

t818

[华为OD机考 2025C卷 - 识文断句 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 简易压缩算法 / 一种字符串压缩表示的解压 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 最大矩阵和 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - IPv4地址转换成整数 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025C卷 - 国际移动用户识别码\(IMSI\)匹配 \(C++ & Python & JAVA & JS & GO\)_华为机试 国际移动用户识别码\(imsi\)匹配-CSDN博客](#)

[华为OD机考2025C卷 - 字母组合 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 猴子爬山 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 数组去重和排序 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 密码输入检测 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 统计射击比赛成绩 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 字符统计及重排 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 矩阵扩散 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - CPU算力分配 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 滑动窗口的最大值 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 最大括号深度 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025C卷 - 流量波峰 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 模拟目录管理 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机考 2025C卷 - 识文断句 (C++ & Python & JAVA & JS & GO)-CSDN博客

识文断句

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试 2025C卷 200分题型

题目描述

考友反馈题目大意如下：

先给定一些短词字符串作为分割词，去分割一段长字符串；从前往后遍历分割词，查找分割长字符串为对应的token；分割词中长度长的优先，长度相等的字典序大的优先；没能找到分割的保持原样，最后按照长字符串的顺序输出，一个括号一个token的形式。

输入描述

先输入短词字符串，每个短词字符串占一行。

空行分割

输入待分词长字符串

输出描述

按照一个括号一个token的形式输出分词后的字符串。

示例1

输入

```
Plain Text  
1 zhong guo  
2 zhong  
3 guo  
4 wo  
5 mei guo  
6  
7 wo ai zhong guo mei guo ye xing
```

输出



Plain Text |

1 (wo)(ai)(zhong guo)(mei guo)(ye)(xing)

题解

思路：本题解决需要用到 前缀树/字典树 的数据结构来处理词典。

1. 接收分割的短语字符串，使用 短语字符串 预构建 字典树，用于处理字符串分词。字典树 也是数据结构中一个比较常见的数据结构，推荐自己去学习学习。
2. 利用预构建 字典树 模拟分词，将待分词的字符串先按照空格进行切割获得一个字符串数组，从前往后循环进行分词(贪心使用长的词进行分割)。具体分词逻辑可以参照如下代码。
3. 注意本题要求尽可能用长的词典进行分割，当使用多单词短语字符串进行分割时，单词和单词之间空格匹配也需要判断是否存在。
4. 分割之后，使用括号包裹每个token，按顺序输出即可。

C++

```
1 #include <cstdio>
2 #include <iostream>
3 #include <vector>
4 #include <string>
5 #include <sstream>
6 #include <algorithm>
7 using namespace std;
8
9 // 通用 split 函数
10 vector<string> split(const string& str, const string& delimiter) {
11     vector<string> result;
12     size_t start = 0;
13     size_t end = str.find(delimiter);
14     while (end != string::npos) {
15         result.push_back(str.substr(start, end - start));
16         start = end + delimiter.length();
17         end = str.find(delimiter, start);
18     }
19     // 添加最后一个部分
20     result.push_back(str.substr(start));
21     return result;
22 }
23
24
25
26
27 // Trie 结构定义
28 struct TrieNode {
29     bool isWord;
30     TrieNode* children[27];
31
32     TrieNode() : isWord(false) {
33         fill(begin(children), end(children), nullptr);
34     }
35 };
36
37 // 根节点
38 TrieNode* root = new TrieNode();
39
40 // 插入单词
41 void insertWord(const string& word) {
42     TrieNode* currentNode = root;
43     for (char c : word) {
44         int index;
45         if (c != ' ') {
```

```
46         index = c - 'a';
47     } else {
48         index = 26;
49     }
50     if (!currentNode->children[index]) {
51         currentNode->children[index] = new TrieNode();
52     }
53     currentNode = currentNode->children[index];
54 }
55 currentNode->isWord = true;
56 }
57
58 int main() {
59     string word;
60     // 构建字典树
61     while(getline(cin, word)) {
62         // 空行
63         if (word.empty()) {
64             break;
65         }
66         insertWord(word);
67     }
68     // 记录每次分割的起始和终止位置
69     vector<pair<int, int>> result;
70     string sentence;
71     getline(cin, sentence);
72
73     int startIndex = 0;
74     vector<string> inputWords = split(sentence, " ");
75     int n = inputWords.size();
76     int index = 0;
77
78     while (index < n){
79         TrieNode* currentNode = root;
80         int tmpIndex = index;
81         // 记录合法匹配的位置
82         int lastValidIndex = -1;
83         // 利用字典树进行匹配
84         while (true) {
85             if (tmpIndex >= n) {
86                 break;
87             }
88             string tmp = inputWords[tmpIndex];
89             bool flag = true;
90             for (int j = 0; j < tmp.size(); j++) {
91                 char c = tmp[j];
92                 int position = c - 'a';
93                 if (!currentNode->children[position]) {
```

```
94         flag = false;
95         break;
96     }
97     currentNode = currentNode -> children[position];
98 }
99 if (!flag) {
100     break;
101 }
102 if (currentNode->isWord) {
103     lastValidIndex = tmpIndex;
104 }
105 // 分割空格处理
106 currentNode = currentNode-> children[26];
107 if (!currentNode) {
108     break;
109 }
110 tmpIndex++;
111 }
112
113 if (lastValidIndex != -1) {
114     result.push_back({index, lastValidIndex});
115     index = lastValidIndex + 1;
116 } else {
117     result.push_back({index, index});
118     index = index + 1;
119 }
120 }
121
122 // 输出结果
123 for (auto p : result) {
124     int start = p.first;
125     int end = p.second;
126     cout << "(";
127     for (int i = start; i <= end; i++) {
128         cout << inputWords[i];
129         if (i != end) {
130             cout << " ";
131         }
132     }
133     cout << ")";
134 }
135 }
136 return 0;
137 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     // 通用 split 函数
5     public static List<String> split(String str, String delimiter) {
6         return Arrays.asList(str.split(delimiter));
7     }
8
9     // Trie 结构定义
10    static class TrieNode {
11        boolean isWord;
12        TrieNode[] children = new TrieNode[27]; // 26字母 + 空格
13
14        TrieNode() {
15            isWord = false;
16        }
17    }
18
19    static TrieNode root = new TrieNode();
20
21    // 插入词典词
22    static void insertWord(String word) {
23        TrieNode node = root;
24        for (char c : word.toCharArray()) {
25            int index = (c == ' ') ? 26 : (c - 'a');
26            if (node.children[index] == null)
27                node.children[index] = new TrieNode();
28            node = node.children[index];
29        }
30        node.isWord = true;
31    }
32
33    public static void main(String[] args) {
34        Scanner sc = new Scanner(System.in);
35
36        while (sc.hasNextLine()) {
37            String word = sc.nextLine();
38            if (word.isEmpty()) break;
39            insertWord(word);
40        }
41
42        String sentence = sc.nextLine();
43        // 按空格进行分割
44        List<String> inputWords = split(sentence, " ");
45        int n = inputWords.size();
```

```
46     int index = 0;
47     // 每个token的起始位置和终止位置记录
48     List<int[]> result = new ArrayList<>();
49     while (index < n) {
50         TrieNode currentNode = root;
51         int tmpIndex = index;
52         // 记录合法匹配的位置
53         int lastValidIndex = -1;
54
55         while (true) {
56             if (tmpIndex >= n) break;
57             String word = inputWords.get(tmpIndex);
58             boolean flag = true;
59             for (char c : word.toCharArray()) {
60                 int pos = c - 'a';
61                 // 不存在
62                 if (currentNode.children[pos] == null) {
63                     flag = false;
64                     break;
65                 }
66                 currentNode = currentNode.children[pos];
67             }
68             if (!flag) break;
69             // 代表词典存在
70             if (currentNode.isWord) lastValidIndex = tmpIndex;
71
72             currentNode = currentNode.children[26]; // 空格分隔
73             if (currentNode == null) break;
74
75             tmpIndex++;
76         }
77
78         if (lastValidIndex != -1) {
79             result.add(new int[]{index, lastValidIndex});
80             index = lastValidIndex + 1;
81         } else {
82             result.add(new int[]{index, index});
83             index++;
84         }
85     }
86
87     for (int[] p : result) {
88         System.out.print("(");
89         for (int i = p[0]; i <= p[1]; i++) {
90             System.out.print(inputWords.get(i));
91             if (i != p[1]) System.out.print(" ");
92         }
93         System.out.print(")");
94     }
95 }
```

```
94         }
95     }
96 }
```

Python

```
1 # 通用 split 函数
2 def split(s, delimiter):
3     return s.strip().split(delimiter)
4
5 # Trie 节点定义
6 class TrieNode:
7     def __init__(self):
8         self.is_word = False
9         self.children = [None] * 27 # 26个字母 + 空格
10
11 # 插入单词
12 def insert_word(word, root):
13     node = root
14     for c in word:
15         index = 26 if c == ' ' else ord(c) - ord('a')
16         if not node.children[index]:
17             node.children[index] = TrieNode()
18         node = node.children[index]
19         node.is_word = True
20
21 import sys
22
23 root = TrieNode()
24 lines = sys.stdin.read().split('\n')
25 i = 0
26 while i < len(lines) and lines[i].strip() != "":
27     insert_word(lines[i], root)
28     i += 1
29
30 i += 1 # 跳过空行
31 sentence = lines[i] if i < len(lines) else ""
32 input_words = split(sentence, " ")
33 n = len(input_words)
34 index = 0
35 # 每个token的起始位置和终止位置记录
36 result = []
37
38 while index < n:
39     current_node = root
40     tmp_index = index
41     # 合法切割位置
42     last_valid_index = -1
43
44     while True:
45         if tmp_index >= n:
```

```
46         break
47     word = input_words[tmp_index]
48     flag = True
49     for c in word:
50         pos = ord(c) - ord('a')
51         # 不存在
52         if not current_node.children[pos]:
53             flag = False
54             break
55         current_node = current_node.children[pos]
56     if not flag:
57         break
58     # 表示短语存在
59     if current_node.is_word:
60         last_valid_index = tmp_index
61     # 空格判断
62     current_node = current_node.children[26]
63     if not current_node:
64         break
65     tmp_index += 1
66
67     if last_valid_index != -1:
68         result.append((index, last_valid_index))
69         index = last_valid_index + 1
70     else:
71         result.append((index, index))
72         index += 1
73
74     for start, end in result:
75         print("(", end="")
76         print(" ".join(input_words[start:end+1]), end="")
77         print(")", end="")
```

JavaScript

```
1 const readline = require("readline");
2
3 class TrieNode {
4     constructor() {
5         this.isWord = false;
6         this.children = Array(27).fill(null);
7     }
8 }
9
10 function insertWord(root, word) {
11     let node = root;
12     for (let ch of word) {
13         let index = ch === ' ' ? 26 : ch.charCodeAt(0) - 'a'.charCodeAt(0);
14         if (!node.children[index]) {
15             node.children[index] = new TrieNode();
16         }
17         node = node.children[index];
18     }
19     node.isWord = true;
20 }
21
22 function main(lines) {
23     let root = new TrieNode();
24     let i = 0;
25     while (lines[i] !== "") {
26         insertWord(root, lines[i]);
27         i++;
28     }
29
30     let sentence = lines[i + 1];
31     let inputWords = sentence.trim().split(" ");
32     let index = 0;
33     // 记录每次token的起始和终止位置
34     let result = [];
35
36     while (index < inputWords.length) {
37         let node = root;
38         let tmpIndex = index;
39         // 记录合法匹配的位置
40         let lastValidIndex = -1;
41
42         while (true) {
43             if (tmpIndex >= inputWords.length) break;
44             let word = inputWords[tmpIndex];
```

```

45     let flag = true;
46     for (let ch of word) {
47         let pos = ch.charCodeAt(0) - 'a'.charCodeAt(0);
48         // 不存在
49         if (!node.children[pos]) {
50             flag = false;
51             break;
52         }
53         node = node.children[pos];
54     }
55     if (!flag) break;
56     // 更新合法位置
57     if (node.isWord) lastValidIndex = tmpIndex;
58     // 空格判断
59     node = node.children[26];
60     if (!node) break;
61     tmpIndex++;
62 }
63
64 if (lastValidIndex !== -1) {
65     result.push([index, lastValidIndex]);
66     index = lastValidIndex + 1;
67 } else {
68     result.push([index, index]);
69     index++;
70 }
71 }
72
73 for (let [start, end] of result) {
74     process.stdout.write("(" + inputWords.slice(start, end + 1).join
75 (" ") + ")");
76 }
77
78 const rl = readline.createInterface({ input: process.stdin });
79 const lines = [];
80 rl.on("line", line => lines.push(line));
81 rl.on("close", () => main(lines));

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // Trie 节点定义
11 type TrieNode struct {
12     isWord    bool
13     // 26字母 + 空格
14     children [27]*TrieNode
15 }
16
17 // 插入词
18 func insertWord(root *TrieNode, word string) {
19     node := root
20     for _, c := range word {
21         index := 26
22         if c != ' ' {
23             index = int(c - 'a')
24         }
25         if node.children[index] == nil {
26             node.children[index] = &TrieNode{}
27         }
28         node = node.children[index]
29     }
30     node.isWord = true
31 }
32
33 func main() {
34     scanner := bufio.NewScanner(os.Stdin)
35     root := &TrieNode{}
36
37     // 读取词典
38     for scanner.Scan() {
39         line := scanner.Text()
40         if line == "" {
41             break
42         }
43         insertWord(root, line)
44     }
45 }
```

```
46     // 读取长字符串
47     scanner.Scan()
48     sentence := scanner.Text()
49     inputWords := strings.Split(sentence, " ")
50     n := len(inputWords)
51     index := 0
52     // 记录每次token的起始和终止位置
53     var result [][]int
54
55     for index < n {
56         node := root
57         tmpIndex := index
58         // 记录合法匹配的位置
59         lastValidIndex := -1
60
61         for {
62             if tmpIndex >= n {
63                 break
64             }
65             word := inputWords[tmpIndex]
66             flag := true
67             for _, ch := range word {
68                 pos := int(ch - 'a')
69                 // 不存在
70                 if node.children[pos] == nil {
71                     flag = false
72                     break
73                 }
74                 node = node.children[pos]
75             }
76             if !flag {
77                 break
78             }
79             // 更新合法位置
80             if node.isWord {
81                 lastValidIndex = tmpIndex
82             }
83             // 空格判断
84             node = node.children[26]
85             if node == nil {
86                 break
87             }
88             tmpIndex++
89         }
90
91         if lastValidIndex != -1 {
92             result = append(result, [2]int{index, lastValidIndex})
93             index = lastValidIndex + 1
```

```
94     } else {
95         result = append(result, [2]int{index, index})
96         index++
97     }
98 }
99
100 // 输出
101 for _, p := range result {
102     start, end := p[0], p[1]
103     fmt.Print("(")
104     fmt.Print(strings.Join(inputWords[start:end+1], " "))
105     fmt.Print(")")
106 }
107 }
```

| 来自: 华为OD机考 2025C卷 – 识文断句 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 简易压缩算法 / 一种字符串压缩表示的解压 (C++ & Python & JAVA & JS & GO)-CSDN博客

简易压缩算法 / 一种字符串压缩表示的解压

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

| 华为OD机试2025C卷 100分题型

题目描述

有一种简易[压缩算法]: 针对全部为小写[英文字母](#)组成的字符串, 将其中连续超过两个相同字母的部分压缩为连续个数加该字母 其他部分保持原样不变.

