = 0;
51
52            for (int neighbor : nodes) {
53                if (inDegree.get(neighbor) > 0) {
54                    inDegreeCount++;
55                }
56                if (outDegree.get(neighbor) > 0) {
57                    outDegreeCount++;
58                }
59            }
60
61            if (inDegreeCount == 0) {
62                result[0] = node;
63            }
64            if (outDegreeCount == 0) {
65                result[1] = node;
66            }
67        }
68
69        return result;
70    }
71}
```

```

46         // 计算入度和出度
47         inDegree.put(end, inDegree.getOrDefault(end, 0) + 1);
48         outDegree.put(start, outDegree.getOrDefault(start, 0) + 1);
49     }
50
51     // 检查是否有环
52     if (hasCycle(nodes, edges)) {
53         return new int[]{-1};
54     }
55
56     // 找出终点
57     List<Integer> endNodes = new ArrayList<>();
58     // 起点
59     int startNode = -1;
60     for (int node : nodes) {
61         // 如果一个节点没有入度, 它是起点
62         if (!inDegree.containsKey(node)) {
63             startNode = node;
64         }
65         // 如果一个节点没有出度, 它是终点
66         if (!outDegree.containsKey(node)) {
67             endNodes.add(node);
68         }
69     }
70
71     // 构建结果数组
72     int[] result = new int[endNodes.size() + 1];
73     result[0] = startNode;
74     for (int i = 0; i < endNodes.size(); i++) {
75         result[i + 1] = endNodes.get(i);
76     }
77
78     return result;
79 }
80
81     // 方法: 检查是否有环
82     private static boolean hasCycle(Set<Integer> nodes, int[] edges) {
83         // 构建图
84         Map<Integer, List<Integer>> graph = new HashMap<>();
85         for (int node : nodes) {
86             graph.put(node, new ArrayList<>());
87         }
88         for (int i = 0; i < edges.length; i += 2) {
89             graph.get(edges[i]).add(edges[i + 1]);
90         }
91
92         // 记录已访问和回溯栈的节点
93         Set<Integer> visited = new HashSet<>();

```

```

94     Set<Integer> recStack = new HashSet<>();
95
96     // 检查每个节点是否有环
97     for (int node : nodes) {
98         if (detectCycle(graph, node, visited, recStack)) {
99             return true;
100        }
101    }
102
103    return false;
104 }
105
106 // 辅助方法: 递归检查环
107 private static boolean detectCycle(Map<Integer, List<Integer>> graph,
108 int node,
109 > recStack) {
110
111     // 如果节点在回溯栈中, 表示有环
112     if (recStack.contains(node)) {
113         return true;
114     }
115     // 如果已经访问过, 不需要再次访问
116     if (visited.contains(node)) {
117         return false;
118     }
119     // 标记节点为已访问, 并加入回溯栈
120     visited.add(node);
121     recStack.add(node);
122
123     // 检查相邻节点
124     List<Integer> neighbors = graph.get(
125         t(node));
126
127     if (detectCycle(graph, neighbor, v
128         isited, recStack)) {
129
130         // 从回溯栈中移除节点
131         recStack.remove(node);
132         return true;
133     }
134 }

```

javaScript

```
1 // 导入必要的模块
2 const readline = require('readline');
3
4 // 创建读取行的接口
5 const rl = readline.createInterface({
6     input: process.stdin,
7     output: process.stdout
8 });
9
10
11 rl.on('line', (N) => {
12     rl.on('line', (edgesString) => {
13         const edges = edgesString.split(' ').map(Number);
14         const result = findStartAndEndNodes(edges);
15         console.log(result.join(' '));
16         rl.close();
17     });
18 });
19
20
21 // 寻找起点和终点节点
22 function findStartAndEndNodes(edges) {
23     // 存储入度和出度
24     let inDegree = new Map();
25     let outDegree = new Map();
26     let nodes = new Set();
27
28     // 构建入度和出度的映射
29     for (let i = 0; i < edges.length; i += 2) {
30         let start = edges[i];
31         let end = edges[i + 1];
32         nodes.add(start);
33         nodes.add(end);
34         inDegree.set(end, (inDegree.get(end) || 0) + 1);
35         outDegree.set(start, (outDegree.get(start) || 0) + 1);
36     }
37
38     // 检测是否有环
39     if (hasCycle(nodes, edges)) {
40         return [-1];
41     }
42
43     // 寻找起点和终点
44     let endNodes = [];
45     let startNode = -1;
```

```

46     for (let node of nodes) {
47         if (!inDegree.has(node)) {
48             startNode = node;
49         }
50         if (!outDegree.has(node)) {
51             endNodes.push(node);
52         }
53     }
54
55     return [startNode, ...endNodes];
56 }
57
58 // 检测图中是否有环
59 function hasCycle(nodes, edges) {
60     let graph = new Map();
61     nodes.forEach(node => graph.set(node, []));
62
63     for (let i = 0; i < edges.length; i += 2) {
64         graph.get(edges[i]).push(edges[i + 1]);
65     }
66
67     let visited = new Set();
68     let recStack = new Set();
69
70     for (let node of nodes) {
71         if (detectCycle(graph, node, visited, recStack)) {
72             return true;
73         }
74     }
75
76     return false;
77 }
78
79 // 递归检测图中是否有环
80 function detectCycle(graph, node, visited, recStack) {
81     if (recStack.has(node)) {
82         return true;
83     }
84     if (visited.has(node)) {
85         return false;
86     }
87
88     visited.add(node);
89     recStack.add(node);
90
91     let neighbors = graph.get(node);
92     for (let neighbor of neighbors) {
93         if (detectCycle(graph, neighbor, visited, recStack)) {

```

```
94         return true;
95     }
96 }
98     recStack.delete(node);
99     return false;
100 }
```

Python

```
1 def find_start_and_end_nodes(edges):
2     """
3         寻找起点和终点节点。
4
5         参数:
6         edges - 边的数组。
7
8         返回:
9         起点和所有终点的数组。
10    """
11    in_degree = {} # 用于存储每个节点的入度
12    out_degree = {} # 用于存储每个节点的出度
13    nodes = set() # 存储所有节点
14
15    # 遍历边, 构建入度和出度的映射
16    for i in range(0, len(edges), 2):
17        start, end = edges[i], edges[i + 1]
18        nodes.add(start)
19        nodes.add(end)
20        in_degree[end] = in_degree.get(end, 0) + 1
21        out_degree[start] = out_degree.get(start, 0) + 1
22
23    # 检测是否有环
24    if has_cycle(nodes, edges):
25        return [-1]
26
27    # 寻找起点和终点
28    end_nodes = []
29    start_node = -1
30    for node in nodes:
31        if node not in in_degree:
32            start_node = node
33        if node not in out_degree:
34            end_nodes.append(node)
35
36    return [start_node] + end_nodes
37
38 def has_cycle(nodes, edges):
39     """
40         检测图中是否有环。
41
42         参数:
43         nodes - 节点集合。
44         edges - 边的数组。
45     """
```

```

46     返回：
47     布尔值，表示图中是否有环。
48     """
49     graph = {node: [] for node in nodes}
50     for i in range(0, len(edges), 2):
51         graph[edges[i]].append(edges[i + 1])
52
53     visited = set()
54     rec_stack = set()
55
56     for node in nodes:
57         if detect_cycle(graph, node, visited, rec_stack):
58             return True
59
60     return False
61
62 def detect_cycle(graph, node, visited, rec_stack):
63     """
64     递归检测图中是否有环。
65
66     参数：
67     graph - 图的邻接表表示。
68     node - 当前检测的节点。
69     visited - 访问过的节点集合。
70     rec_stack - 递归栈。
71
72     返回：
73     布尔值，表示从当前节点开始是否有环。
74     """
75     if node in rec_stack:
76         return True
77     if node in visited:
78         return False
79
80     visited.add(node)
81     rec_stack.add(node)
82
83     for neighbor in graph[node]:
84         if detect_cycle(graph, neighbor, visited, rec_stack):
85             return True
86
87     rec_stack.remove(node)
88     return False
89
90 # 主函数
91 def main():
92     N = int(input())
93     edges = list(map(int, input().split()))

```

```
94  
95     result = find_start_and_end_nodes(edges)  
96     print(" ".join(map(str, result)))  
97  
98 if __name__ == "__main__":  
99     main()
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #define MAX_EDGES 1000 // 定义最大边数
6
7 // 图的结构体定义
8 typedef struct {
9     int node; // 节点编号
10    int neighbors[MAX_EDGES]; // 邻接节点数组
11    int neighbor_count; // 邻接节点的数量
12 } Graph;
13
14 // 集合的结构体定义
15 typedef struct {
16     int items[MAX_EDGES]; // 集合中的元素数组
17     int size; // 集合中元素的数量
18 } Set;
19
20 // 向集合中插入元素
21 void set_insert(Set *set, int item) {
22     for (int i = 0; i < set->size; i++) {
23         if (set->items[i] == item) {
24             return; // 如果元素已存在, 直接返回
25         }
26     }
27     set->items[set->size++] = item; // 插入元素到集合中
28 }
29
30 // 检查集合中是否包含某元素
31 int set_contains(Set *set, int item) {
32     for (int i = 0; i < set->size; i++) {
33         if (set->items[i] == item) {
34             return 1; // 如果找到元素, 返回1
35         }
36     }
37     return 0; // 未找到元素, 返回0
38 }
39
40 // 从集合中删除元素
41 void set_erase(Set *set, int item) {
42     for (int i = 0; i < set->size; i++) {
43         if (set->items[i] == item) {
44             set->items[i] = set->items[--set->size]; // 删除元素, 通过将最后一个元素移动到删除位置
```

```

45         return;
46     }
47 }
48 }

50 // 检测图中是否存在环
51 int detectCycle(Graph *graph, int node, Set *visited, Set *recStack) {
52     if (set_contains(recStack, node)) {
53         return 1; // 如果节点在递归栈中, 表示找到环
54     }
55     if (set_contains(visited, node)) {
56         return 0; // 如果节点已访问, 跳过
57     }
58

59     set_insert(visited, node); // 标记节点为已访问
60     set_insert(recStack, node); // 将节点加入递归栈
61

62     for (int i = 0; i < graph[node].neighbor_count; i++) {
63         if (detectCycle(graph, graph[node].neighbors[i], visited, recStack)) {
64             return 1; // 递归检测邻接节点, 如果找到环, 返回1
65         }
66     }
67

68     set_erase(recStack, node); // 从递归栈中移除节点
69     return 0; // 未找到环, 返回0
70 }

71

72 // 检测图中是否有环的主函数
73 int hasCycle(Graph *graph, Set *nodes) {
74     Set visited = {0}, recStack = {0};
75

76     for (int i = 0; i < nodes->size; i++) {
77         if (detectCycle(graph, nodes->items[i], &visited, &recStack)) {
78             return 1; // 如果检测到环, 返回1
79         }
80     }
81