例如字符串aaabbcccd 经过压缩变成字符串 3abb4cd

请您编写解压函数,根据输入的字符串,判断其是否为合法压缩过的字符串

- 若输入合法则输出解压缩后的字符串
- 否则输出字符串!error来报告错误

输入描述

输入一行, 为一个 ASCII 字符串

长度不超过100字符

用例保证输出的字符串长度也不会超过100字符串

输出描述

若判断输入为合法的经过压缩后的字符串

则输出压缩前的字符串

若输入不合法 则输出字符串!error

示例1

输入

▼	Plain Text
1 4dff	

输出

Plain Text |

```
1 ddddff
```

说明

4d 扩展为 4 个 d , 故解压后的字符串为 ddddff

示例2

输入

Plain Text |

```
1 4d@A
```

输出

Plain Text |

```
1 !error
```

说明

全部由小写[英文字母]做成的字符串, 压缩后不会出现特殊字符@和大写字母 A
故输入不合法

题解

思路： 模拟

1. 根据题目进行执行解压操作即可，主要是细心。
2. 解压过程注意这两种非法情况，出现直接输出 !error
 - 出现非小写字母非数字时
 - 出现的数字小于2

C++

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     string s;
6     cin >> s;
7
8     string num = ""; // 存储数字部分
9     string res = ""; // 存储解压后的结果
10
11    for (char c : s) {
12        // 非小写字母或数字, 直接报错
13        if (!isdigit(c) && (c < 'a' || c > 'z')) {
14            cout << "!error" << endl;
15            return 0;
16        }
17
18        if (isdigit(c)) {
19            num += c; // 记录数字
20        } else {
21            // 解析之前累积的数字
22            if (!num.empty()) {
23                int repeat = stoi(num);
24                if (repeat <= 2) { // 解压缩要求大于2, 否则非法
25                    cout << "!error" << endl;
26                    return 0;
27                }
28                res.append(repeat, c); // 追加字符
29                num.clear(); // 重置数字
30            } else {
31                res += c; // 直接追加字符
32            }
33        }
34    }
35
36    cout << res << endl;
37    return 0;
38 }
```

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 s = scanner.next();
7         scanner.close(); // 关闭输入流
8
9         StringBuilder res = new StringBuilder();
10        StringBuilder num = new StringBuilder();
11
12        for (char c : s.toCharArray()) {
13            // 非小写字母或数字，直接报错
14            if (!Character.isDigit(c) && (c < 'a' || c > 'z')) {
15                System.out.println("!error");
16                return;
17            }
18
19            if (Character.isDigit(c)) {
20                num.append(c); // 记录数字
21            } else {
22                // 解析之前累积的数字
23                if (num.length() > 0) {
24                    int repeat = Integer.parseInt(num.toString());
25                    if (repeat <= 2) { // 解压缩要求大于2，否则非法
26                        System.out.println("!error");
27                        return;
28                    }
29                    res.append(String.valueOf(c).repeat(repeat)); // 追加字符
30                    num.setLength(0); // 清空 num
31                } else {
32                    res.append(c); // 直接追加字符
33                }
34            }
35        }
36
37        System.out.println(res.toString());
38    }
39 }
```

Python

```
1 import sys
2
3 def decompress_string(s):
4     res = []
5     num = ""
6
7     for c in s:
8         # 非小写字母或数字，直接报错
9         if not c.isdigit() and not ('a' <= c <= 'z'):
10            print("!error")
11            return
12
13     if c.isdigit():
14         num += c # 记录数字
15     else:
16         # 解析之前累积的数字
17         if num:
18             repeat = int(num)
19             if repeat <= 2: # 解压缩要求大于2，否则非法
20                 print("!error")
21             return
22             res.append(c * repeat) # 追加字符
23             num = "" # 重置数字
24         else:
25             res.append(c) # 直接追加字符
26
27     print("".join(res))
28
29 if __name__ == "__main__":
30     s = sys.stdin.readline().strip()
31     decompress_string(s)
```

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", (s) => {
9     let res = "";
10    let num = "";
11
12    for (let c of s) {
13        // 非小写字母或数字，直接报错
14        if (!/[0-9a-z]/.test(c)) {
15            console.log("!error");
16            rl.close();
17            return;
18        }
19
20        if (/^\d/.test(c)) {
21            num += c; // 记录数字
22        } else {
23            // 解析之前累积的数字
24            if (num !== "") {
25                let repeat = parseInt(num, 10);
26                if (repeat <= 2) { // 解压缩要求大于2，否则非法
27                    console.log("!error");
28                    rl.close();
29                    return;
30                }
31                res += c.repeat(repeat); // 追加字符
32                num = ""; // 重置数字
33            } else {
34                res += c; // 直接追加字符
35            }
36        }
37    }
38
39    console.log(res);
40    rl.close();
41});
```

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "unicode"
9 )
10
11 func main() {
12     scanner := bufio.NewScanner(os.Stdin)
13     if scanner.Scan() {
14         s := scanner.Text()
15
16         var res string
17         var num string
18
19         for _, c := range s {
20             // 非小写字母或数字，直接报错
21             if !unicode.IsDigit(c) && (c < 'a' || c > 'z') {
22                 fmt.Println("!error")
23                 return
24             }
25
26             if unicode.IsDigit(c) {
27                 num += string(c) // 记录数字
28             } else {
29                 // 解析之前累积的数字
30                 if num != "" {
31                     repeat, _ := strconv.Atoi(num)
32                     if repeat <= 2 { // 解压缩要求大于2, 否则非法
33                         fmt.Println("!error")
34                         return
35                     }
36                     res += repeatChar(c, repeat) // 追加字符
37                     num = "" // 清空 num
38                 } else {
39                     res += string(c) // 直接追加字符
40                 }
41             }
42         }
43
44         fmt.Println(res)
45     }
}
```

```
46    }
47
48 // repeatChar 生成重复字符的字符串
49 func repeatChar(c rune, count int) string {
50     result := make([]rune, count)
51     for i := range result {
52         result[i] = c
53     }
54     return string(result)
55 }
```

来自: 华为OD机试2025C卷 – 简易压缩算法 / 一种字符串压缩表示的解压 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 最大矩阵和 (C++ & Python & JAVA & JS & GO)-CSDN博客

最大矩阵和

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

给定一个二维整数矩阵，要在这个矩阵中选出一个子矩阵，使得这个子矩阵内所有的数字和尽量大，我们把这个子矩阵称为和[最大子矩阵](#)，子矩阵的选取原则是原矩阵中一块相互连续的矩形区域。

输入描述

输入的第一行包含2个整数n, m($1 \leq n, m \leq 10$)，表示一个n行m列的矩阵，下面有n行，每行有m个整数，同一行中，每2个数字之间有1个空格，最后一个数字后面没有空格，所有的数字在 $[-1000, 1000]$ 之间。

输出描述

输出一行一个数字，表示选出的和最大子矩阵内所有的数字和。

用例1

输入

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

输出

```
1 20
```

说明

| 一个3*4的矩阵中，后面3列的子矩阵求和加起来等于20， 和最大。

题解

思路： 逻辑分析、贪心

1. 转换思路，将多维问题转换为单维问题：通过枚举子矩阵的开始行和结束行，将枚举的开始行和结束行内的相同列相加就会压缩成一个一维数组。
2. 问题就变成了在一行中求最大连续和子数组了。最大和子数组的状态转移方程： $dp[i] = \max(dp[i-1], 0) + nums[i]$ 。
3. 记录每种情况下的连续子数组和最大值，其中的最大值就是结果。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<climits>
8 using namespace std;
9
10
11 // kadane算法,本质就是贪心
12 int kadane(const vector<int>& arr) {
13     int maxHere = arr[0], maxSoFar = arr[0];
14     for (int i = 1; i < arr.size(); i++) {
15         maxHere = max(arr[i], maxHere + arr[i]);
16         maxSoFar = max(maxSoFar, maxHere);
17     }
18     return maxSoFar;
19 }
20
21
22 int solve(vector<vector<int>>& grid, int n, int m) {
23     int res = INT_MIN;
24     // 枚举上边界
25     for (int top = 0; top < n; top++) {
26         vector<int> colSum(m, 0);
27         // 枚举下标界
28         for (int bottom = top; bottom < n; bottom++) {
29             // 压缩为一维
30             for (int c = 0; c < m; c++) {
31                 colSum[c] += grid[bottom][c];
32             }
33             res = max(res, kadane(colSum));
34         }
35     }
36     return res;
37 }
38
39
40 int main() {
41     int n, m;
42     cin >> n >> m;
43     vector<vector<int>> grid(n, vector<int>(m));
44     for (int i = 0; i < n; i++) {
45         for (int j = 0; j < m; j++) {
```

```
46             cin >> grid[i][j];
47         }
48     }
49     int res = solve(grid, n, m);
50     cout << res;
51 }
```

JAVA

```
1 import java.io.*;
2 import java.util.*;
3
4 public class Main {
5     // kadane算法，本质就是贪心
6     static int kadane(int[] arr) {
7         int maxHere = arr[0], maxSoFar = arr[0];
8         for (int i = 1; i < arr.length; i++) {
9             maxHere = Math.max(arr[i], maxHere + arr[i]);
10            maxSoFar = Math.max(maxSoFar, maxHere);
11        }
12        return maxSoFar;
13    }
14
15    static int solve(int[][] grid, int n, int m) {
16        int res = Integer.MIN_VALUE;
17        // 枚举上边界
18        for (int top = 0; top < n; top++) {
19            int[] colSum = new int[m];
20            // 枚举下边界
21            for (int bottom = top; bottom < n; bottom++) {
22                // 压缩为一维
23                for (int c = 0; c < m; c++) {
24                    colSum[c] += grid[bottom][c];
25                }
26                res = Math.max(res, kadane(colSum));
27            }
28        }
29        return res;
30    }
31
32    public static void main(String[] args) throws IOException {
33        // 缓冲流读入
34        BufferedReader br = new BufferedReader(new InputStreamReader(Syste
m.in));
35        String[] nm = br.readLine().trim().split("\\s+");
36        int n = Integer.parseInt(nm[0]);
37        int m = Integer.parseInt(nm[1]);
38
39        int[][] grid = new int[n][m];
40        for (int i = 0; i < n; i++) {
41            String[] parts = br.readLine().trim().split("\\s+");
42            for (int j = 0; j < m; j++) {
43                grid[i][j] = Integer.parseInt(parts[j]);
44            }
45        }
46    }
47}
```

```
45     }
46
47     int res = solve(grid, n, m);
48     System.out.println(res);
49 }
50 }
```

Python

Plain Text |

```
1 import sys
2
3 # kadane算法，本质就是贪心
4 def kadane(arr):
5     max_here = max_so_far = arr[0]
6     for x in arr[1:]:
7         max_here = max(x, max_here + x)
8         max_so_far = max(max_so_far, max_here)
9     return max_so_far
10
11 def solve(grid, n, m):
12     res = -10**18
13     # 枚举上边界
14     for top in range(n):
15         col_sum = [0] * m
16         # 枚举下边界
17         for bottom in range(top, n):
18             # 压缩为一维
19             for c in range(m):
20                 col_sum[c] += grid[bottom][c]
21             res = max(res, kadane(col_sum))
22     return res
23
24 if __name__ == "__main__":
25     data = sys.stdin.read().strip().split()
26     n = int(data[0])
27     m = int(data[1])
28     grid = []
29     idx = 2
30     for _ in range(n):
31         row = list(map(int, data[idx:idx+m]))
32         grid.append(row)
33         idx += m
34     print(solve(grid, n, m))
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = [];
9
10 // kadane算法，本质就是贪心
11 function kadane(arr) {
12     let maxHere = arr[0], maxSoFar = arr[0];
13     for (let i = 1; i < arr.length; i++) {
14         maxHere = Math.max(arr[i], maxHere + arr[i]);
15         maxSoFar = Math.max(maxSoFar, maxHere);
16     }
17     return maxSoFar;
18 }
19
20 function solve(grid, n, m) {
21     let res = -Infinity;
22     // 枚举上边界
23     for (let top = 0; top < n; top++) {
24         let colSum = new Array(m).fill(0);
25         // 枚举下边界
26         for (let bottom = top; bottom < n; bottom++) {
27             // 压缩为一维
28             for (let c = 0; c < m; c++) {
29                 colSum[c] += grid[bottom][c];
30             }
31             res = Math.max(res, kadane(colSum));
32         }
33     }
34     return res;
35 }
36
37 rl.on("line", (line) => {
38     input.push(...line.trim().split(/\s+/).map(Number));
39 }).on("close", () => {
40     let idx = 0;
41     const n = input[idx++], m = input[idx++];
42     let grid = [];
43     for (let i = 0; i < n; i++) {
44         let row = input.slice(idx, idx + m);
45         idx += m;
```

```
46         grid.push(row);
47     }
48     console.log(solve(grid, n, m));
49 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 // kadane算法，本质就是贪心
10 func kadane(arr []int) int {
11     maxHere, maxSoFar := arr[0], arr[0]
12     for i := 1; i < len(arr); i++ {
13         if arr[i] > maxHere+arr[i] {
14             maxHere = arr[i]
15         } else {
16             maxHere = maxHere + arr[i]
17         }
18         if maxHere > maxSoFar {
19             maxSoFar = maxHere
20         }
21     }
22     return maxSoFar
23 }
24
25 func solve(grid [][]int, n, m int) int {
26     res := -1 << 60 // 很小的数
27     // 枚举上边界
28     for top := 0; top < n; top++ {
29         colSum := make([]int, m)
30         // 枚举下边界
31         for bottom := top; bottom < n; bottom++ {
32             // 压缩为一维
33             for c := 0; c < m; c++ {
34                 colSum[c] += grid[bottom][c]
35             }
36             val := kadane(colSum)
37             if val > res {
38                 res = val
39             }
40         }
41     }
42     return res
43 }
44
45 func main() {
```

```
46     in := bufio.NewReader(os.Stdin)
47     var n, m int
48     fmt.Fscanf(in, "%d %d", &n, &m)
49     grid := make([][]int, n)
50     for i := 0; i < n; i++ {
51         grid[i] = make([]int, m)
52         for j := 0; j < m; j++ {
53             fmt.Fscanf(in, "%d", &grid[i][j])
54         }
55     }
56     fmt.Println(solve(grid, n, m))
57 }
```

| 来自: 华为OD机试2025C卷 – 最大矩阵和 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - IPv4地址转换成整数 (C++ & Python & JAVA & JS & GO)-CSDN博客

IPv4地址转换成整数

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

存在一种虚拟[IPv4地址]，由4小节组成，每节的范围为0~255，以#号间隔，虚拟IPv4地址可以转换为一个32位的整数，例如：

- 128#0#255#255，转换为32位整数的结果为2147549183 (0x8000FFFF)
- 1#0#0#0，转换为32位整数的结果为16777216 (0x01000000)

现以字符串形式给出一个虚拟IPv4地址，限制第1小节的范围为1-128，即每一节范围分别为(1-128) #(0-255) #(0-255) #(0-255)，要求每个IPv4地址只能对应到唯一的整数上。如果是非法IPv4，返回invalid IP

输入描述

输入一行，虚拟IPv4地址格式字符串

输出描述

输出一行，按照要求输出整型或者特定字符

示例1

输入

```
1 100#101#1#5
```

输出

```
1 1684340997
```

示例2

输入

Plain Text |

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

输出

Plain Text |

```
1 invalid IP
```

题解

解题思路：

1. 将输入字符串以 # 进行分割成一个字符串数组，如果数组长度不为4，代表不合法。
2. 遍历检查数组中每个字符串是否合法(只包含数字并且不为空， 不能包含前导零)。
3. 接下来判断数组中的字符串代表的int数字大小是否符合范围就行。
4. 进行求和计算， ip地址每一部分相当8位二进制， $2^8 = 256$ 。

C++实现源码

```
1 #include <iostream>
2 #include <sstream>
3 #include <vector>
4
5 using namespace std;
6
7 // 判断字符串是否为数字
8 bool isNumeric(const string& str) {
9     for (char c : str) {
10         if (!isdigit(c)) {
11             return false; // 如果有非数字字符则返回false
12         }
13     }
14     return true; // 全部为数字则返回true
15 }
16
17 int main() {
18     string input;
19     getline(cin, input); // 获取输入的字符串
20
21     // 将输入的字符串按照"#"分割成4个小节
22     stringstream ss(input);
23     vector<string> ipSections;
24     string section;
25     while (getline(ss, section, '#')) {
26         ipSections.push_back(section);
27     }
28
29     if (ipSections.size() != 4) { // 如果分割后的小节数量不等于4，则说明输入的IP
v4地址格式不正确
30         cout << "invalid IP" << endl;
31         return 0; // 结束程序
32     }
33
34     // 遍历每个部分进行检查
35     for (const string& section : ipSections) {
36         if (section.empty() || !isNumeric(section)) { // 检查是否为空或者是否
每部分都是数字
37             cout << "invalid IP" << endl;
38             return 0; // 结束程序
39         }
40
41         // 检查前导零的情况
42         if (section.length() > 1 && section[0] == '0') {
43             cout << "invalid IP" << endl;
```