82     return 0; // 未检测到环, 返回0
83 }
84

85 // 寻找起点和终点节点
86 int findStartAndEndNodes(Graph *graph, Set *nodes, int *edges, int edge_count,
87 int *startNode, Set *endNodes) {
88     int inDegree[MAX_EDGES] = {0}, outDegree[MAX_EDGES] = {0}; // 入度和出
89     度数组
90

91     // 构建图的邻接表, 计算每个节点的入度和出度

```

```

90         for (int i = 0; i < edge_count; i += 2) {
91             int start = edges[i];
92             int end = edges[i + 1];
93             set_insert(nodes, start);
94             set_insert(nodes, end);
95             inDegree[end]++;
96             outDegree[start]++;
97             graph[start].neighbors[graph[start].neighbor_count++] = end;
98         }
99
100        // 如果图中有环，返回-1
101        if (hasCycle(graph, nodes)) {
102            return -1;
103        }
104
105        // 寻找起点和终点
106        *startNode = -1;
107        for (int i = 0; i < nodes->size; i++) {
108            int node = nodes->items[i];
109            if (inDegree[node] == 0) {
110                *startNode = node; // 找到入度为0的节点作为起点
111            }
112            if (outDegree[node] == 0) {
113                set_insert(endNodes, node); // 找到出度为0的节点作为终点
114            }
115        }
116
117        return 0;
118    }
119
120    // 主函数
121    int main() {
122        char line[1024];
123        fgets(line, sizeof(line), stdin); // 读取N, 但在这个程序中不使用
124        fgets(line, sizeof(line), stdin); // 读取边的数据
125
126        int edges[MAX_EDGES], edge_count = 0;
127        char *token = strtok(line, " ");
128        while (token != NULL) {
129            edges[edge_count++] = atoi(token); // 解析边数据
130            token = strtok(NULL, " ");
131        }
132
133        Graph graph[MAX_EDGES] = {0};
134        Set nodes = {0}, endNodes = {0};
135        int startNode = -1;
136
137        // 寻找起点和终点

```

```
138     if (findStartAndEndNodes(graph, &nodes, edges, edge_count, &startNode, &endNodes) == -1) {
139         printf("-1\n"); // 如果有环, 输出-1
140     } else {
141         printf("%d ", startNode); // 输出起点
142         for (int i = 0; i < endNodes.size; i++) {
143             printf("%d ", endNodes.items[i]); // 输出终点
144         }
145         printf("\n");
146     }
147
148     return 0;
149 }
```

完整用例

用例1

```
1 3
2 0 1 1 2 2 3
```

用例2

```
1 3
2 0 1 1 2 2 0
```

用例3

```
1 4
2 0 1 0 2 0 3 1 4
```

用例4

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

用例5

```
▼ Plain Text |  
1   6  
2   0 1 0 2 1 3 2 3 3 4 4 2
```

用例6

```
▼ Plain Text |  
1   4  
2   0 1 1 0 1 2 2 3
```

用例7

```
▼ Plain Text |  
1   3  
2   0 1 0 2 1 3
```

用例8

```
▼ Plain Text |  
1   7  
2   1 2 2 3 2 4 2 5 3 6 4 6 4 7
```

用例9

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

用例10

```
▼ Plain Text |  
1   4  
2   0 1 1 0 2 3 3 2
```

文章目录

- 最新华为OD机试
- 题目描述
- 输入描述
- 输出描述
 - 备注
- 用例1
- 用例2
- 解题思路
 - 环的检测
 - 确定起点和终点
- C++
- Java
- javaScript
- Python
- C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10

机考真题 华为OD

No No No



CSDN @算法大师

来自: 华为OD机考2025B卷 – 查找一个有向网络的头节点和尾节点 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 删除重复数字后的最大数字（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

给定一个由纯数字组成以字符串表示的数值，现要求字符串中的每个数字最多只能出现2次，超过的需要进行删除；

删除某个重复的数字后，其它数字相对位置保持不变。

如”34533”，数字3重复超过2次，需要删除其中一个3，删除第一个3后获得最大数值”4533”

请返回经过删除操作后的最大的数值，以字符串表示。

输入描述

第一行为一个纯数字组成的字符串，长度范围：[1,100000]

输出描述

输出经过删除操作后的最大的数值

示例1

输入

▼		Plain Text
1 34533		

输出

▼		Plain Text
1 4533		

说明

示例2

输入

Plain Text

```
1 5445795045
```

输出

Plain Text

```
1 5479504
```

说明

解题思路

类似题，可以参考题解：

[316. 去除重复字母 – 力扣（LeetCode）](#)

题目解读

1. **删除多余的数字**: 字符串中每个数字最多只能出现两次，若某个数字出现超过两次，则需要删除多余的次数。例如，字符串 `34533` 中的数字 `3` 出现了三次，超过了两次限制，需要删除一次 `3`。
2. **保持相对位置**: 在删除多余数字后，数字的相对顺序必须保持不变。即在删除多余数字后，剩余的数字排列顺序与原字符串中相同。
3. **最大化数值**: 题目要求删除后得到的结果应该是最大可能的数值字符串。例如，从 `34533` 中删除第一个 `3` 后得到 `4533`，这是可能的最大数值。

示例解析

示例1

- 输入: `34533`
- 过程:
 - 数字 `3` 出现了三次，超过了限制。我们删除第一个 `3`。
 - 删除后的字符串为 `4533`，这是最大值。
- 输出: `4533`

示例2

- 输入: `5445795045`
- 过程:
 - 数字 `4` 和 `5` 出现了三次，需要删除多余的 `4` 和 `5`。

- 删除最靠前的 4 和 5，得到 5479504，这是可能的最大数。
- 输出：5479504

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         // 输入一个字符串
6         Scanner scanner = new Scanner(System.in);
7         String numStr = scanner.next();
8
9         // 使用数组记录每个数字的使用次数, 索引 0-9 对应字符 '0'-'9'
10        int[] usedNum = new int[10];
11        int[] totalNum = new int[10];
12
13        // 遍历字符串, 统计每个数字的总出现次数
14        for (int i = 0; i < numStr.length(); i++) {
15            totalNum[numStr.charAt(i) - '0']++;
16        }
17
18        // 存储最终的数字序列
19        LinkedList<Character> numList = new LinkedList<>();
20
21        // 遍历字符串
22        for (int i = 0; i < numStr.length(); i++) {
23            char c = numStr.charAt(i);
24            int num = c - '0';
25
26            // 如果当前数字已经被使用了2次, 那么就跳过
27            if (usedNum[num] == 2) {
28                totalNum[num]--;
29                continue;
30            }
31
32            // 如果当前数字比栈顶元素大, 并且栈顶元素可以重新加入, 则移除栈顶元素以获
33            // 取更大的数值
34            while (!numList.isEmpty()) {
35                char top = numList.getLast();
36                int topNum = top - '0';
37
38                if (top < c && totalNum[topNum] + usedNum[topNum] - 1 >= 2)
39                {
40                    numList.removeLast();
41                    usedNum[topNum]--;
42                } else {
43                    break;
44                }
45            }
46        }
47    }
48}
```

```
44
45      // 将当前数字添加到栈中
46      numList.addLast(c);
47      // 标记当前数字已经使用了一次
48      totalNum[num]--;
49      usedNum[num]++;
50  }
51
52  // 将栈中的数字转换为字符串
53  StringBuilder result = new StringBuilder();
54  for (char c : numList) {
55      result.append(c);
56  }
57  // 输出最终的数字序列
58  System.out.println(result.toString());
59
60 }
```

Python

```
1 # 导入deque用于存储最终的数字序列
2 from collections import deque
3
4 # 输入一个字符串
5 num_str = input().strip()
6
7 # 使用列表记录每个数字的使用次数，索引 0-9 对应字符 '0'-'9'
8 used_num = [0] * 10
9 total_num = [0] * 10
10
11 # 遍历字符串，统计每个数字的总出现次数
12 for c in num_str:
13     total_num[int(c)] += 1
14
15 # 存储最终的数字序列
16 num_list = deque()
17
18 # 遍历字符串
19 for c in num_str:
20     num = int(c)
21
22     # 如果当前数字已经被使用了2次，那么就跳过
23     if used_num[num] == 2:
24         total_num[num] -= 1 # 减少未使用的数量
25         continue
26
27     # 如果当前数字比队列尾部元素大，且尾部元素可以重新加入，则移除尾部元素
28     while num_list and num_list[-1] < c:
29         top = num_list[-1]
30         top_num = int(top)
31
32         # 判断栈顶元素是否可以重新加入
33         if total_num[top_num] + used_num[top_num] - 1 >= 2:
34             num_list.pop()
35             used_num[top_num] -= 1
36         else:
37             break
38
39     # 将当前数字添加到队列中
40     num_list.append(c)
41     # 标记当前数字已经使用了一次
42     total_num[num] -= 1
43     used_num[num] += 1
44
45 # 将队列中的数字转换为字符串并输出
```

```
46     result = ''.join(num_list)
47     print(result)
```

JavaScript

```
1 const readline = require('readline');
2 const rl = readline.createInterface({
3     input: process.stdin,
4     output: process.stdout
5 });
6
7 rl.on('line', (numStr) => {
8     // 使用数组记录每个数字的使用次数，索引 0-9 对应字符 '0'-'9'
9     let usedNum = Array(10).fill(0);
10    let totalNum = Array(10).fill(0);
11
12    // 遍历字符串，统计每个数字的总出现次数
13    for (let char of numStr) {
14        totalNum[parseInt(char)]++;
15    }
16
17    // 存储最终的数字序列
18    let numList = [];
19
20    // 遍历字符串
21    for (let char of numStr) {
22        let num = parseInt(char);
23
24        // 如果当前数字已经被使用了2次，那么就跳过
25        if (usedNum[num] === 2) {
26            totalNum[num]--;
27            continue;
28        }
29
30        // 如果当前数字比数组最后元素大，且最后元素可以重新加入，则移除最后元素
31        while (numList.length > 0 && numList[numList.length - 1] < char) {
32            let top = numList[numList.length - 1];
33            let topNum = parseInt(top);
34
35            // 判断数组最后元素是否可以重新加入
36            if (totalNum[topNum] + usedNum[topNum] - 1 >= 2) {
37                numList.pop();
38                usedNum[topNum]--;
39            } else {
40                break;
41            }
42        }
43
44        // 将当前数字添加到数组中
45        numList.push(char);
```

```
46     // 标记当前数字已经使用了一次
47     totalNum[num]--;
48     usedNum[num]++;
49 }
50
51     // 输出最终的数字序列
52     console.log(numList.join(''));
53     rl.close();
54 });

});
```