```

44         return 0; // 结束程序
45     }
46 }
47
48 // 检查第一个小节的大小范围
49 int firstSection = stoi(ipSections[0]); // 将第一个小节转换为整数
50 if (firstSection < 1 || firstSection > 128) { // 如果第一个小节的值不在1~128的范围内
51     cout << "invalid IP" << endl;
52     return 0; // 结束程序
53 }
54
55 // 检查其余3个小节的范围
56 for (int i = 1; i < 4; i++) {
57     int sectionValue = stoi(ipSections[i]); // 将当前小节转换为整数
58     if (sectionValue < 0 || sectionValue > 255) { // 如果不在0~255范围内
59         cout << "invalid IP" << endl;
60         return 0; // 结束程序
61     }
62 }
63
64 // 计算最终的32位整数
65 long ipValue = 0;
66 for (int i = 0; i < 4; i++) {
67     // 也可以使用位运算, 计算效率更高
68     //ipValue = ipValue << 8 + stoi(ipSections[i]);
69     ipValue = ipValue * 256 + stoi(ipSections[i]); // 每个小节对应一个字节, 计算最终的整数值
70 }
71
72 cout << ipValue << endl; // 输出最终的32位整数
73 return 0;
74 }

```

JAVA源码实现

```
1 import java.util.*;
2
3 public class Main {
4
5     // 判断字符串是否为数字
6     private static boolean isNumeric(String str) {
7         for (char c : str.toCharArray()) {
8             if (!Character.isDigit(c)) {
9                 return false; // 如果有非数字字符则返回 false
10            }
11        }
12        return true; // 全部为数字则返回 true
13    }
14
15    public static void main(String[] args) {
16        Scanner scanner = new Scanner(System.in);
17        String input = scanner.nextLine(); // 获取输入的字符串
18        scanner.close();
19
20        // 将输入的字符串按照 "#" 分割成 4 个小节
21        String[] ipSections = input.split("#");
22
23        if (ipSections.length != 4) { // 如果分割后的小节数量不等于 4, 则说明输入的 IPv4 地址格式不正确
24            System.out.println("invalid IP");
25            return; // 结束程序
26        }
27
28        // 遍历每个部分进行检查
29        for (String section : ipSections) {
30            if (section.isEmpty() || !isNumeric(section)) { // 检查是否为空
或者是否每部分都是数字
31                System.out.println("invalid IP");
32                return; // 结束程序
33            }
34
35            // 检查前导零的情况
36            if (section.length() > 1 && section.charAt(0) == '0') {
37                System.out.println("invalid IP");
38                return; // 结束程序
39            }
40        }
41
42        // 检查第一个小节的大小范围
43    }
```

```
44         int firstSection = Integer.parseInt(ipSections[0]); // 将第一个小节
45 转换为整数
46         if (firstSection < 1 || firstSection > 128) { // 如果第一个小节的值不
47 在 1~128 的范围内
48             System.out.println("invalid IP");
49             return; // 结束程序
50     }
51
52         // 检查其余 3 个小节的范围
53         for (int i = 1; i < 4; i++) {
54             int sectionValue = Integer.parseInt(ipSections[i]); // 将当前小
55 节转换为整数
56             if (sectionValue < 0 || sectionValue > 255) { // 如果不在 0~25
57 5 范围内
58                 System.out.println("invalid IP");
59                 return; // 结束程序
60             }
61         }
62
63         // 计算最终的 32 位整数
64         long ipValue = 0;
65         for (int i = 0; i < 4; i++) {
66             ipValue = ipValue * 256 + Integer.parseInt(ipSections[i]); //
67 每个小节对应一个字节，计算最终的整数值
68         }
69
70         System.out.println(ipValue); // 输出最终的 32 位整数
71     }
72 }
```

Python实现源码

```
1 def main():
2     # 获取输入的字符串
3     input_str = input().strip()
4
5     # 将输入的字符串按照"#"分割成4个小节
6     ip_sections = input_str.split('#')
7
8     if len(ip_sections) != 4:  # 如果分割后的小节数量不等于4，则说明输入的IPv4地
9         print("invalid IP")
10        return
11
12     # 遍历每个部分进行检查
13     for section in ip_sections:
14         if not section or not section.isdigit():  # 检查是否为空或者是否每部分
15             print("invalid IP")
16             return
17
18         # 检查前导零的情况
19         if len(section) > 1 and section[0] == '0':
20             print("invalid IP")
21             return
22
23         # 检查第一个小节的大小范围
24         first_section = int(ip_sections[0])  # 将第一个小节转换为整数
25         if first_section < 1 or first_section > 128:  # 如果第一个小节的值不在1~1
26             print("invalid IP")
27             return
28
29         # 检查其余3个小节的范围
30         for section in ip_sections[1:]:
31             section_value = int(section)  # 将当前小节转换为整数
32             if section_value < 0 or section_value > 255:  # 如果不在0~255范围内
33                 print("invalid IP")
34                 return
35
36         # 计算最终的32位整数
37         ip_value = 0
38         for section in ip_sections:
39             ip_value = ip_value * 256 + int(section)  # 每个小节对应一个字节，计算
40             最终的整数值
41         print(ip_value)  # 输出最终的32位整数
```

```
42  
43     if __name__ == "__main__":  
44         main()
```

Go实现源码

```
1 package main
2
3 import (
4     "fmt"
5     "strconv"
6     "strings"
7 )
8
9 // isNumeric 判断字符串是否为数字
10 func isNumeric(str string) bool {
11     for _, c := range str {
12         if c < '0' || c > '9' {
13             return false
14         }
15     }
16     return true
17 }
18
19 func main() {
20     // 获取输入的字符串
21     var input string
22     fmt.Scanln(&input)
23     input = strings.TrimSpace(input)
24
25     // 将输入的字符串按照"#"分割成4个小节
26     ipSections := strings.Split(input, "#")
27
28     if len(ipSections) != 4 { // 如果分割后的小节数量不等于4，则说明输入的IPv4地址
29         // 格式不正确
30         fmt.Println("invalid IP")
31         return
32     }
33
34     // 遍历每个部分进行检查
35     for _, section := range ipSections {
36         if section == "" || !isNumeric(section) { // 检查是否为空或者是否每部分都是
37             // 数字
38             fmt.Println("invalid IP")
39             return
40         }
41         // 检查前导零的情况
42         if len(section) > 1 && section[0] == '0' {
43             fmt.Println("invalid IP")
44             return
45         }
46     }
47 }
```

```
44     }
45 }
46
47 // 检查第一个小节的大小范围
48 firstSection, err := strconv.Atoi(ipSections[0]) // 将第一个小节转换为整数
49 if err != nil || firstSection < 1 || firstSection > 128 { // 如果第一个小
50   节的值不在1~128的范围内
51   fmt.Println("invalid IP")
52   return
53 }
54
55 // 检查其余3个小节的范围
56 for _, section := range ipSections[1:] {
57   sectionValue, err := strconv.Atoi(section) // 将当前小节转换为整数
58   if err != nil || sectionValue < 0 || sectionValue > 255 { // 如果不在0~
59   255范围内
60   fmt.Println("invalid IP")
61   return
62 }
63
64 // 计算最终的32位整数
65 ipValue := 0
66 for _, section := range ipSections {
67   sectionValue, _ := strconv.Atoi(section)
68   ipValue = ipValue*256 + sectionValue // 每个小节对应一个字节，计算最终的整数
69   值
70 }
71 fmt.Println(ipValue) // 输出最终的32位整数
72 }
```

JS实现源码

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 // 判断字符串是否为数字
9 function isNumeric(str) {
10     for (let c of str) {
11         if (c < '0' || c > '9') {
12             return false; // 如果有非数字字符则返回 false
13         }
14     }
15     return true; // 全部为数字则返回 true
16 }
17
18 rl.on("line", (input) => {
19     // 将输入的字符串按照 "#" 分割成 4 个小节
20     const ipSections = input.trim().split("#");
21
22     if (ipSections.length !== 4) { // 小节数量不等于 4
23         console.log("invalid IP");
24         return;
25     }
26
27     // 遍历每个部分进行检查
28     for (let section of ipSections) {
29         if (section.length === 0 || !isNumeric(section)) { // 空或非数字
30             console.log("invalid IP");
31             return;
32         }
33
34         // 检查前导零的情况
35         if (section.length > 1 && section[0] === '0') {
36             console.log("invalid IP");
37             return;
38         }
39     }
40
41     // 检查第一个小节的范围
42     const firstSection = parseInt(ipSections[0], 10);
43     if (firstSection < 1 || firstSection > 128) {
44         console.log("invalid IP");
45         return;
46     }
47 }
```

```
46     }
47
48     // 检查其余三个小节的范围
49     for (let i = 1; i < 4; i++) {
50         const sectionValue = parseInt(ipSections[i], 10);
51         if (sectionValue < 0 || sectionValue > 255) {
52             console.log("invalid IP");
53             return;
54         }
55     }
56
57     // 计算最终的 32 位整数
58     let ipValue = 0;
59     for (let i = 0; i < 4; i++) {
60         ipValue = ipValue * 256 + parseInt(ipSections[i], 10);
61     }
62
63     console.log(ipValue);
64 });

});
```

来自: 华为OD机试2025C卷 – IPv4地址转换成整数 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025C卷 - 国际移动用户识别码(IMS)匹配 (C++ & Python & JAVA & JS & GO)_ 华为机试 国际移动用户识别码(imsi)匹配-CSDN 博客

国际移动用户识别码(IMS)匹配

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

| 华为OD机试2025C卷 100分题型

题目描述

小明是核心网工程师，客户交给小明一个任务：给定一个网络配置列表，每个配置是一个字符串，仅有数字和“*”、“？”符号组成。

输入用户的IMSI(国际移动用户识别码)，根据以下规则匹配配置列表：

- “*”匹配0个或连续多个任意字符。
- “?”匹配下标为奇数的单个字符，比如123?中的”?”可以匹配123456789012345下标为3的字符’4’，下标从0开始。

输入描述

输入第一行为网络配置列表，列表中的每个配置是由数字和“*”、“？”组成的字符串，每个字符串中“*”不会超过一个，“？”若干，网络配置列表长度小于200，每个字符串以英文逗号隔开。

输入第二行为用户的IMSI(国际移动用户识别码)，仅有数字组成，长度等于15.

备注

| 保证输入格式正确，无需考虑格式错误.

输出描述

输出为满足匹配规则的配置字符串列表，列表按字典序升序输出，每个字符以英文逗号隔开。若没有满足条件的配置，则返回字符串“null”

用例1

输入

Plain Text

```
1 1234567,1234567*
2 123456789012345
```

输出

Plain Text

```
1 1234567*
```

说明

*可以匹配0或多个任意字符，故输出1234567 *

用例2

输入

Plain Text

```
1 123?????????345,123????*????345
2 123456789012345
```

输出

Plain Text

```
1 null
```

说明

'?'字符只能匹配IMSI中为奇数下标的字符，故都不符合要求，返回null

题解

思路： 模拟实现

1. 此题主要注意三个点

- ? 能且只能匹配一个字符，并且匹配字符要位于待匹配字符奇数位置。
- * 能匹配0或多个任意字符，并且 * 最多只会出现依次。
- 根据上面两个点，其实也可以推断出如果模式字符串中不包含 *，两个字符串长度需要相等。不过需要额外考虑一种情况，就是 123456789012345*, 123456789012345，就是前15个字符

已经和IMSI匹配好了, pattern额外最后多了一个*的情况, 这种也是可以匹配成功的.

1. 明白上面两个点之后, 从前往后进行一一匹配就行, 处理逻辑如下, 我们使用 `pattern` 表示配置, `str` 比较用户的IMSI:
 - a. `pattern[i] == str[i]` : 跳过, 比较下一个字符。
 - b. `pattern[i] != str[i] && pattern[i] == '?'` : 此时只需要判断i是否为偶数, 为偶数的话说明不能匹配, 直接匹配失败。
 - c. `pattern[i] != str[i] && pattern[i] == '*'` : 其实遇到 `*` 之后, 并且题目已知 `*` 最多只会出现1次, 此时只需要比较后缀即可。举个例子 `pattern = 123%45, str = 123456789012345`, 知道`*`可以匹配任意多个字符, 此时我们只需要考虑pattern在 `*` 之后后缀 `45` 能否和str末尾能否匹配上即可。能匹配上就说明能匹配成功, 否则会失败。后缀匹配阶段也需要考虑?
2. 根据上面两点进行模拟即可, 找出匹配的配置加入数组。如果不存在满足条件的输出 `null`, 否则把满足要求的配置升序之后输出。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11 // 通用 split 函数
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 // 匹配函数
27 bool judge(string& pattern, string& str) {
28     // 直接匹配成功
29     if (pattern == "*") {
30         return true;
31     }
32     int i = 0;
33     int n = str.size();
34     int m = pattern.size();
35     for (; i < str.size() && i < pattern.size(); i++) {
36         if (pattern[i] == str[i]) {
37             continue;
38         } else if (pattern[i] == '?') {
39             // 偶数位置不能匹配 又不相同
40             if (i % 2 == 0) {
41                 return false;
42             }
43         } else if (pattern[i] == '*') {
44             // 如果是pattern最后一个肯定可以匹配成功
45             if (i == m - 1) {
```

```

46             return true;
47         }
48         string suffix = pattern.substr(i+1);
49         int pos = suffix.size() - 1;
50         // 后缀进行匹配
51         for (int j = n - 1; j >= i && pos >= 0; j--,pos--) {
52             if (suffix[pos] == str[j]) {
53                 continue;
54             } else if (suffix[pos] == '?') {
55                 if (j % 2 == 0) {
56                     break;
57                 }
58             }
59         }
60         // 判断后缀是否匹配成功
61         return pos < 0;
62     } else {
63         return false;
64     }
65 }
66 // 从这里返回说明前15格字符不存在* 所以必需两者长度相同/ 或者pattern.size 长度
67 比str大一,并且最后一个为*
68     return n == m || (m == n + 1 && pattern[m-1] == '*');
69 }

70 int main() {
71     string input;
72     getline(cin, input);
73     vector<string> settings = split(input, ",");
74     string number;
75     getline(cin, number);

76     vector<string> res;
77     int n = settings.size();
78     for (int i = 0; i < n; i++) {
79         string pattern = settings[i];
80         bool flag = judge(pattern, number);
81         if (flag) {
82             res.push_back(pattern);
83         }
84     }
85 }
86 if (res.empty()) {
87     cout << "null";
88     return 0;
89 }
90 // 字典序升序
91 sort(res.begin(), res.end());
92 int m = res.size();

```

```
93     for (int i = 0; i < m; i++) {
94         cout << res[i];
95         if (i != m - 1) {
96             cout << ",";
97         }
98     }
99     return 0;
100 }
```

JAVA

```

1 import java.util.*;
2
3 public class Main {
4     // 匹配函数
5     public static boolean judge(String pattern, String str) {
6         if (pattern.equals("*")) return true;
7         int n = str.length(), m = pattern.length();
8         int i = 0;
9         for (; i < n && i < m; i++) {
10             char pc = pattern.charAt(i), sc = str.charAt(i);
11             if (pc == sc) continue;
12             else if (pc == '?') {
13                 // 偶数位置不能匹配又不相同
14                 if (i % 2 == 0) return false;
15             } else if (pc == '*') {
16                 // pattern的最后一位*肯定可以匹配完成
17                 if (i == m - 1) return true;
18                 // pattern的后缀必须完整匹配
19                 String suffix = pattern.substring(i + 1);
20                 int pos = suffix.length() - 1;
21                 for (int j = n - 1; j >= i && pos >= 0; j--, pos--) {
22                     char sp = suffix.charAt(pos), sj = str.charAt(j);
23                     if (sp == sj) continue;
24                     else if (sp == '?') {
25                         if (j % 2 == 0) break;
26                     } else break;
27                 }
28                 return pos < 0;
29             } else return false;
30         }
31         // 从这里返回说明前15格字符不存在* 所以必需两者长度相同/ 或者pattern.size
32         // 长度比str大一，并且最后一个为*
33         return n == m || (m == n + 1 && pattern.charAt(m-1) == '*');
34     }
35
36     public static void main(String[] args) {
37         Scanner sc = new Scanner(System.in);
38         String[] settings = sc.nextLine().split(",");
39         String number = sc.nextLine();
40         List<String> res = new ArrayList<>();
41         for (String pattern : settings) {
42             if (judge(pattern, number)) res.add(pattern);
43         }
44         if (res.isEmpty()) {
45             System.out.println("null");
46         }
47     }
48 }
```

```
45     } else {
46         Collections.sort(res);
47         System.out.println(String.join(", ", res));
48     }
49 }
50 }
```

Python