C++

```
1 #include <iostream>
2 #include <deque>
3 #include <string>
4 using namespace std;
5
6 int main() {
7     // 输入一个字符串
8     string numStr;
9     cin >> numStr;
10
11    // 使用数组记录每个数字的使用次数，索引 0-9 对应字符 '0'-'9'
12    int usedNum[10] = {0};
13    int totalNum[10] = {0};
14
15    // 遍历字符串，统计每个数字的总出现次数
16    for (char c : numStr) {
17        totalNum[c - '0']++;
18    }
19
20    // 存储最终的数字序列
21    deque<char> numList;
22
23    // 遍历字符串
24    for (char c : numStr) {
25        int num = c - '0';
26
27        // 如果当前数字已经被使用了2次，那么就跳过
28        if (usedNum[num] == 2) {
29            totalNum[num]--;
30            continue;
31        }
32
33        // 如果当前数字比队列尾部元素大，且尾部元素可以重新加入，则移除尾部元素
34        while (!numList.empty() && numList.back() < c) {
35            char top = numList.back();
36            int topNum = top - '0';
37
38            if (totalNum[topNum] + usedNum[topNum] - 1 >= 2) {
39                numList.pop_back();
40                usedNum[topNum]--;
41            } else {
42                break;
43            }
44        }
45    }
```

```
46     // 将当前数字添加到队列中
47     numList.push_back(c);
48     // 标记当前数字已经使用了一次
49     totalNum[num]--;
50     usedNum[num]++;
51 }
52
53 // 输出最终的数字序列
54 for (char c : numList) {
55     cout << c;
56 }
57 cout << endl;
58
59 return 0;
60 }
```

C语言

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main() {
5     // 输入一个字符串
6     char numStr[100001];
7     scanf("%s", numStr);
8
9     // 使用数组记录每个数字的使用次数，索引 0~9 对应字符 '0'~'9'
10    int usedNum[10] = {0};
11    int totalNum[10] = {0};
12    int len = strlen(numStr);
13
14    // 遍历字符串，统计每个数字的总出现次数
15    for (int i = 0; i < len; i++) {
16        totalNum[numStr[i] - '0']++;
17    }
18
19    // 存储最终的数字序列
20    char numList[100001];
21    int listSize = 0;
22
23    // 遍历字符串
24    for (int i = 0; i < len; i++) {
25        char c = numStr[i];
26        int num = c - '0';
27
28        // 如果当前数字已经被使用了2次，那么就跳过
29        if (usedNum[num] == 2) {
30            totalNum[num]--;
31            continue;
32        }
33
34        // 如果当前数字比栈顶元素大，且栈顶元素可以重新加入，则移除栈顶元素
35        while (listSize > 0 && numList[listSize - 1] < c) {
36            char top = numList[listSize - 1];
37            int topNum = top - '0';
38
39            if (totalNum[topNum] + usedNum[topNum] - 1 >= 2) {
40                listSize--;
41                usedNum[topNum]--;
42            } else {
43                break;
44            }
45        }
}
```

```
46  
47     // 将当前数字添加到数组中  
48     numList[listSize++] = c;  
49     // 标记当前数字已经使用了一次  
50     totalNum[num]--;  
51     usedNum[num]++;
52 }
53
54 // 输出最终的数字序列
55 numList[listSize] = '\0'; // 添加字符串结束符
56 printf("%s\n", numList);
57
58 return 0;
59 }
```



来自: 华为OD机考2025B卷 – 删除重复数字后的最大数字 (Java & Python& JS & C++ & C) –
CSDN博客

华为OD机考2025B卷 - 最大岛屿体积 (Java & Python& JS & C++ & C) -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

题目描述

给你一个由 大于0的数（陆地）和 0（水）组成的二维网格，请你计算网格中最大岛屿的体积。陆地的数表示所在岛屿的体积。

岛屿总是被水包围，并且每座岛屿只能由水平方向和/或竖直方向上相邻的陆地连接形成。

此外，你可以假设该网格的四条边均被水包围。

输入描述

第一行是二维网格的宽和高。

后面几行是二维网格。

输出描述

输出岛屿的最大体积。

示例1

输入

```
1 3 3
2 0 0 0
3 0 0 0
4 0 0 0
```

输出

```
1 0
```

说明

示例2

输入

```
Plain Text |  
1 5 3  
2 1 0 1 0 1  
3 1 0 1 0 1  
4 1 0 1 0 1
```

输出

```
Plain Text |  
1 3
```

说明

解题思路

1. 问题本质

这个问题本质上是在求解一个[连通分量](#)的最大值问题：

- 网格中每个值大于0的单元格可视为一个节点
- 两个相邻（上下左右四个方向）的节点构成连接关系
- 需要找出网格中体积最大的连通分量

2. 算法详解

2.1 遍历与标记策略

- 使用双层循环遍历整个网格的每个单元格
- 对每个未访问的陆地单元格(值>0)，启动一次DFS
- 使用布尔数组 `visited[][]` 标记已经访问过的单元格，避免重复计算

2.2 DFS实现细节

- 基本情况处理：
 - 当索引越界时返回0
 - 当单元格已被访问过时返回0
 - 当单元格是水域(值为0)时返回0
- 递归探索过程：
 - a. 标记当前单元格为已访问
 - b. 记录当前单元格的体积值
 - c. 递归探索四个方向的相邻单元格

d. 将当前单元格和所有相邻连通单元格的体积累加起来

2.3 最大值更新机制

- 每完成一次DFS，就得到一个完整岛屿的体积
- 使用变量 `maxVolume` 记录已发现的最大岛屿体积
- 每次DFS结束后，将新计算的岛屿体积与当前最大值比较并更新