```

1 # 匹配函数
2 def judge(pattern, s):
3     if pattern == '*':
4         return True
5     n, m = len(s), len(pattern)
6     i = 0
7     while i < n and i < m:
8         if pattern[i] == s[i]:
9             pass
10        elif pattern[i] == '?':
11            # 偶数位置不能匹配又不相同
12            if i % 2 == 0:
13                return False
14        elif pattern[i] == '*':
15            # pattern末尾为*
16            if i == m - 1:
17                return True
18            # 后缀匹配
19            suffix = pattern[i + 1:]
20            pos = len(suffix) - 1
21            j = n - 1
22            while j >= i and pos >= 0:
23                if suffix[pos] == s[j]:
24                    pass
25                elif suffix[pos] == '?':
26                    if j % 2 == 0:
27                        break
28                    else:
29                        break
30                    j -= 1
31                    pos -= 1
32            return pos < 0
33        else:
34            return False
35        i += 1
36    # 从这里返回说明前15格字符不存在* 所以必需两者长度相同/ 或者pattern.size 长度比s
37    # tr大一,并且最后一个为*
38    return m == n or (m == n + 1 and pattern[m-1] == '*')
39
40 settings = input().split(',')
41 number = input()
42 res = []
43 for pattern in settings:
44     if judge(pattern, number):

```

```
45         res.append(pattern)
46
47     if not res:
48         print("null")
49     else:
50         print(", ".join(sorted(res)))
```

JavaScript

```

1  const rl = require('readline').createInterface({
2      input: process.stdin,
3      output: process.stdout
4 });
5
6  let lines = [];
7  rl.on('line', line => {
8      lines.push(line);
9      if (lines.length === 2) {
10          const settings = lines[0].split(',');
11          const number = lines[1];
12          const res = [];
13
14          // 匹配函数
15          function judge(pattern, str) {
16              if (pattern === '*') return true;
17              let n = str.length, m = pattern.length;
18              let i = 0;
19              for (; i < n && i < m; i++) {
20                  if (pattern[i] === str[i]) continue;
21                  else if (pattern[i] === '?') {
22                      // 偶数位置不能匹配又不相同
23                      if (i % 2 === 0) return false;
24                  } else if (pattern[i] === '*') {
25                      // 达到pattern的末尾并且为*肯定可以匹配
26                      if (i === m - 1) return true;
27                      const suffix = pattern.slice(i + 1);
28                      // 后缀匹配
29                      let pos = suffix.length - 1;
30                      for (let j = n - 1; j >= i && pos >= 0; j--, pos--) {
31                          if (suffix[pos] === str[j]) continue;
32                          else if (suffix[pos] === '?') {
33                              if (j % 2 === 0) break;
34                          } else break;
35                      }
36                      return pos < 0;
37                  } else return false;
38              }
39              //从这里返回说明前15格字符不存在* 所以必需两者长度相同/ 或者pattern.size 长度比str大一,并且最后一个为*
40              return m === n || (m === n + 1 && pattern[m-1] === '*');
41          }
42
43          for (const pattern of settings) {
44              if (judge(pattern, number)) {

```

```
45             res.push(pattern);
46         }
47     }
48
49     if (res.length === 0) console.log("null");
50     else console.log(res.sort().join(','));
51
52     rl.close();
53 }
54});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strings"
9 )
10
11 // 匹配函数
12 func judge(pattern, str string) bool {
13     if pattern == "*" {
14         return true
15     }
16     n, m := len(str), len(pattern)
17     i := 0
18     for i < n && i < m {
19         pc, sc := pattern[i], str[i]
20         if pc == sc {
21             // continue
22         } else if pc == '?' {
23             // 偶数位置不能匹配又不相同
24             if i%2 == 0 {
25                 return false
26             }
27         } else if pc == '*' {
28             if i == m-1 {
29                 return true
30             }
31             suffix := pattern[i+1:]
32             pos := len(suffix) - 1
33             for j := n - 1; j >= i && pos >= 0; j, pos = j-1, pos-1 {
34                 sp := suffix[pos]
35                 sj := str[j]
36                 if sp == sj {
37                     // continue
38                 } else if sp == '?' {
39                     if j%2 == 0 {
40                         break
41                     }
42                 } else {
43                     break
44                 }
45             }
46         }
47     }
48 }
```

```
46         return pos < 0
47     } else {
48         return false
49     }
50     i++
51 }
52 return m == n || (m == n + 1 && pattern[m-1] == '*')
53 }
54
55 func main() {
56     scanner := bufio.NewScanner(os.Stdin)
57     scanner.Scan()
58     settings := strings.Split(scanner.Text(), ",")
59     scanner.Scan()
60     number := scanner.Text()
61
62     var res []string
63     for _, pattern := range settings {
64         if judge(pattern, number) {
65             res = append(res, pattern)
66         }
67     }
68
69     if len(res) == 0 {
70         fmt.Println("null")
71     } else {
72         sort.Strings(res)
73         fmt.Println(strings.Join(res, ","))
74     }
75 }
```

来自: 华为OD机试 2025C卷 – 国际移动用户识别码(IMSI)匹配 (C++ & Python & JAVA & JS & GO)
华为机试 国际移动用户识别码(imsi)匹配–CSDN博客

华为OD机考2025C卷 - 字母组合 (C++ & Python & JAVA & JS & GO)-CSDN博客

字母组合

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 200分题型

题目描述

每个数字关联多个字母，关联关系如下：

- 0 关联 “a”, “b”, “c”
- 1 关联 “d”, “e”, “f”
- 2 关联 “g”, “h”, “i”
- 3 关联 “j”, “k”, “l”
- 4 关联 “m”, “n”, “o”
- 5 关联 “p”, “q”, “r”
- 6 关联 “s”, “t”
- 7 关联 “u”, “v”
- 8 关联 “w”, “x”
- 9 关联 “y”, “z”

输入一串数字后，通过数字和字母的对应关系可以得到多个字母[字符串]（要求按照数字的顺序组合字母字符串）；

屏蔽字符串：屏蔽字符串中的所有字母不能同时在输出的字符串出现，如屏蔽字符串是abc，则要求字符串中不能同时出现a,b,c，但是允许同时出现a,b或a,c或b,c等；

给定一个数字字符串和一个屏蔽字符串，输出所有可能的字符组合；

例如输入数字字符串78和屏蔽字符串ux，输出结果为uw, vw, vx；数字字符串78，可以得到如下字符串uw, ux, vw, vx；由于ux是屏蔽字符串，因此排除ux，最终的输出是uw, vw, vx；

输入描述

第一行输入为一串数字字符串，数字字符串中的数字不允许重复，数字字符串的长度大于0，小于等于5；

第二行输入是屏蔽字符串，屏蔽字符串的长度一定小于数字字符串的长度，屏蔽字符串中字符不会重复；

输出描述

输出可能的字符串组合

注：字符串之间使用逗号隔开，最后一个字符串后携带逗号

用例1

输入

```
▼ Plain Text |  
1 78  
2 ux
```

输出

```
▼ Plain Text |  
1 uw, vw, vx,
```

说明

| ux完全包含屏蔽字符串ux，因此剔除

用例2

输入

```
▼ Plain Text |  
1 78  
2 x
```

输出

```
▼ Plain Text |  
1 uw, vw,
```

题解

思路：题目数据量十分小，直接使用 递归回溯 枚举出数字字符串能组成的所有字符串，取其中 不包含所有过滤字符 的字符串添加到结果中。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <unordered_set>
4 using namespace std;
5
6 vector<string> map = {"abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv",
7 "wx", "yz"};
8 unordered_set<char> filterSet;
9
10 bool isContainsAllFilterChars(unordered_set<char>& used) {
11     for (auto x : filterSet) {
12         if (used.find(x) == used.end()) {
13             return true;
14         }
15     }
16     return false;
17 }
18
19 void dfs(vector<string>& letters, int index, string path, string& res, string& filter, unordered_set<char>& used) {
20     if (index == letters.size()) {
21         if (isContainsAllFilterChars(used)) {
22             res += path + ",";
23         }
24         return;
25     }
26
27     // 对于每个数字，遍历其对应的字母字符串
28     for (int i = 0; i < letters[index].length(); i++) {
29         char c = letters[index][i];
30         // 递归回溯
31         path += c;
32         used.insert(c);
33         dfs(letters, index + 1, path, res, filter, used);
34         path.pop_back();
35         used.erase(c);
36     }
37 }
38
39 int main() {
40     string digits;
41     cin >> digits;
42     string filter;
43     cin >> filter;
```

```
44
45     for (int i = 0; i < filter.size(); i++) {
46         filterSet.insert(filter[i]);
47     }
48
49     // 根据数字字符串得到每个数字对应的字母字符串
50     vector<string> letters(digits.length());
51     for (int i = 0; i < digits.length(); i++) {
52         letters[i] = map[digits[i] - '0'];
53     }
54
55     // 用于存储结果的字符串
56     string res = "";
57     // 开始进行深度优先搜索
58     unordered_set<char> used;
59     dfs(letters, 0, "", res, filter, used);
60
61     // 输出结果
62     cout << res << endl;
63
64     return 0;
65 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     static List<String> map = Arrays.asList("abc", "def", "ghi", "jkl", "m
no", "pqr", "st", "uv", "wx", "yz");
5     static Set<Character> filterSet = new HashSet<>();
6
7     // 判断是否包含所有过滤字符
8     static boolean isContainsAllFilterChars(Set<Character> used) {
9         for (char x : filterSet) {
10             if (!used.contains(x)) {
11                 return true;
12             }
13         }
14         return false;
15     }
16
17     // 递归回溯
18     static void dfs(List<String> letters, int index, StringBuilder path, S
tringBuilder res, Set<Character> used) {
19         if (index == letters.size()) {
20             if (isContainsAllFilterChars(used)) {
21                 res.append(path).append(",");
22             }
23             return;
24         }
25
26         // 遍历当前数字对应的字母
27         for (char c : letters.get(index).toCharArray()) {
28             path.append(c);
29             used.add(c);
30             dfs(letters, index + 1, path, res, used);
31             path.deleteCharAt(path.length() - 1);
32             used.remove(c);
33         }
34     }
35
36     public static void main(String[] args) {
37         Scanner scanner = new Scanner(System.in);
38         String digits = scanner.next();
39         String filter = scanner.next();
40         scanner.close();
41
42         // 记录需要筛选的字符
43         for (char c : filter.toCharArray()) {
```

```
44         filterSet.add(c);
45     }
46
47     // 获取对应的字母映射
48     List<String> letters = new ArrayList<>();
49     for (char c : digits.toCharArray()) {
50         letters.add(map.get(c - '0'));
51     }
52
53     StringBuilder res = new StringBuilder();
54     dfs(letters, 0, new StringBuilder(), res, new HashSet<>());
55
56     // 输出结果
57     System.out.println(res.toString());
58 }
59 }
```

Python

```
1 # 定义映射关系
2 char_map = ["abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "wx", "yz"]
3 filter_set = set()
4
5 # 判断是否包含所有过滤字符
6 def is_contains_all_filter_chars(used):
7     return any(x not in used for x in filter_set)
8
9 # 递归回溯
10 def dfs(letters, index, path, res, used):
11     if index == len(letters):
12         if is_contains_all_filter_chars(used):
13             res.append("".join(path) + ",")
14         return
15
16     # 遍历当前数字对应的字母
17     for c in letters[index]:
18         path.append(c)
19         used.add(c)
20         dfs(letters, index + 1, path, res, used)
21         path.pop()
22         used.remove(c)
23
24 # 处理输入
25 digits = input().strip()
26 filter_chars = input().strip()
27
28 filter_set = set(filter_chars)
29 letters = [char_map[int(d)] for d in digits]
30
31 # 开始递归
32 res = []
33 dfs(letters, 0, [], res, set())
34
35 # 输出结果
36 print("".join(res))
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 const charMap = ["abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "w
x", "yz"];
9 let filterSet = new Set();
10
11 // 判断是否包含全部单词
12 function isContainsAllFilterChars(used) {
13     for (let x of filterSet) {
14         if (!used.has(x)) {
15             return true;
16         }
17     }
18     return false;
19 }
20
21 function dfs(letters, index, path, res, used) {
22     if (index === letters.length) {
23         if (isContainsAllFilterChars(used)) {
24             res.push(path.join("") + ",");
25         }
26         return;
27     }
28
29     // 遍历当前数字对应的字母
30     for (let c of letters[index]) {
31         path.push(c);
32         used.add(c);
33         dfs(letters, index + 1, path, res, used);
34         path.pop();
35         used.delete(c);
36     }
37 }
38
39 rl.question("", (digits) => {
40     rl.question("", (filter) => {
41         filterSet = new Set(filter);
42         let letters = digits.split("").map(d => charMap[parseInt(d)]);
43
44         let res = [];
```

```
45     dfs(letters, 0, [], res, new Set());
46
47     console.log(res.join(""));
48     rl.close();
49   });
50 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 var charMap = []string{"abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv",
11     "wx", "yz"}
12 var filterSet map[rune]struct{}
13
14 // 判断是否包含所有过滤字符
15 func isContainsAllFilterChars(used map[rune]struct{}) bool {
16     for x := range filterSet {
17         if _, exists := used[x]; !exists {
18             return true
19         }
20     }
21     return false
22 }
23
24 // 递归回溯
25 func dfs(letters []string, index int, path []rune, res *[]string, used map[rune]struct{}) {
26     if index == len(letters) {
27         if isContainsAllFilterChars(used) {
28             *res = append(*res, string(path)+",")
29         }
30         return
31     }
32
33     // 遍历当前数字对应的字母
34     for _, c := range letters[index] {
35         path = append(path, c)
36         used[c] = struct{}{}
37         dfs(letters, index+1, path, res, used)
38         path = path[:len(path)-1]
39         delete(used, c)
40     }
41 }
42 func main() {
43     scanner := bufio.NewScanner(os.Stdin)
```

```
44
45     // 读取数字字符串
46     scanner.Scan()
47     digits := scanner.Text()
48
49     // 读取过滤字符
50     scanner.Scan()
51     filter := scanner.Text()
52
53     // 记录需要筛选的字符
54     filterSet = make(map[rune]struct{})
55     for _, c := range filter {
56         filterSet[c] = struct{}{}
57     }
58
59     // 获取对应的字母映射
60     letters := make([]string, len(digits))
61     for i, c := range digits {
62         letters[i] = charMap[c-'0']
63     }
64
65     // 结果存储
66     var res []string
67     dfs(letters, 0, []rune{}, &res, make(map[rune]struct{}))
68
69     // 输出结果
70     fmt.Println(strings.Join(res, ""))
71 }
```

来自: 华为OD机考2025C卷 – 字母组合 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 猴子爬山 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 – 猴子爬山 (C++ & Python & JAVA & JS & GO)

猴子爬山

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

一天一只顽猴想去从山脚爬到山顶，途中经过一个有N个台阶的阶梯，但是这猴子有一个习惯：每一次只能跳1步或跳3步，试问猴子通过这个阶梯有多少种不同的跳跃方式？

输入描述

输入只有一个整数N (0<N<=50) 此阶梯有多少个台阶。

输出描述

输出有多少种跳跃方式（解决方案数）。

用例1

输入

```
1 50
```

输出

```
1 122106097
```

用例2

输入

Plain Text |

1 3

输出

Plain Text |

1 2

题解

思路：非常经典一道 动态规划 题。

- 定义 `dp[]` 数组，其中 `dp[i]` 表示跳到i台阶的方案数量。
- 状态转移方程 `dp[i] = dp[i-1] + dp[i-3]` ,只需要注意i大小对应的边界问题就行。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 using namespace std;
8
9 int main() {
10     int n;
11     cin >> n;
12     vector<int> dp(n+1, 0);
13     dp[0] = 1;
14     for (int i = 0; i <= n; i++) {
15         // 走一步
16         if (i > 0) {
17             dp[i] += dp[i-1];
18         }
19         // 走三步
20         if (i > 2) {
21             dp[i] += dp[i-3];
22         }
23     }
24
25     cout << dp[n];
26     return 0;
27 }
```

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         int n = scanner.nextInt();
7         scanner.close();
8
9         int[] dp = new int[n + 1];
10        dp[0] = 1;
11
12        for (int i = 0; i <= n; i++) {
13            // 走一步
14            if (i > 0) {
15                dp[i] += dp[i - 1];
16            }
17            // 走三步
18            if (i > 2) {
19                dp[i] += dp[i - 3];
20            }
21        }
22
23        System.out.println(dp[n]);
24    }
25 }
```

Python

```
1 import sys
2
3 def main():
4     n = int(sys.stdin.readline().strip())
5
6     dp = [0] * (n + 1)
7     dp[0] = 1
8
9     for i in range(n + 1):
10         # 走一步
11         if i > 0:
12             dp[i] += dp[i - 1]
13         # 走三步
14         if i > 2:
15             dp[i] += dp[i - 3]
16
17     print(dp[n])
18
19 if __name__ == "__main__":
20     main()
```