```
1 import java.util.Scanner;
2
3
4 public class Main {
5     public static void main(String[] args) {
6         // 创建Scanner对象用于读取输入
7         Scanner scanner = new Scanner(System.in);
8
9         // 读取网格的宽度和高度
10        int width = scanner.nextInt(); // 网格宽度
11        int height = scanner.nextInt(); // 网格高度
12
13        // 创建二维数组表示网格，行数为高度，列数为宽度
14        int[][] grid = new int[height][width];
15
16        // 读取网格中每个单元格的值
17        // 大于0的数表示陆地及其体积，0表示水
18        for (int i = 0; i < height; i++) {
19            for (int j = 0; j < width; j++) {
20                grid[i][j] = scanner.nextInt();
21            }
22        }
23
24        // 调用方法计算网格中最大岛屿的体积
25        int maxVolume = getMaxIslandVolume(grid, height, width);
26
27        // 输出最大岛屿体积
28        System.out.println(maxVolume);
29
30        // 关闭Scanner，释放资源
31        scanner.close();
32    }
33
34
35     public static int getMaxIslandVolume(int[][] grid, int height, int width) {
36         // 初始化最大体积为0
37         int maxVolume = 0;
38         // 创建访问标记数组，用于DFS中标记已访问的单元格
39         boolean[][] visited = new boolean[height][width];
40
41         // 遍历整个网格
42         for (int i = 0; i < height; i++) {
43             for (int j = 0; j < width; j++) {
44                 // 如果当前单元格是陆地(值>0)且未被访问过
```

```

45             if (grid[i][j] > 0 && !visited[i][j]) {
46                 // 通过DFS计算以当前单元格为起点的岛屿体积
47                 int volume = dfs(grid, visited, i, j, height, width);
48                 // 更新最大岛屿体积
49                 maxVolume = Math.max(maxVolume, volume);
50             }
51         }
52     }
53
54     // 返回最大岛屿体积
55     return maxVolume;
56 }
57
58 public static int dfs(int[][][] grid, boolean[][][] visited, int i, int j,
59 int height, int width) {
60     // 边界检查: 如果索引越界、单元格已访问或是水(值为0), 则返回0
61     if (i < 0 || i >= height || j < 0 || j >= width || visited[i][j] ||
62     | grid[i][j] == 0) {
63         return 0;
64     }
65
66     // 标记当前单元格为已访问
67     visited[i][j] = true;
68
69     // 获取当前单元格的体积值
70     int currentVolume = grid[i][j];
71
72     // 初始化岛屿体积为当前单元格的体积
73     int volume = currentVolume;
74     // 递归探索四个方向(上、下、左、右)相邻的单元格, 累加体积
75     volume += dfs(grid, visited, i+1, j, height, width); // 向下探索
76     volume += dfs(grid, visited, i-1, j, height, width); // 向上探索
77     volume += dfs(grid, visited, i, j+1, height, width); // 向右探索
78     volume += dfs(grid, visited, i, j-1, height, width); // 向左探索
79
80     // 返回计算得到的岛屿总体积
81     return volume;
82 }

```

Python

```
1 # 深度优先搜索函数，用于计算单个岛屿的体积
2 def dfs(grid, visited, i, j, height, width):
3     # 边界检查：如果索引越界、单元格已访问或是水(值为0)，则返回0
4     if i < 0 or i >= height or j < 0 or j >= width or visited[i][j] or grid[i][j] == 0:
5         return 0
6
7     # 标记当前单元格为已访问
8     visited[i][j] = True
9
10    # 获取当前单元格的体积值
11    current_volume = grid[i][j]
12
13    # 初始化岛屿体积为当前单元格的体积
14    volume = current_volume
15    # 递归探索四个方向(上、下、左、右)相邻的单元格，累加体积
16    volume += dfs(grid, visited, i+1, j, height, width) # 向下探索
17    volume += dfs(grid, visited, i-1, j, height, width) # 向上探索
18    volume += dfs(grid, visited, i, j+1, height, width) # 向右探索
19    volume += dfs(grid, visited, i, j-1, height, width) # 向左探索
20
21    # 返回计算得到的岛屿总体积
22    return volume
23
24 # 计算网格中最大岛屿的体积
25 def get_max_island_volume(grid, height, width):
26     # 初始化最大体积为0
27     max_volume = 0
28     # 创建访问标记数组，用于DFS中标记已访问的单元格
29     visited = [[False for _ in range(width)] for _ in range(height)]
30
31     # 遍历整个网格
32     for i in range(height):
33         for j in range(width):
34             # 如果当前单元格是陆地(值>0)且未被访问过
35             if grid[i][j] > 0 and not visited[i][j]:
36                 # 通过DFS计算以当前单元格为起点的岛屿体积
37                 volume = dfs(grid, visited, i, j, height, width)
38                 # 更新最大岛屿体积
39                 max_volume = max(max_volume, volume)
40
41     # 返回最大岛屿体积
42     return max_volume
43
44 def main():
```

```
45     # 读取网格的宽度和高度
46     width, height = map(int, input().split())
47
48     # 创建二维列表表示网格
49     grid = []
50
51     # 读取网格中每个单元格的值
52     # 大于0的数表示陆地及其体积, 0表示水
53     for _ in range(height):
54         row = list(map(int, input().split()))
55         grid.append(row)
56
57     # 调用函数计算网格中最大岛屿的体积
58     max_volume = get_max_island_volume(grid, height, width)
59
60     # 输出最大岛屿体积
61     print(max_volume)
62
63 if __name__ == "__main__":
64     main()
```

JavaScript