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", (line) => {
9     let n = parseInt(line.trim());
10    let dp = new Array(n + 1).fill(0);
11    dp[0] = 1;
12
13    for (let i = 0; i <= n; i++) {
14        // 走一步
15        if (i > 0) {
16            dp[i] += dp[i - 1];
17        }
18        // 走三步
19        if (i > 2) {
20            dp[i] += dp[i - 3];
21        }
22    }
23
24    console.log(dp[n]);
25    rl.close();
26});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8 )
9
10 func main() {
11     reader := bufio.NewReader(os.Stdin)
12     nStr, _ := reader.ReadString('\n')
13     n, _ := strconv.Atoi(nStr[:len(nStr)-1]) // 去掉换行符
14
15     dp := make([]int, n+1)
16     dp[0] = 1
17
18     for i := 0; i <= n; i++ {
19         // 走一步
20         if i > 0 {
21             dp[i] += dp[i-1]
22         }
23         // 走三步
24         if i > 2 {
25             dp[i] += dp[i-3]
26         }
27     }
28
29     fmt.Println(dp[n])
30 }
```

| 来自: 华为OD机考2025C卷 – 猴子爬山 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 数组去重和排序 (C++ & Python & JAVA & JS & GO)-CSDN博客

数组去重和排序

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

给定一个乱序的数组，[删除所有](#)的重复元素，使得每个元素只出现一次，并且按照出现的次数从高到低进行排序，相同出现次数按照第一次出现顺序进行先后排序。

输入描述

一个数组

输出描述

去[重排序](#)后的数组

用例1

输入

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

输出

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

说明

数组大小不超过100 数组元素值大小不超过100。

题解

思路：一道简单的 [哈希表和自定义排序](#) 练习题

1. 使用哈希表进行统计：定义 `positionMap` 和 `countMp` 分别记录每个数字出现的第一次下标和出现的次数
2. 从 `positionMap` 或者 `countMp` 中可以知道出现的所有数字，使用一个数组保存出现过的数组
3. 数组自定排序：数字去重之后依靠两个哈希表进行自定义排序。先按照出现次数降序，出现次序相同按照首次出现位置升序
4. 输出结果排序之后数组。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<map>
8 using namespace std;
9
10
11 // 通用 split 函数
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 int main() {
27     string s;
28
29     getline(cin, s);
30     vector<string> ans = split(s, ",");
31     // 记录数字出现的首次位置
32     map<string, int> firstPosition;
33     // 统计数字出现的次数
34     map<string, int> count;
35     int n = ans.size();
36     for (int i = 0; i < n; i++) {
37         string num = ans[i];
38         if (firstPosition.find(num) == firstPosition.end()) {
39             firstPosition[num] = i;
40         }
41         count[num]++;
42     }
43     // 去重后的数字
44     vector<string> res;
45     for (auto p : firstPosition) {
```

```
46     res.push_back(p.first);
47 }
48 // 自定义排序 先按出现降序，相同次数按首位置的升序
49 sort(res.begin(), res.end(), [&](const string &a, const string &b) {
50     if (count[a] == count[b]) {
51         return firstPosition[a] < firstPosition[b];
52     }
53     return count[a] > count[b];
54 });
55
56 n = res.size();
57 for (int i = 0; i < n; i++) {
58     cout << res[i];
59     if (i != n - 1) {
60         cout << ",";
61     }
62 }
63
64 return 0;
}
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         String s = scanner.nextLine();
7         scanner.close();
8
9
10        String[] ans = s.split(",");
11
12        // 记录数字首次出现的位置
13        Map<String, Integer> firstPosition = new HashMap<>();
14        // 统计数字出现的次数
15        Map<String, Integer> count = new HashMap<>();
16        int n = ans.length;
17
18        for (int i = 0; i < n; i++) {
19            String num = ans[i];
20            firstPosition.putIfAbsent(num, i);
21            count.put(num, count.getOrDefault(num, 0) + 1);
22        }
23
24        // 去重后的数字
25        List<String> res = new ArrayList<>(firstPosition.keySet());
26
27        // 自定义排序: 按出现次数降序, 次数相同时按首次出现位置升序
28        res.sort((a, b) -> {
29            if (!count.get(a).equals(count.get(b))) {
30                return count.get(b) - count.get(a); // 按出现次数降序
31            }
32            return firstPosition.get(a) - firstPosition.get(b); // 首次出现
33            // 位置升序
34        });
35
36        // 输出结果
37        System.out.println(String.join(",", res));
38    }
}
```

Python

```
1 def main():
2     import sys
3     input_data = sys.stdin.read().strip()
4     if not input_data:
5         print("")
6         return
7
8     ans = input_data.split(",")
9
10    # 记录数字首次出现的位置
11    first_position = {}
12    # 统计数字出现的次数
13    count = {}
14
15    for i, num in enumerate(ans):
16        if num not in first_position:
17            first_position[num] = i
18        count[num] = count.get(num, 0) + 1
19
20    # 去重后的数字
21    res = list(first_position.keys())
22
23    # 自定义排序: 按出现次数降序, 次数相同时按首次出现位置升序
24    res.sort(key=lambda x: (-count[x], first_position[x]))
25
26    # 输出结果
27    print(",".join(res))
28
29
30 if __name__ == "__main__":
31     main()
```

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", (s) => {
9     if (!s.trim()) {
10         console.log("");
11         rl.close();
12         return;
13     }
14
15     let ans = s.split(",");
16
17     // 记录数字首次出现的位置
18     let firstPosition = new Map();
19     // 统计数字出现的次数
20     let count = new Map();
21
22     ans.forEach((num, i) => {
23         if (!firstPosition.has(num)) {
24             firstPosition.set(num, i);
25         }
26         count.set(num, (count.get(num) || 0) + 1);
27     });
28
29     // 去重后的数字
30     let res = Array.from(firstPosition.keys());
31
32     // 自定义排序: 按出现次数降序, 次数相同时按首次出现位置升序
33     res.sort((a, b) => {
34         if (count.get(a) !== count.get(b)) {
35             return count.get(b) - count.get(a); // 出现次数降序
36         }
37         return firstPosition.get(a) - firstPosition.get(b); // 首次出现位置
38         升序
39     });
40
41     console.log(res.join(","));
42     rl.close();
43 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strings"
9 )
10
11 func main() {
12     scanner := bufio.NewScanner(os.Stdin)
13     scanner.Scan()
14     s := scanner.Text()
15
16     if len(s) == 0 {
17         fmt.Println("")
18         return
19     }
20
21     // 直接使用 strings.Split 进行分割
22     ans := strings.Split(s, ",")
23
24     // 记录数字首次出现的位置
25     firstPosition := make(map[string]int)
26     // 统计数字出现的次数
27     count := make(map[string]int)
28
29     for i, num := range ans {
30         if _, exists := firstPosition[num]; !exists {
31             firstPosition[num] = i
32         }
33         count[num]++
34     }
35
36     // 去重后的数字
37     uniqueNumbers := make([]string, 0, len(firstPosition))
38     for num := range firstPosition {
39         uniqueNumbers = append(uniqueNumbers, num)
40     }
41
42     // 自定义排序: 按出现次数降序, 次数相同时按首次出现位置升序
43     sort.Slice(uniqueNumbers, func(i, j int) bool {
44         a, b := uniqueNumbers[i], uniqueNumbers[j]
45         if count[a] == count[b] {
```

```
46         return firstPosition[a] < firstPosition[b] // 首次出现位置升序
47     }
48     return count[a] > count[b] // 出现次数降序
49 }
50
51 // 输出结果
52 fmt.Println(strings.Join(uniqueNumbers, ","))
53 }
```

| 来自: 华为OD机试2025C卷 – 数组去重和排序 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 密码输入检测 (C++ & Python & JAVA & JS & GO)-CSDN博客

密码输入检测

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

给定用户密码输入流 `input`, 输入流中字符 ‘<’ 表示退格, 可以清除前一个输入的字符, 请你编写程序, 输出最终得到的密码字符, 并判断密码是否满足如下的密码安全要求。

密码安全要求如下:

1. 密码长度 ≥ 8 ;
2. 密码至少需要包含 1 个大写字母;
3. 密码至少需要包含 1 个小写字母;
4. 密码至少需要包含 1 个数字;
5. 密码至少需要包含 1 个字母和数字以外的非空白特殊字符;

注意空串退格后仍然为空串, 且 [用户输入](#) 的字符串不包含 ‘<’ 字符和空白字符。

输入描述

用一行字符串表示输入的用户数据, 输入的字符串中 ‘<’ 字符标识退格, 用户输入的字符串不包含空白字符, 例如:

ABC<c89%000<

输出描述

输出经过程序处理后, 输出的实际密码字符串, 并输出改密码字符串是否满足密码安全要求。两者间由 ‘,’ 分隔, 例如:

ABc89%00,true

用例1

输入

```
1 ABC<c89%000<
Plain Text
```

输出

```
Plain Text |  
1 ABc89%00,true
```

说明

多余的C和0由于退格被去除,最终用户输入的密码为ABc89%00, 且满足密码安全要求, 输出true

题解

思路: 模拟

1. 使用 数组 模拟栈模拟输入, 遇到 < 移除栈顶元素, 其它的直接压入栈。
2. 统计处理之后的密码中各种字符数量, 并判断是否符合合法密码规则
3. 输出处理之后密码及是否合法标志。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11 int main() {
12     string input;
13     getline(cin, input);
14     string res;
15
16     // 模拟输入
17     int n = input.size();
18     for (int i = 0; i < n; i++) {
19         char c = input[i];
20         if (c != '<') {
21             res.push_back(c);
22             continue;
23         }
24         if (res.empty()) {
25             continue;
26         }
27         res.pop_back();
28     }
29     // 统计
30     int upperCount, lowCount, numberCount, otherCount;
31     upperCount = lowCount = numberCount = otherCount = 0;
32
33     for (int i = 0; i < res.size(); i++) {
34         char c = res[i];
35         if ('a' <= c && c <= 'z') {
36             lowCount++;
37         } else if ('A' <= c && c <= 'Z') {
38             upperCount++;
39         } else if ('0' <= c && c <= '9') {
40             numberCount++;
41         } else {
42             otherCount++;
43         }
44     }
45     int totalCount = upperCount + lowCount + numberCount + otherCount;
```

```
46     string flag = "true";
47     // 判断是否合法
48     if (totalCount < 8 || upperCount < 1 || lowCount < 1 || numberCount <
49         1 || otherCount < 1) {
50         flag = "false";
51     }
52     cout << res << "," << flag;
53
54     return 0;
55 }
```

JAVA

```
1 import java.io.*;
2
3 public class Main {
4     public static void main(String[] args) throws IOException {
5         // 使用缓冲流读取输入
6         BufferedReader br = new BufferedReader(new InputStreamReader(Syste
m.in));
7         String input = br.readLine();
8
9         // 用 StringBuilder 模拟退格操作
10        StringBuilder res = new StringBuilder();
11
12        // 遍历字符串模拟输入
13        for (char c : input.toCharArray()) {
14            if (c != '<') {
15                // 普通字符, 加入结果
16                res.append(c);
17            } else {
18                // 碰到退格符 '<', 删除最后一个字符
19                if (res.length() > 0) {
20                    res.deleteCharAt(res.length() - 1);
21                }
22            }
23        }
24
25        // 统计不同类型字符个数
26        int upperCount = 0, lowCount = 0, numberCount = 0, otherCount = 0;
27        for (int i = 0; i < res.length(); i++) {
28            char c = res.charAt(i);
29            if (c >= 'a' && c <= 'z') {
30                lowCount++;
31            } else if (c >= 'A' && c <= 'Z') {
32                upperCount++;
33            } else if (c >= '0' && c <= '9') {
34                numberCount++;
35            } else {
36                otherCount++;
37            }
38        }
39
40        // 总长度
41        int totalCount = upperCount + lowCount + numberCount + otherCount;
42
43        // 判断是否满足密码合法性
44        String flag = "true";
```

```
45         if (totalCount < 8 || upperCount < 1 || lowCount < 1 || numberCoun  
46 t < 1 || otherCount < 1) {  
47     flag = "false";  
48 }  
49 // 输出结果  
50 System.out.println(res.toString() + "," + flag);  
51 }  
52 }
```

Python

```
1 import sys
2
3 # 读取一行输入
4 s = sys.stdin.readline().strip()
5
6 res = []
7
8 # 模拟退格处理
9 for c in s:
10     if c != '<':
11         # 普通字符加入结果
12         res.append(c)
13     else:
14         # 遇到退格符，删除最后一个字符（如果存在）
15         if res:
16             res.pop()
17
18 # 转为字符串
19 res_str = "".join(res)
20
21 # 统计不同类型的字符数量
22 upperCount = sum(1 for c in res if c.isupper())
23 lowCount = sum(1 for c in res if c.islower())
24 numberCount = sum(1 for c in res if c.isdigit())
25 otherCount = len(res) - upperCount - lowCount - numberCount
26
27 # 总长度
28 totalCount = upperCount + lowCount + numberCount + otherCount
29
30 # 判断是否合法
31 flag = "true"
32 if totalCount < 8 or upperCount < 1 or lowCount < 1 or numberCount < 1 or
otherCount < 1:
33     flag = "false"
34
35 # 输出结果
36 print(f"{res_str},{flag}")
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = "";
9
10 rl.on("line", (line) => {
11     input = line.trim();
12 }).on("close", () => {
13     let res = [];
14
15     // 模拟退格处理
16     for (let c of input) {
17         if (c !== "<") {
18             // 普通字符直接加入
19             res.push(c);
20         } else {
21             // 碰到退格符 '<', 删除最后一个字符
22             if (res.length > 0) res.pop();
23         }
24     }
25
26     let resStr = res.join("");
27
28     // 统计不同类型字符个数
29     let upperCount = 0, lowCount = 0, numberCount = 0, otherCount = 0;
30     for (let c of res) {
31         if (c >= "a" && c <= "z") lowCount++;
32         else if (c >= "A" && c <= "Z") upperCount++;
33         else if (c >= "0" && c <= "9") numberCount++;
34         else otherCount++;
35     }
36
37     let totalCount = upperCount + lowCount + numberCount + otherCount;
38
39     // 判断是否合法
40     let flag = "true";
41     if (totalCount < 8 || upperCount < 1 || lowCount < 1 || numberCount <
42     1 || otherCount < 1) {
43         flag = "false";
44     }
45 }
```

```
45      // 输出结果
46      console.log(resStr + "," + flag);
47  });

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 func main() {
11     // 使用缓冲读取输入
12     reader := bufio.NewReader(os.Stdin)
13     line, _ := reader.ReadString('\n')
14     line = strings.TrimSpace(line)
15
16     var res []rune
17
18     // 模拟退格处理
19     for _, c := range line {
20         if c != '<' {
21             // 普通字符加入结果
22             res = append(res, c)
23         } else {
24             // 碰到退格符 '<', 删除最后一个字符
25             if len(res) > 0 {
26                 res = res[:len(res)-1]
27             }
28         }
29     }
30
31     resStr := string(res)
32
33     // 统计不同类型字符数量
34     upperCount, lowCount, numberCount, otherCount := 0, 0, 0, 0
35     for _, c := range res {
36         switch {
37         case c >= 'a' && c <= 'z':
38             lowCount++
39         case c >= 'A' && c <= 'Z':
40             upperCount++
41         case c >= '0' && c <= '9':
42             numberCount++
43         default:
44             otherCount++
45     }
```

```
46     }
47
48     totalCount := upperCount + lowCount + numberCount + otherCount
49
50     // 判断是否合法
51     flag := "true"
52     if totalCount < 8 || upperCount < 1 || lowCount < 1 || numberCount < 1 ||
53     | otherCount < 1 {
54         flag = "false"
55     }
56
57     // 输出结果
58     fmt.Printf("%s,%s\n", resStr, flag)
59 }
```

| 来自: 华为OD机考2025C卷 – 密码输入检测 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 统计射击比赛成绩 (C++ & Python & JAVA & JS & GO)-CSDN博客

统计射击比赛成绩

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

给定一个射击比赛成绩单，包含多个选手若干次射击的成绩分数，请对每个选手按其最高3个分数之和进行降序排名，输出降序排名后的选手ID序列。

条件如下：

1. 一个选手可以有多个射击成绩的分数，且次序不固定。
2. 如果一个选手成绩少于3个，则认为选手的所有成绩无效，排名忽略该选手。
3. 如果选手的成绩之和相等，则成绩之和相等的选手按照其ID降序排列。

输入描述

- 输入第一行，一个整数N，表示该场比赛总共进行了N次射击，产生N个成绩分数 ($2 \leq N \leq 100$)。
- 输入第二行，一个长度为N整数序列，表示参与每次射击的选手ID ($0 \leq ID \leq 99$)。
- 输入第三行，一个长度为N整数序列，表示参与每次射击的选手对应的成绩 ($0 \leq 成绩 \leq 100$)。

输出描述

符合题设条件的降序排名后的选手ID序列。

用例1

输入

```
▼ Plain Text |  
1 13  
2 3,3,7,4,4,4,4,7,7,3,5,5,5  
3 53,80,68,24,39,76,66,16,100,55,53,80,55
```

输出

▼

Plain Text |

1 5,3,7,4

说明

▼

Plain Text |

1 该场射击比赛进行了13次
2
3 参赛的选手为3,4,5,7
4
5 3号选手成绩: 53,80,55, 最高3个成绩的和为: $80+55+53=188$ 。
6 4号选手成绩: 24,39,76,66, 最高3个成绩的和为: $76+66+39=181$ 。
7 5号选手成绩: 53,80,55, 最高3个成绩的和为: $80+55+53=188$ 。
8 7号选手成绩: 68,16,100, 最高3个成绩的和为: $100+68+16=184$ 。
9 比较各个选手最高3个成绩的和, 有3号=5号>7号>4号, 由于3号和5号成绩相等且ID号5>3, 所以输出为: 5,3,7,4

题解

思路: 哈希表 + 自定义排序

1. 使用 哈希表 分类存储各个选手所获得的成绩。
2. 忽略哈希表中成绩小于3个的选手, 成绩多余三个选手取最高三个分数作为他的总成绩。
3. 将第二步得到的(选手, 总成绩)自定义进行排序, 排序规则为 优先按总成绩降序排序, 总成绩相同按照id降序排序。
4. 输出排序后的选手顺序。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11
12 // 通用 split 函数
13 vector<string> split(const string& str, const string& delimiter) {
14     vector<string> result;
15     size_t start = 0;
16     size_t end = str.find(delimiter);
17     while (end != string::npos) {
18         result.push_back(str.substr(start, end - start));
19         start = end + delimiter.length();
20         end = str.find(delimiter, start);
21     }
22     // 添加最后一个部分
23     result.push_back(str.substr(start));
24     return result;
25 }
26
27 // 自定义排序规则
28 bool cmp(pair<int, int>& p1, pair<int,int>& p2) {
29     if (p1.first == p2.first) {
30         return p1.second > p2.second;
31     }
32     return p1.first > p2.first;
33 }
34
35 int main() {
36     int n;
37     map<int, vector<int>> idScoreMap;
38     cin >> n;
39     cin.ignore();
40     string idSequence,scoreSequence;
41     getline(cin, idSequence);
42     getline(cin, scoreSequence);
43
44     vector<string> ids = split(idSequence, ",");
45     vector<string> scores = split(scoreSequence, ",");
```

```
46
47 // 分类存储每个人的射击成绩
48 for (int i = 0; i < n; i++) {
49     idScoreMap[stoi(ids[i])].push_back(stoi(scores[i]));
50 }
51
52 vector<pair<int, int>> persons;
53
54 //统计出成绩数量 > 3 并且 计算最大三个成绩综合
55 for (auto p : idScoreMap) {
56     vector<int> scores = p.second;
57     int id = p.first;
58     if (scores.size() < 3) {
59         continue;
60     }
61     sort(scores.begin(), scores.end());
62     int m = scores.size();
63     int total = 0;
64     for (int i = m - 1; i >= m-3; i--) {
65         total += scores[i];
66     }
67     persons.push_back({total, id});
68 }
69
70 // 自定义排序
71 sort(persons.begin(), persons.end(), cmp);
72 int m = persons.size();
73 for (int i = 0; i < m; i++) {
74     cout << persons[i].second;
75     if (i != m -1) {
76         cout << ",";
77     }
78 }
79 return 0;
80 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 自定义排序: 按总成绩降序, 若相等按id升序
5     static class Person implements Comparable<Person> {
6         int id;
7         int total;
8
9         Person(int id, int total) {
10             this.id = id;
11             this.total = total;
12         }
13
14         public int compareTo(Person other) {
15             if (this.total != other.total) {
16                 return other.total - this.total; // 成绩降序
17             }
18             return other.id - this.id; // id降序
19         }
20     }
21
22     public static void main(String[] args) {
23         Scanner sc = new Scanner(System.in);
24         int n = Integer.parseInt(sc.nextLine());
25         String[] idArr = sc.nextLine().split(",");
26         String[] scoreArr = sc.nextLine().split(",");
27
28         Map<Integer, List<Integer>> idScoreMap = new HashMap<>();
29
30         for (int i = 0; i < n; i++) {
31             int id = Integer.parseInt(idArr[i]);
32             int score = Integer.parseInt(scoreArr[i]);
33             idScoreMap.computeIfAbsent(id, k -> new ArrayList<>()).add(score);
34         }
35         // 使用数组替代类也可以
36         List<Person> persons = new ArrayList<>();
37
38         for (Map.Entry<Integer, List<Integer>> entry : idScoreMap.entrySet()
39             ()) {
40             List<Integer> scores = entry.getValue();
41             if (scores.size() < 3) continue;
42             Collections.sort(scores, Collections.reverseOrder());
43             int total = scores.get(0) + scores.get(1) + scores.get(2);
44             persons.add(new Person(entry.getKey(), total));
45         }
46     }
47 }
```

```
44     }
45
46     Collections.sort(persons);
47
48     for (int i = 0; i < persons.size(); i++) {
49         System.out.print(persons.get(i).id);
50         if (i != persons.size() - 1) System.out.print(",");
51     }
52 }
53 }
```

Python

Plain Text |

```
1 n = int(input())
2 ids = input().split(',')
3 scores = input().split(',')
4
5 from collections import defaultdict
6
7 id_score_map = defaultdict(list)
8
9 # 分类存储每个人的成绩
10 for i in range(n):
11     id_score_map[int(ids[i])].append(int(scores[i]))
12
13 persons = []
14
15 # 统计三个最高分数总和
16 for id_, score_list in id_score_map.items():
17     if len(score_list) < 3:
18         continue
19     score_list.sort(reverse=True)
20     total = sum(score_list[:3])
21     persons.append((total, id_))
22
23 # 按照总分降序, 如果相同则按id降序
24 persons.sort(key=lambda x: (-x[0], -x[1]))
25
26 print(",".join(str(p[1]) for p in persons))
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout,
6 });
7
8 let inputLines = [];
9 rl.on("line", (line) => {
10     inputLines.push(line);
11     if (inputLines.length === 3) {
12         const n = parseInt(inputLines[0]);
13         const ids = inputLines[1].split(',');
14         const scores = inputLines[2].split(',');
15
16         const map = new Map();
17         // 分类存储每个人的成绩
18         for (let i = 0; i < n; i++) {
19             const id = parseInt(ids[i]);
20             const score = parseInt(scores[i]);
21             if (!map.has(id)) map.set(id, []);
22             map.get(id).push(score);
23         }
24
25         const persons = [];
26         // 统计三个最高分数总和
27         for (const [id, scoreList] of map.entries()) {
28             if (scoreList.length < 3) continue;
29             scoreList.sort((a, b) => b - a);
30             const total = scoreList[0] + scoreList[1] + scoreList[2];
31             persons.push([total, id]);
32         }
33         // 按照总分降序, 如果相同则按id降序
34         persons.sort((a, b) => {
35             if (a[0] !== b[0]) return b[0] - a[0];
36             return b[1] - a[1];
37         });
38
39         console.log(persons.map(p => p[1]).join(","));
40         rl.close();
41     }
42 });
```

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10    )
11
12 // 定义一个结构体
13 type Person struct {
14     id      int
15     total   int
16 }
17
18 func main() {
19     scanner := bufio.NewScanner(os.Stdin)
20     scanner.Scan()
21     n, _ := strconv.Atoi(scanner.Text())
22     scanner.Scan()
23     idStr := strings.Split(scanner.Text(), ",")
24     scanner.Scan()
25     scoreStr := strings.Split(scanner.Text(), ",")
26
27     idScoreMap := make(map[int][]int)
28     // 分类存储每个人的成绩
29     for i := 0; i < n; i++ {
30         id, _ := strconv.Atoi(idStr[i])
31         score, _ := strconv.Atoi(scoreStr[i])
32         idScoreMap[id] = append(idScoreMap[id], score)
33     }
34
35     var persons []Person
36     // 统计三个最高分数总和
37     for id, scores := range idScoreMap {
38         if len(scores) < 3 {
39             continue
40         }
41         sort.Sort(sort.Reverse(sort.IntSlice(scores)))
42         total := scores[0] + scores[1] + scores[2]
43         persons = append(persons, Person{id, total})
44     }
45     // 按照总分降序, 如果相同则按id降序
```

```
46     sort.Slice(persons, func(i, j int) bool {
47         if persons[i].total == persons[j].total {
48             return persons[i].id > persons[j].id
49         }
50         return persons[i].total > persons[j].total
51     })
52
53     for i, p := range persons {
54         if i != 0 {
55             fmt.Print(",")
56         }
57         fmt.Println(p.id)
58     }
59 }
```

| 来自: 华为OD机考2025C卷 – 统计射击比赛成绩 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 字符统计及重排 (C++ & Python & JAVA & JS & GO)-CSDN博客

字符统计及重排

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

给出一个仅包含字母的字符串，不包含空格，统计字符串中各个字母（区分大小写）出现的次数，并按照字母出现次数从大到小的顺序。输出各个字母及其出现次数。

如果次数相同，按照自然顺序进行排序，且小写字母在大写字母之前。

输入描述

输入一行，为一个仅包含字母的字符串。

输出描述

按照字母出现次数从大到小的顺序输出各个字母和字母次数，用英文分号分隔，注意末尾的分号；字母和次数间用英文冒号分隔。

示例1

输入

```
1 xyxyXX
```

输出

```
1 x:2;y:2;X:2;
```

示例2

输入

Plain Text |

1 abababb

输出

Plain Text |

1 b:4;a:3;

说明

b的出现个数比a多，故b排在a之前

题解

思路： 哈希数据结构 + 自定义排序

1. 使用哈希表统计字符串每个字母出现的次数。
2. 按照题目要求进行自定义排序，排序规则 先按照出现次数降序排序，出现次数相同按照自然顺序进行排序，且小写字母在大写字母之前 .
3. 按题目要求格式输出排序之后的结果。

C++

```
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int main() {
5     string str;
6     getline(cin, str); // 读入字符串
7
8     unordered_map<char, int> count; // 统计字符出现次数
9     for (char ch : str) {
10         count[ch]++;
11     }
12
13     vector<pair<char, int>> charFreq(count.begin(), count.end());
14
15     // 排序规则:
16     // 1. 按出现次数降序
17     // 2. 若次数相同, 按自然顺序 (小写在前, 大写在后)
18     sort(charFreq.begin(), charFreq.end(), [] (const pair<char, int>& a, const pair<char, int>& b) {
19         if (a.second == b.second) {
20             // 小写字母优先, 因此大写字母 ASCII 码小, 需要反向排序
21             if (islower(a.first) && isupper(b.first)) return true;
22             if (isupper(a.first) && islower(b.first)) return false;
23             return a.first < b.first; // 其他情况按字母顺序
24         }
25         return a.second > b.second; // 先按次数降序
26     });
27
28     // 构造结果字符串
29     string result;
30     for (const auto& [ch, freq] : charFreq) {
31         result += ch + string(":") + to_string(freq) + ";";
32     }
33
34     // 输出结果
35     cout << result << endl;
36     return 0;
37 }
```

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         String str = scanner.nextLine(); // 读入字符串
7
8         Map<Character, Integer> count = new HashMap<>(); // 统计字符出现次数
9         for (char ch : str.toCharArray()) {
10             count.put(ch, count.getOrDefault(ch, 0) + 1);
11         }
12
13         List<Map.Entry<Character, Integer>> charFreq = new ArrayList<>(count.entrySet());
14
15         // 排序规则:
16         // 1. 按出现次数降序
17         // 2. 若次数相同, 按自然顺序 (小写在前, 大写在后)
18         charFreq.sort((a, b) -> {
19             if (a.getValue().equals(b.getValue())) {
20                 // 小写字母优先, 因此大写字母 ASCII 码小, 需要反向排序
21                 if (Character.isLowerCase(a.getKey()) && Character.isUpperCase(b.getKey())) return -1;
22                 if (Character.isUpperCase(a.getKey()) && Character.isLowerCase(b.getKey())) return 1;
23                 return a.getKey() - b.getKey(); // 其他情况按字母顺序
24             }
25             return b.getValue() - a.getValue(); // 先按次数降序
26         });
27
28         // 构造结果字符串
29         StringBuilder result = new StringBuilder();
30         for (Map.Entry<Character, Integer> entry : charFreq) {
31             result.append(entry.getKey()).append(":").append(entry.getValue()).append(";");
32         }
33
34         // 输出结果
35         System.out.println(result.toString());
36     }
37 }
```

Python

```
1 from collections import Counter
2
3 # 读入字符串
4 str_input = input()
5
6 # 统计字符出现次数
7 count = Counter(str_input)
8
9 # 排序规则:
10 # 1. 按出现次数降序
11 # 2. 若次数相同, 按自然顺序 (小写在前, 大写在后)
12 sorted_freq = sorted(count.items(), key=lambda x: (-x[1], x[0].isupper(), x[0]))
13
14 # 构造结果字符串
15 result = ""
16 for ch, freq in sorted_freq:
17     result += f"{ch}:{freq};"
18
19 # 输出结果
20 print(result)
```

JavaScript

```
1 // 引入 readline 模块
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 // 读入字符串
11 rl.question('', (str) => {
12   // 统计字符出现次数
13   const count = {};
14   for (let ch of str) {
15     count[ch] = (count[ch] || 0) + 1;
16   }
17
18   // 将字典转换为数组，进行排序
19   const charFreq = Object.entries(count);
20
21   // 排序规则：
22   // 1. 按出现次数降序
23   // 2. 若次数相同，两个字母排序规则（小写在前，大写在后，两个字母都为大写或者小写保持ascii升序）
24   charFreq.sort(([charA, freqA], [charB, freqB]) => {
25     if (freqA === freqB) {
26       const isALower = charA >= 'a' && charA <= 'z'; // 使用字符本身来判断
是否为小写
27       const isBLower = charB >= 'a' && charB <= 'z';
28
29       if (isALower === isBLower) {
30         return charA.localeCompare(charB); // 使用 localeCompare 来处理
字符的顺序
31       } else {
32         return isALower ? -1 : 1;
33       }
34     }
35     return freqB - freqA; // 先按次数降序
36   });
37
38 // 构造结果字符串
39 let result = '';
40 for (const [ch, freq] of charFreq) {
41   result += `${ch}:${freq};`;
42 }
```

```
43
44    // 输出结果
45    console.log(result);
46
47    // 关闭 readline 接口
48    rl.close();
49});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "sort"
7     "strings"
8     "unicode"
9     "os"
10    )
11
12 func main() {
13     // 读入字符串
14     reader := bufio.NewReader(os.Stdin)
15     str, _ := reader.ReadString('\n')
16     str = strings.TrimSpace(str) // 去掉末尾的换行符
17
18     // 统计字符出现次数
19     count := make(map[rune]int)
20     for _, ch := range str {
21         count[ch]++
22     }
23
24     // 将字典转换为切片，进行排序
25     var charFreq []struct {
26         char rune
27         freq int
28     }
29     for ch, freq := range count {
30         charFreq = append(charFreq, struct {
31             char rune
32             freq int
33         }{ch, freq})
34     }
35
36     // 排序规则：
37     // 1. 按出现次数降序
38     // 2. 若次数相同，按自然顺序（小写在前，大写在后）
39     sort.Slice(charFreq, func(i, j int) bool {
40         if charFreq[i].freq == charFreq[j].freq {
41             // 小写字母优先，因此大写字母 ASCII 码小，需要反向排序
42             if unicode.IsLower(charFreq[i].char) && unicode.IsUpper(charFreq[j].char) {
43                 return true
44             }
45         }
46     })
47 }
```