```
1 // 引入readline模块用于读取用户输入
2 const readline = require('readline');
3
4 // 创建readline接口
5 const rl = readline.createInterface({
6   input: process.stdin,
7   output: process.stdout
8 });
9
10 // 保存输入行
11 const lines = [];
12
13 // 监听输入事件
14 rl.on('line', (line) => {
15   lines.push(line);
16 });
17
18 // 监听关闭事件，处理输入数据
19 rl.on('close', () => {
20   // 解析第一行获取网格的宽度和高度
21   const [width, height] = lines[0].split(' ').map(Number);
22
23   // 创建二维数组表示网格
24   const grid = [];
25
26   // 读取网格中每个单元格的值
27   // 大于0的数表示陆地及其体积，0表示水
28   for (let i = 0; i < height; i++) {
29     grid.push(lines[i + 1].split(' ').map(Number));
30   }
31
32   // 调用函数计算网格中最大岛屿的体积
33   const maxVolume = getMaxIslandVolume(grid, height, width);
34
35   // 输出最大岛屿体积
36   console.log(maxVolume);
37 });
38
39 /**
40  * 计算网格中最大岛屿的体积
41  * @param {number[][]} grid - 表示网格的二维数组
42  * @param {number} height - 网格的高度
43  * @param {number} width - 网格的宽度
44  * @returns {number} - 最大岛屿体积
45 */
```

```

46  function getMaxIslandVolume(grid, height, width) {
47      // 初始化最大体积为0
48      let maxVolume = 0;
49      // 创建访问标记数组，用于DFS中标记已访问的单元格
50      const visited = Array(height).fill().map(() => Array(width).fill(false));
51
52      // 遍历整个网格
53      for (let i = 0; i < height; i++) {
54          for (let j = 0; j < width; j++) {
55              // 如果当前单元格是陆地(值>0)且未被访问过
56              if (grid[i][j] > 0 && !visited[i][j]) {
57                  // 通过DFS计算以当前单元格为起点的岛屿体积
58                  const volume = dfs(grid, visited, i, j, height, width);
59                  // 更新最大岛屿体积
60                  maxVolume = Math.max(maxVolume, volume);
61              }
62          }
63      }
64
65      // 返回最大岛屿体积
66      return maxVolume;
67  }
68
69 /**
70 * 深度优先搜索函数，用于计算单个岛屿的体积
71 * @param {number[][]} grid - 表示网格的二维数组
72 * @param {boolean[][]} visited - 表示单元格是否已访问的二维数组
73 * @param {number} i - 当前单元格的行索引
74 * @param {number} j - 当前单元格的列索引
75 * @param {number} height - 网格的高度
76 * @param {number} width - 网格的宽度
77 * @returns {number} - 当前岛屿的体积
78 */
79 function dfs(grid, visited, i, j, height, width) {
80     // 边界检查：如果索引越界、单元格已访问或是水(值为0)，则返回0
81     if (i < 0 || i >= height || j < 0 || j >= width || visited[i][j] || grid[i][j] === 0) {
82         return 0;
83     }
84
85     // 标记当前单元格为已访问
86     visited[i][j] = true;
87
88     // 获取当前单元格的体积值
89     const currentVolume = grid[i][j];
90
91     // 初始化岛屿体积为当前单元格的体积

```

```
92     let volume = currentVolume;
93     // 递归探索四个方向(上、下、左、右)相邻的单元格，累加体积
94     volume += dfs(grid, visited, i+1, j, height, width); // 向下探索
95     volume += dfs(grid, visited, i-1, j, height, width); // 向上探索
96     volume += dfs(grid, visited, i, j+1, height, width); // 向右探索
97     volume += dfs(grid, visited, i, j-1, height, width); // 向左探索
98
99     // 返回计算得到的岛屿总体积
100    return volume;
101 }
```

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5 // 深度优先搜索函数，用于计算单个岛屿的体积
6 int dfs(vector<vector<int>>& grid, vector<vector<bool>>& visited,
7         int i, int j, int height, int width) {
8     // 边界检查：如果索引越界、单元格已访问或是水(值为0)，则返回0
9     if (i < 0 || i >= height || j < 0 || j >= width || visited[i][j] || gr
id[i][j] == 0) {
10         return 0;
11     }
12
13     // 标记当前单元格为已访问
14     visited[i][j] = true;
15
16     // 获取当前单元格的体积值
17     int currentVolume = grid[i][j];
18
19     // 初始化岛屿体积为当前单元格的体积
20     int volume = currentVolume;
21     // 递归探索四个方向(上、下、左、右)相邻的单元格，累加体积
22     volume += dfs(grid, visited, i+1, j, height, width); // 向下探索
23     volume += dfs(grid, visited, i-1, j, height, width); // 向上探索
24     volume += dfs(grid, visited, i, j+1, height, width); // 向右探索
25     volume += dfs(grid, visited, i, j-1, height, width); // 向左探索
26
27     // 返回计算得到的岛屿总体积
28     return volume;
29 }
30
31 // 计算网格中最大岛屿的体积
32 int getMaxIslandVolume(vector<vector<int>>& grid, int height, int width) {
33     // 初始化最大体积为0
34     int maxVolume = 0;
35     // 创建访问标记数组，用于DFS中标记已访问的单元格
36     vector<vector<bool>> visited(height, vector<bool>(width, false));
37
38     // 遍历整个网格
39     for (int i = 0; i < height; i++) {
40         for (int j = 0; j < width; j++) {
41             // 如果当前单元格是陆地(值>0)且未被访问过
42             if (grid[i][j] > 0 && !visited[i][j]) {
43                 // 通过DFS计算以当前单元格为起点的岛屿体积
44                 int volume = dfs(grid, visited, i, j, height, width);
45             }
46         }
47     }
48
49     return maxVolume;
50 }
```

```
45 // 更新最大岛屿体积
46     maxVolume = max(maxVolume, volume);
47 }
48 }
49 }
50 }
51 // 返回最大岛屿体积
52 return maxVolume;
53 }
54 }
55 int main() {
56     // 读取网格的宽度和高度
57     int width, height;
58     cin >> width >> height;
59 }
60 // 创建二维向量表示网格，行数为高度，列数为宽度
61 vector<vector<int>> grid(height, vector<int>(width));
62 }
63 // 读取网格中每个单元格的值
64 // 大于0的数表示陆地及其体积，0表示水
65 for (int i = 0; i < height; i++) {
66     for (int j = 0; j < width; j++) {
67         cin >> grid[i][j];
68     }
69 }
70 }
71 // 调用方法计算网格中最大岛屿的体积
72 int maxVolume = getMaxIslandVolume(grid, height, width);
73 }
74 // 输出最大岛屿体积
75 cout << maxVolume << endl;
76 }
77 }
78 }
```