```
45         if unicode.ToUpper(charFreq[i].char) && unicode.ToLower(charFreq[j].char) {
46             return false
47         }
48         return charFreq[i].char < charFreq[j].char // 其他情况按字母顺序
49     }
50     return charFreq[i].freq > charFreq[j].freq // 先按次数降序
51 }
52
53 // 构造结果字符串
54 var result strings.Builder
55 for _, entry := range charFreq {
56     result.WriteString(fmt.Sprintf("%c:%d;", entry.char, entry.freq))
57 }
58
59 // 输出结果
60 fmt.Println(result.String())
61 }
```

来自: 华为OD机考2025C卷 – 字符统计及重排 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 矩阵扩散 (C++ & Python & JAVA & JS & GO)-CSDN博客

矩阵扩散

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

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或1。

其中值为1的成员具备扩散性，每经过1S，将上下左右值为0的成员同化为1。

二维数组的成员初始值都为0，将第 $[i,j]$ 和 $[k,l]$ 两个位置上元素修改成1后，求矩阵的所有元素变为1需要多长时间。

输入描述

输入数据中的前2个数字表示这是一个 $m \times n$ 的矩阵， m 和 n 不会超过1024大小；

中间两个数字表示一个初始扩散点位置为 i,j ；

最后2个数字表示另一个扩散点位置为 k,l 。

输出描述

输出矩阵的所有元素变为1所需要秒数。

用例1

输入

```
1 4,4,0,0,3,3
```

输出

```
1 2
```

说明

- 1 输入数据中的前2个数字表示这是一个 4×4 的矩阵；中间两个数字表示一个初始扩散点位置为 $0, 0$ ；最后2个数字表示另一个扩散点位置为 $3, 3$ 。给出的样例是一个简单模型，初始点在对角线上，达到中间的位置分别为3次迭代，即3秒。所以输出为3。

题解

思路： 多源BFS 算法

1. 使用 队列 模拟BFS算法，初始将两个位置加入队列并将 `matrix[i][j] = maxtrix[k][l]=1`。
2. 执行循环迭代扩散，每一轮将上一轮坐标的所有点往四周扩散，将新同化的点添加到一个新队列中用于下一轮。循环迭代退出条件为 没有可扩散点使用
3. 结果就是扩散的轮数

C++

```
1 #include <algorithm>
2 #include <iostream>
3 #include <queue>
4 #include <sstream>
5 #include <string>
6 #include <utility>
7 #include <vector>
8 using namespace std;
9
10 // 通用 split 函数
11 vector<int> split(const string &str, const string &delimiter) {
12     vector<int> result;
13     size_t start = 0;
14     size_t end = str.find(delimiter);
15     while (end != string::npos) {
16         result.push_back(stoi(str.substr(start, end - start)));
17         start = end + delimiter.length();
18         end = str.find(delimiter, start);
19     }
20     // 添加最后一个部分
21     result.push_back(stoi(str.substr(start)));
22     return result;
23 }
24
25 int main() {
26     string s;
27     cin >> s;
28     // 处理输入
29     vector<int> ans = split(s, ",");
30     int m = ans[0];
31     int n = ans[1];
32     int x1 = ans[2];
33     int y1 = ans[3];
34     int x2 = ans[4];
35     int y2 = ans[5];
36     vector<vector<int>> matrix(m, vector<int>(n, 0));
37     // 初始点越界
38     if (x1 < 0 || x1 >= m || y1 < 0 || y1 >= n) {
39         cout << -1;
40         return 0;
41     }
42     // 初始点越界
43     if (x2 < 0 || x2 >= m || y2 < 0 || y2 >= n) {
44         cout << -1;
45         return 0;
```

```

46    }
47
48    matrix[x1][y1] = 1;
49    matrix[x2][y2] = 1;
50    // 定义扩散方向
51    int direct[2][4] = {{1, 0, -1, 0}, {0, 1, 0, -1}};
52    queue<pair<int, int>> q;
53    q.push({x1, y1});
54    q.push({x2, y2});
55    int time = 0;
56    // bfs 使用队列模拟
57    while (!q.empty()) {
58        queue<pair<int, int>> tmp;
59        while (!q.empty()) {
60            pair<int, int> p = q.front();
61            q.pop();
62            int x = p.first;
63            int y = p.second;
64            for (int i = 0; i < 4; i++) {
65                int nx = x + direct[0][i];
66                int ny = y + direct[1][i];
67                // 边界检查
68                if (nx < 0 || nx >= m || ny < 0 || ny >= n) {
69                    continue;
70                }
71                // 已经被同化
72                if (matrix[nx][ny] == 1) {
73                    continue;
74                }
75                matrix[nx][ny] = 1;
76                tmp.push({nx, ny});
77            }
78        }
79        time++;
80        q = tmp;
81    }
82    cout << time - 1;
83    return 0;
84}

```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // BFS 扩散
5     public static void main(String[] args) {
6         Scanner in = new Scanner(System.in);
7         String s = in.next(); // 输入如: "5,5,0,0,4,4"
8         String[] parts = s.split(",");
9         int m = Integer.parseInt(parts[0]);
10        int n = Integer.parseInt(parts[1]);
11        int x1 = Integer.parseInt(parts[2]);
12        int y1 = Integer.parseInt(parts[3]);
13        int x2 = Integer.parseInt(parts[4]);
14        int y2 = Integer.parseInt(parts[5]);
15
16        // 检查初始点是否越界
17        if (x1 < 0 || x1 >= m || y1 < 0 || y1 >= n ||
18            x2 < 0 || x2 >= m || y2 < 0 || y2 >= n) {
19            System.out.println(-1);
20            return;
21        }
22
23        int[][] matrix = new int[m][n];
24        matrix[x1][y1] = 1;
25        matrix[x2][y2] = 1;
26
27        // 定义扩散方向
28        int[] dx = {1, 0, -1, 0};
29        int[] dy = {0, 1, 0, -1};
30
31        Queue<int[]> q = new LinkedList<>();
32        q.add(new int[]{x1, y1});
33        q.add(new int[]{x2, y2});
34        int time = 0;
35
36        // BFS 扩散
37        while (!q.isEmpty()) {
38            Queue<int[]> tmp = new LinkedList<>();
39            while (!q.isEmpty()) {
40                int[] p = q.poll();
41                int x = p[0], y = p[1];
42                for (int i = 0; i < 4; i++) {
43                    int nx = x + dx[i];
44                    int ny = y + dy[i];
45                    // 边界检查
```

```
46         if (nx < 0 || nx >= m || ny < 0 || ny >= n) continue;
47         // 已经被同化
48         if (matrix[nx][ny] == 1) continue;
49         matrix[nx][ny] = 1;
50         tmp.add(new int[]{nx, ny});
51     }
52 }
53 time++;
54 q = tmp;
55 }
56 System.out.println(time - 1);
57 }
58 }
```

Python

```
1  from collections import deque
2
3  def main():
4      s = input().strip() # 输入如 "5,5,0,0,4,4"
5      parts = list(map(int, s.split(",")))
6      m, n, x1, y1, x2, y2 = parts
7
8      # 初始点越界检查
9      if x1 < 0 or x1 >= m or y1 < 0 or y1 >= n or \
10         x2 < 0 or x2 >= m or y2 < 0 or y2 >= n:
11         print(-1)
12         return
13
14     matrix = [[0] * n for _ in range(m)]
15     matrix[x1][y1] = 1
16     matrix[x2][y2] = 1
17
18     # 定义扩散方向
19     directions = [(1,0), (0,1), (-1,0), (0,-1)]
20
21     q = deque([(x1, y1), (x2, y2)])
22     time = 0
23
24     # BFS 扩散
25     while q:
26         tmp = deque()
27         while q:
28             x, y = q.popleft()
29             for dx, dy in directions:
30                 nx, ny = x + dx, y + dy
31                 # 边界检查
32                 if nx < 0 or nx >= m or ny < 0 or ny >= n:
33                     continue
34                 if matrix[nx][ny] == 1:
35                     continue
36                 matrix[nx][ny] = 1
37                 tmp.append((nx, ny))
38             time += 1
39             q = tmp
40     print(time - 1)
41
42     if __name__ == "__main__":
43         main()
```

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", function (line) {
9     const parts = line.trim().split(",").map(Number);
10    const [m, n, x1, y1, x2, y2] = parts;
11
12    // 初始点越界检查
13    if (
14        x1 < 0 || x1 >= m || y1 < 0 || y1 >= n ||
15        x2 < 0 || x2 >= m || y2 < 0 || y2 >= n
16    ) {
17        console.log(-1);
18        rl.close();
19        return;
20    }
21
22    const matrix = Array.from({ length: m }, () => Array(n).fill(0));
23    matrix[x1][y1] = 1;
24    matrix[x2][y2] = 1;
25    // 扩散放心
26    const dx = [1, 0, -1, 0];
27    const dy = [0, 1, 0, -1];
28
29    let q = [[x1, y1], [x2, y2]];
30    let time = 0;
31
32    // BFS 扩散
33    while (q.length > 0) {
34        const tmp = [];
35        while (q.length > 0) {
36            const [x, y] = q.shift();
37            for (let i = 0; i < 4; i++) {
38                const nx = x + dx[i];
39                const ny = y + dy[i];
40                // 越界检查
41                if (nx < 0 || nx >= m || ny < 0 || ny >= n) continue;
42                // 已经被同化
43                if (matrix[nx][ny] === 1) continue;
44                matrix[nx][ny] = 1;
45                tmp.push([nx, ny]);
46            }
47        }
48        q = tmp;
49        time++;
50    }
51
52    console.log(time);
53}
```

```
46      }
47    }
48    time++;
49    q = tmp;
50  }
51  console.log(time - 1);
52  rl.close();
53});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     in := bufio.NewScanner(os.Stdin)
13     in.Scan()
14     line := in.Text()
15     parts := strings.Split(line, ",")
16     nums := make([]int, len(parts))
17     for i, p := range parts {
18         val, _ := strconv.Atoi(p)
19         nums[i] = val
20     }
21     m, n, x1, y1, x2, y2 := nums[0], nums[1], nums[2], nums[3], nums[4], nums[5]
22
23     // 初始点越界检查
24     if x1 < 0 || x1 >= m || y1 < 0 || y1 >= n ||
25         x2 < 0 || x2 >= m || y2 < 0 || y2 >= n {
26         fmt.Println(-1)
27         return
28     }
29
30     matrix := make([][]int, m)
31     for i := range matrix {
32         matrix[i] = make([]int, n)
33     }
34     matrix[x1][y1] = 1
35     matrix[x2][y2] = 1
36     // 定义扩散方向
37     dx := []int{1, 0, -1, 0}
38     dy := []int{0, 1, 0, -1}
39
40     q := [][]2]int{{x1, y1}, {x2, y2}}
41     time := 0
42
43     // BFS 扩散
44     for len(q) > 0 {
```

```
45     var tmp [][]int
46     for len(q) > 0 {
47         p := q[0]
48         q = q[1:]
49         x, y := p[0], p[1]
50         for i := 0; i < 4; i++ {
51             nx, ny := x+dx[i], y+dy[i]
52                 // 越界
53             if nx < 0 || nx >= m || ny < 0 || ny >= n {
54                 continue
55             }
56                 // 已经被同化
57             if matrix[nx][ny] == 1 {
58                 continue
59             }
60             matrix[nx][ny] = 1
61             tmp = append(tmp, [2]int{nx, ny})
62         }
63     }
64     time++
65     q = tmp
66 }
67 fmt.Println(time - 1)
68 }
```

| 来自: 华为OD机试2025C卷 – 矩阵扩散 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - CPU算力分配 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD2025机试C卷 100分题型

题目描述

现有两组服务器A和B，每组有多个算力不同的CPU，其中 $A[i]$ 是 A 组第 i 个CPU的运算能力， $B[i]$ 是 B 组 第 i 个CPU的运算能力。

一组服务器的总算力是各CPU的算力之和。

为了让两组服务器的算力相等，允许从每组各选出一个CPU进行一次交换，

求两组服务器中，用于交换的CPU的算力，并且要求从A组服务器中选出的CPU，算力尽可能小。

输入描述

第一行输入为 L_1 和 L_2 ，以空格分隔， L_1 表示A组服务器中的CPU数量， L_2 表示B组服务器中的CPU数量。

第二行输入为A组服务器中各个CPU的算力值，以空格分隔。

第三行输入为B组服务器中各个CPU的算力值，以空格分隔。

- $1 \leq L_1 \leq 10000$
- $1 \leq L_2 \leq 10000$
- $1 \leq A[i] \leq 100000$
- $1 \leq B[i] \leq 100000$

输出描述

对于每组测试数据，输出两个整数，以空格分隔，依次表示A组选出的CPU算力，B组选出的CPU算力。

要求从A组选出的CPU的算力尽可能小。

备注

- 保证两组服务器的初始总算力不同。
- 答案肯定存在

用例1

输入

▼

Plain Text |

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

输出

▼

Plain Text |

```
1 1 2
```

说明

从A组中选出算力为1的CPU，与B组中算力为2的进行交换，使两组服务器的算力都等于3。

用例2

输入

▼

Plain Text |

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

输出

▼

Plain Text |

```
1 1 2
```

用例3

输入

▼

Plain Text |

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

输出

```
Plain Text |  
1 2 3
```

用例4

输入

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

输出

```
Plain Text |  
1 5 4
```

题解

思路： 模拟问题

1. 分别计算服务器A和服务器B的算力和使用 `sumA` 和 `sumB` 存储。初始算力不一致并且只能交换依次，要想两个服务器交换之后算力和相同。那这次交换，服务器A需要获得 `targetDiff = (sumB - sumA)/2` 的额外算力。
2. 由于题目尽量从A中选出较小的CPU进行交换，可以将服务器A的CPU进行升序排序，从前到后进行判断 `cpuA[i] + targetDiff` 是否存在？存在直接输出结果即可。判断过程可以使用 `集合` 进行加速。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 #include<set>
10 using namespace std;
11
12 int main() {
13     int l1, l2;
14     cin >> l1 >> l2;
15     vector<int> cpuA(l1);
16     vector<int> cpuB(l2);
17     // 计算初始两个服务器算力和
18     int sumA,sumB;
19     sumA = sumB = 0;
20     for (int i = 0; i < l1; i++) {
21         cin >> cpuA[i];
22         sumA += cpuA[i];
23     }
24     // 用于快速判断指定大小cpuB服务器中是否存在
25     set<int> numberSet;
26     for (int i = 0; i < l2; i++) {
27         cin >> cpuB[i];
28         sumB += cpuB[i];
29         numberSet.insert(cpuB[i]);
30     }
31     // 交换的两个值差值
32     int targetDiff = (sumB - sumA) / 2;
33
34     sort(cpuA.begin(), cpuA.end());
35
36     for (int i = 0; i < l1; i++) {
37         int targetCpu = cpuA[i] + targetDiff;
38         if (numberSet.count(targetCpu)) {
39             cout << cpuA[i] << " " << targetCpu;
40             return 0;
41         }
42     }
43
44     return 0;
45 }
```

JAVA

```
Plain Text |  
1 import java.util.*;  
2  
3 public class Main {  
4     public static void main(String[] args) {  
5         Scanner scanner = new Scanner(System.in);  
6  
7         int l1 = scanner.nextInt();  
8         int l2 = scanner.nextInt();  
9  
10        int[] cpuA = new int[l1];  
11        int[] cpuB = new int[l2];  
12        // 计算服务器算力和  
13        int sumA = 0, sumB = 0;  
14  
15        for (int i = 0; i < l1; i++) {  
16            cpuA[i] = scanner.nextInt();  
17            sumA += cpuA[i];  
18        }  
19        // 后于后续加速判断  
20        Set<Integer> numberSet = new HashSet<>();  
21        for (int i = 0; i < l2; i++) {  
22            cpuB[i] = scanner.nextInt();  
23            sumB += cpuB[i];  
24            numberSet.add(cpuB[i]);  
25        }  
26        // 需要获得算力增益(可能为负)  
27        int targetDiff = (sumB - sumA) / 2;  
28        Arrays.sort(cpuA);  
29  
30        for (int i = 0; i < l1; i++) {  
31            int targetCpu = cpuA[i] + targetDiff;  
32            if (numberSet.contains(targetCpu)) {  
33                System.out.println(cpuA[i] + " " + targetCpu);  
34                return;  
35            }  
36        }  
37    }  
38}
```