C语言

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <stdbool.h>
4
5 // 深度优先搜索函数，用于计算单个岛屿的体积
6 int dfs(int** grid, bool** visited, int i, int j, int height, int width) {
7     // 边界检查：如果索引越界、单元格已访问或是水（值为0），则返回0
8     if (i < 0 || i >= height || j < 0 || j >= width || visited[i][j] || gr
9         id[i][j] == 0) {
10         return 0;
11     }
12
13     // 标记当前单元格为已访问
14     visited[i][j] = true;
15
16     // 获取当前单元格的体积值
17     int currentVolume = grid[i][j];
18
19     // 初始化岛屿体积为当前单元格的体积
20     int volume = currentVolume;
21     // 递归探索四个方向（上、下、左、右）相邻的单元格，累加体积
22     volume += dfs(grid, visited, i+1, j, height, width); // 向下探索
23     volume += dfs(grid, visited, i-1, j, height, width); // 向上探索
24     volume += dfs(grid, visited, i, j+1, height, width); // 向右探索
25     volume += dfs(grid, visited, i, j-1, height, width); // 向左探索
26
27     // 返回计算得到的岛屿总体积
28     return volume;
29 }
30
31 // 计算网格中最大岛屿的体积
32 int getMaxIslandVolume(int** grid, int height, int width) {
33     // 初始化最大体积为0
34     int maxVolume = 0;
35
36     // 创建访问标记数组，用于DFS中标记已访问的单元格
37     bool** visited = (bool**)malloc(height * sizeof(bool*));
38     for (int i = 0; i < height; i++) {
39         visited[i] = (bool*)calloc(width, sizeof(bool)); // 初始化所有值为fa
40     }
41
42     // 遍历整个网格
43     for (int i = 0; i < height; i++) {
44         for (int j = 0; j < width; j++) {
```

```

44         // 如果当前单元格是陆地(值>0)且未被访问过
45         if (grid[i][j] > 0 && !visited[i][j]) {
46             // 通过DFS计算以当前单元格为起点的岛屿体积
47             int volume = dfs(grid, visited, i, j, height, width);
48             // 更新最大岛屿体积
49             if (volume > maxVolume) {
50                 maxVolume = volume;
51             }
52         }
53     }
54 }
55
56 // 释放访问标记数组的内存
57 for (int i = 0; i < height; i++) {
58     free(visited[i]);
59 }
60 free(visited);
61
62 // 返回最大岛屿体积
63 return maxVolume;
64 }
65
66 int main() {
67     // 读取网格的宽度和高度
68     int width, height;
69     scanf("%d %d", &width, &height);
70
71     // 动态分配二维数组用于存储网格
72     int** grid = (int**)malloc(height * sizeof(int*));
73     for (int i = 0; i < height; i++) {
74         grid[i] = (int*)malloc(width * sizeof(int));
75     }
76
77     // 读取网格中每个单元格的值
78     // 大于0的数表示陆地及其体积, 0表示水
79     for (int i = 0; i < height; i++) {
80         for (int j = 0; j < width; j++) {
81             scanf("%d", &grid[i][j]);
82         }
83     }
84
85     // 调用函数计算网格中最大岛屿的体积
86     int maxVolume = getMaxIslandVolume(grid, height, width);
87
88     // 输出最大岛屿体积
89     printf("%d\n", maxVolume);
90
91     // 释放网格内存

```

```
92     for (int i = 0; i < height; i++) {  
93         free(grid[i]);  
94     }  
95     free(grid);  
96  
97     return 0;  
98 }
```

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
- [输出描述](#)
- [示例1](#)
- [示例2](#)
- [解题思路](#)
 - [1. 问题本质](#)
 - [2. 算法详解](#)
 - [2.1 遍历与标记策略](#)
 - [2.2 DFS实现细节](#)
 - [2.3 最大值更新机制](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)

机考真题 华为OD

No No No



CSDN @算法大师

| 来自: 华为OD机考2025B卷 – 最大岛屿体积 (Java & Python& JS & C++ & C) –CSDN博客