Python

```
1 # 标准输入处理
2 l1, l2 = map(int, input().split())
3 cpuA = list(map(int, input().split()))
4 cpuB = list(map(int, input().split()))
5
6 sumA = sum(cpuA)
7 sumB = sum(cpuB)
8
9 # 用 set 快速判断目标数是否存在于 B 中
10 numberSet = set(cpuB)
11
12 # 交换目标差值
13 targetDiff = (sumB - sumA) // 2
14
15 cpuA.sort()
16
17 for a in cpuA:
18     targetCpu = a + targetDiff
19     if targetCpu in numberSet:
20         print(a, targetCpu)
21         break
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on('line', (line) => {
11     inputLines.push(line.trim());
12     if (inputLines.length === 3) {
13         rl.close();
14     }
15 });
16
17 rl.on('close', () => {
18     const [l1, l2] = inputLines[0].split(' ').map(Number);
19     const cpuA = inputLines[1].split(' ').map(Number);
20     const cpuB = inputLines[2].split(' ').map(Number);
21     // 计算初始服务器算力和
22     let sumA = cpuA.reduce((a, b) => a + b, 0);
23     let sumB = cpuB.reduce((a, b) => a + b, 0);
24     // 用于加速后续判断
25     const numberSet = new Set(cpuB);
26     // # 交换目标差值
27     const targetDiff = Math.floor((sumB - sumA) / 2);
28     cpuA.sort((a, b) => a - b);
29
30     for (let i = 0; i < cpuA.length; i++) {
31         let targetCpu = cpuA[i] + targetDiff;
32         if (numberSet.has(targetCpu)) {
33             console.log(` ${cpuA[i]} ${targetCpu}`);
34             return;
35         }
36     }
37});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9     "sort"
10    )
11
12 func main() {
13     scanner := bufio.NewScanner(os.Stdin)
14     scanner.Scan()
15     parts := strings.Fields(scanner.Text())
16     l1, _ := strconv.Atoi(parts[0])
17     l2, _ := strconv.Atoi(parts[1])
18
19     // 读取 cpuA
20     scanner.Scan()
21     parts = strings.Fields(scanner.Text())
22     cpuA := make([]int, l1)
23     sumA := 0
24     for i := 0; i < l1; i++ {
25         cpuA[i], _ = strconv.Atoi(parts[i])
26         sumA += cpuA[i]
27     }
28
29     // 读取 cpuB
30     scanner.Scan()
31     parts = strings.Fields(scanner.Text())
32     cpuB := make([]int, l2)
33     sumB := 0
34     // 用于加速后续判断
35     numberSet := make(map[int]bool)
36     for i := 0; i < l2; i++ {
37         cpuB[i], _ = strconv.Atoi(parts[i])
38         sumB += cpuB[i]
39         numberSet[cpuB[i]] = true
40     }
41
42     // 计算差值
43     targetDiff := (sumB - sumA) / 2
44     sort.Ints(cpuA)
45 }
```

```
46     for _, a := range cpuA {
47         targetCpu := a + targetDiff
48         if numberSet[targetCpu] {
49             fmt.Println(a, targetCpu)
50             return
51         }
52     }
53 }
```

| 来自: 华为OD机试2025C卷 – CPU算力分配 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 滑动窗口的最大值 (C++ & Python & JAVA & JS & GO)-CSDN博客

滑动窗口的最大值

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

有一个N个整数的数组，和一个长度为M的窗口，窗口从数组内的第一个数开始滑动直到窗口不能滑动为止，

每次[窗口滑动](#)产生一个窗口和（窗口内所有数的和），求窗口滑动产生的所有窗口和的最大值。

输入描述

- 第一行输入一个正整数N，表示整数个数。（ $0 < N < 100000$ ）
- 第二行输入N个整数，整数的取值范围为 $[-100, 100]$ 。
- 第三行输入一个正整数M，M代表窗口的大小， $M \leq 100000$ ，且 $M \leq N$ 。

输出描述

窗口滑动产生所有窗口和的最大值。

用例1

输入

```
1 6  
2 10 20 30 15 23 12  
3 3
```

输出

```
1 68
```

说明

窗口长度为3，窗口滑动产生的窗口和分别为

$10+20+30=60$,

$20+30+15=65$,

$30+15+23=68$,

$15+23+12=50$,

所以窗口滑动产生的所有窗口和的最大值为68。

题解

思路： 滑动窗口 算法应用题。

- 定义 `left = 0, right = 0` 分别表示 左边界作为，右边界下一个移动的位置， 定义 `tmpSum` 记录窗口中数字的和。
- 初始先让右边界一直右移，直到 `right - left + 1 == m`。
- 之后窗口 `left++ right++` 让窗口整体右移，直到 `right >= n` 结束。
- 输出其中出现最大窗口和

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<set>
11 using namespace std;
12
13 int main() {
14     int n, m;
15     cin >> n;
16     vector<int> ans(n);
17     for (int i = 0; i < n; i++) {
18         cin >> ans[i];
19     }
20     cin >> m;
21     // left 左边界 right右边界下一个选择的位置
22     int left = 0, right = 0;
23     int res = -100000000;
24     int tmp = 0;
25
26     while (right < n) {
27         // 只考虑窗口为M的情况
28         if (right - left + 1 < m) {
29             tmp += ans[right];
30             right++;
31             continue;
32         }
33         // 窗口大小为M+1, left 右移
34         if (right - left + 1 > m) {
35             tmp -= ans[left];
36             left++;
37         }
38         tmp += ans[right];
39         res = max(res, tmp);
40         right++;
41     }
42
43     cout << res;
44     return 0;
45 }
```

JAVA

Plain Text |

```
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         int n = sc.nextInt(); // 输入数组长度
8         int[] ans = new int[n];
9         for (int i = 0; i < n; i++) {
10             ans[i] = sc.nextInt();
11         }
12
13         int m = sc.nextInt(); // 滑动窗口大小
14
15         int left = 0, right = 0;
16         int res = Integer.MIN_VALUE;
17         int tmp = 0;
18
19         while (right < n) {
20             // 窗口还不够大，继续加元素
21             if (right - left + 1 < m) {
22                 tmp += ans[right];
23                 right++;
24                 continue;
25             }
26
27             // 窗口太大了，缩小窗口
28             if (right - left + 1 > m) {
29                 tmp -= ans[left];
30                 left++;
31             }
32
33             tmp += ans[right];
34             res = Math.max(res, tmp);
35             right++;
36         }
37
38         System.out.println(res);
39     }
40 }
```

Python

Plain Text |

```
1 # -*- coding: utf-8 -*-
2 n = int(input())
3 ans = list(map(int, input().split()))
4 m = int(input())
5
6 left = 0
7 right = 0
8 res = -10**9
9 tmp = 0
10
11 while right < n:
12     # 窗口还没满, 继续加
13     if right - left + 1 < m:
14         tmp += ans[right]
15         right += 1
16         continue
17
18     # 窗口太大了, 缩小
19     if right - left + 1 > m:
20         tmp -= ans[left]
21         left += 1
22
23     tmp += ans[right]
24     res = max(res, tmp)
25     right += 1
26
27 print(res)
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on("line", function (line) {
11     inputLines.push(line.trim());
12     if (inputLines.length === 3) {
13         main();
14         rl.close();
15     }
16 });
17
18 function main() {
19     const n = parseInt(inputLines[0]);
20     const ans = inputLines[1].split(' ').map(Number);
21     const m = parseInt(inputLines[2]);
22
23     let left = 0, right = 0;
24     let res = -1e9;
25     let tmp = 0;
26
27     while (right < n) {
28         // 窗口不够大
29         if (right - left + 1 < m) {
30             tmp += ans[right];
31             right++;
32             continue;
33         }
34
35         // 窗口太大
36         if (right - left + 1 > m) {
37             tmp -= ans[left];
38             left++;
39         }
40
41         tmp += ans[right];
42         res = Math.max(res, tmp);
43         right++;
44     }
45 }
```

```
46     console.log(res);  
47 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13
14     // 读取 n
15     line, _ := reader.ReadString('\n')
16     n, _ := strconv.Atoi(strings.TrimSpace(line))
17
18     // 读取数组
19     line, _ = reader.ReadString('\n')
20     fields := strings.Fields(line)
21     ans := make([]int, n)
22     for i := 0; i < n; i++ {
23         ans[i], _ = strconv.Atoi(fields[i])
24     }
25
26     // 读取 m
27     line, _ = reader.ReadString('\n')
28     m, _ := strconv.Atoi(strings.TrimSpace(line))
29
30     left, right := 0, 0
31     res := -100000000
32     tmp := 0
33
34     for right < n {
35         if right-left+1 < m {
36             tmp += ans[right]
37             right++
38             continue
39         }
40
41         if right-left+1 > m {
42             tmp -= ans[left]
43             left++
44         }
45     }
```

```
46         tmp += ans[right]
47         if tmp > res {
48             res = tmp
49         }
50         right++
51     }
52
53     fmt.Println(res)
54 }
```

滑动窗口的最大值

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

有一个N个整数的数组，和一个长度为M的窗口，窗口从数组内的第一个数开始滑动直到窗口不能滑动为止，

每次窗口滑动产生一个窗口和（窗口内所有数的和），求窗口滑动产生的所有窗口和的最大值。

输入描述

- 第一行输入一个正整数N，表示整数个数。（ $0 < N < 100000$ ）
- 第二行输入N个整数，整数的取值范围为 $[-100, 100]$ 。
- 第三行输入一个正整数M，M代表窗口的大小， $M \leq 100000$ ，且 $M \leq N$ 。

输出描述

窗口滑动产生所有窗口和的最大值。

用例1

输入

```
1 6
2 10 20 30 15 23 12
3 3
```

输出

Plain Text

1 68

说明

窗口长度为3，窗口滑动产生的窗口和分别为

$$10+20+30=60,$$

$$20+30+15=65,$$

$$30+15+23=68,$$

$$15+23+12=50,$$

所以窗口滑动产生的所有窗口和的最大值为68。

题解

思路： 滑动窗口 算法应用题。

- 定义 `left = 0, right = 0` 分别表示 左边界作为，右边界下一个移动的位置， 定义 `tmpSum` 记录窗口中数字的和。
- 初始先让右边界一直右移，直到 `right - left + 1 == m`。
- 之后窗口 `left++ right++` 让窗口整体右移，直到 `right >= n` 结束。
- 输出其中出现最大窗口和

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<set>
11 using namespace std;
12
13 int main() {
14     int n, m;
15     cin >> n;
16     vector<int> ans(n);
17     for (int i = 0; i < n; i++) {
18         cin >> ans[i];
19     }
20     cin >> m;
21     // left 左边界 right右边界下一个选择的位置
22     int left = 0, right = 0;
23     int res = -100000000;
24     int tmp = 0;
25
26     while (right < n) {
27         // 只考虑窗口为M的情况
28         if (right - left + 1 < m) {
29             tmp += ans[right];
30             right++;
31             continue;
32         }
33         // 窗口大小为M+1, left 右移
34         if (right - left + 1 > m) {
35             tmp -= ans[left];
36             left++;
37         }
38         tmp += ans[right];
39         res = max(res, tmp);
40         right++;
41     }
42
43     cout << res;
44     return 0;
45 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6
7         int n = sc.nextInt(); // 输入数组长度
8         int[] ans = new int[n];
9         for (int i = 0; i < n; i++) {
10             ans[i] = sc.nextInt();
11         }
12
13         int m = sc.nextInt(); // 滑动窗口大小
14
15         int left = 0, right = 0;
16         int res = Integer.MIN_VALUE;
17         int tmp = 0;
18
19         while (right < n) {
20             // 窗口还不够大，继续加元素
21             if (right - left + 1 < m) {
22                 tmp += ans[right];
23                 right++;
24                 continue;
25             }
26
27             // 窗口太大了，缩小窗口
28             if (right - left + 1 > m) {
29                 tmp -= ans[left];
30                 left++;
31             }
32
33             tmp += ans[right];
34             res = Math.max(res, tmp);
35             right++;
36         }
37
38         System.out.println(res);
39     }
40 }
```

Plain Text |

Python

Plain Text |

```
1 # -*- coding: utf-8 -*-
2 n = int(input())
3 ans = list(map(int, input().split()))
4 m = int(input())
5
6 left = 0
7 right = 0
8 res = -10**9
9 tmp = 0
10
11 while right < n:
12     # 窗口还没满, 继续加
13     if right - left + 1 < m:
14         tmp += ans[right]
15         right += 1
16         continue
17
18     # 窗口太大了, 缩小
19     if right - left + 1 > m:
20         tmp -= ans[left]
21         left += 1
22
23     tmp += ans[right]
24     res = max(res, tmp)
25     right += 1
26
27 print(res)
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on("line", function (line) {
11     inputLines.push(line.trim());
12     if (inputLines.length === 3) {
13         main();
14         rl.close();
15     }
16 });
17
18 function main() {
19     const n = parseInt(inputLines[0]);
20     const ans = inputLines[1].split(' ').map(Number);
21     const m = parseInt(inputLines[2]);
22
23     let left = 0, right = 0;
24     let res = -1e9;
25     let tmp = 0;
26
27     while (right < n) {
28         // 窗口不够大
29         if (right - left + 1 < m) {
30             tmp += ans[right];
31             right++;
32             continue;
33         }
34
35         // 窗口太大
36         if (right - left + 1 > m) {
37             tmp -= ans[left];
38             left++;
39         }
40
41         tmp += ans[right];
42         res = Math.max(res, tmp);
43         right++;
44     }
45 }
```

```
46     console.log(res);  
47 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13
14     // 读取 n
15     line, _ := reader.ReadString('\n')
16     n, _ := strconv.Atoi(strings.TrimSpace(line))
17
18     // 读取数组
19     line, _ = reader.ReadString('\n')
20     fields := strings.Fields(line)
21     ans := make([]int, n)
22     for i := 0; i < n; i++ {
23         ans[i], _ = strconv.Atoi(fields[i])
24     }
25
26     // 读取 m
27     line, _ = reader.ReadString('\n')
28     m, _ := strconv.Atoi(strings.TrimSpace(line))
29
30     left, right := 0, 0
31     res := -100000000
32     tmp := 0
33
34     for right < n {
35         if right-left+1 < m {
36             tmp += ans[right]
37             right++
38             continue
39         }
40
41         if right-left+1 > m {
42             tmp -= ans[left]
43             left++
44         }
45     }
```

```
46         tmp += ans[right]
47         if tmp > res {
48             res = tmp
49         }
50         right++
51     }
52
53     fmt.Println(res)
54 }
```

| 来自: 华为OD机考2025C卷 – 滑动窗口的最大值 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考2025C卷 – 滑动窗口的最大值 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 最大括号深度 (C++ & Python & JAVA & JS & GO)-CSDN博客

最大括号深度

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

现有一字符串仅由 ‘(， ’)’， ‘{， ’}’， ’[， ’]’六种括号组成。

若字符串满足以下条件之一，则为无效字符串：

1. 任一类型的左右括号数量不相等；
2. 存在未按正确顺序（先左后右）闭合的括号。

输出括号的最大嵌套深度，若字符串无效则输出0。

0≤[字符串长度](#)≤100000

输入描述

一个只包括 ‘(， ’)’， ‘{， ’}’， ’[， ’]’的字符串

输出描述

一个整数，最大的括号深度

用例1

输入

```
Plain Text  
1  []
```

输出

```
Plain Text  
1  1
```

用例2

输入

```
▼ Plain Text |  
1 ([{}])
```

输出

```
▼ Plain Text |  
1 3
```

说明

| 有效字符串，最大嵌套深度为3

用例3

输入

```
▼ Plain Text |  
1 ()
```

输出

```
▼ Plain Text |  
1 0
```

说明

| 无效字符串，有两种类型的左右括号数量不相等

用例4

输入

```
▼ Plain Text |  
1 ([])
```

输出

```
1 0
```

说明

无效字符串，存在未按正确顺序闭合的括号

题解

思路： 栈 数据结构运用题。使用 栈 来求最大括号深度，处理逻辑如下：

1. 从前往后遍历输入字符串

- 如果遇到 [{ (之后，直接压入栈。 在输入字符串括号匹配无异常的情况下，最大括号深度其实就为遍历过程中栈的出现过的最大长度
- 如果遇到] }) ， 判断是否能正确匹配(栈是否为空、栈顶元素是否为对应左括号)，不能正确匹配说明为无效字符串，直接输出0，结束。否则 弹出栈顶元素。

2. 遍历完输入字符串之后，如果栈不为空，说明存在 任一类型的左右括号数量不相等 的情况，直接输出0。如果为空的情况，输出遍历完过程中记录的最大栈长度即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 #include<stack>
10 #include<map>
11 using namespace std;
12
13 int main() {
14     int res = 0;
15     string input;
16     getline(cin, input);
17     // 定义括号对应关系，减少后续进行if判断
18     map<char, char> mp = {{')', '('}, {'[', '['}, {'}', '{'}, {'{', '}'}};
19
20     // 栈
21     stack<char> st;
22     for (int i = 0; i < input.size(); i++) {
23         char c = input[i];
24
25         // 左括号压栈
26         if (c == '(' || c == '{' || c == '[') {
27             st.push(c);
28             res = max(res, (int)st.size());
29         } else {
23         // 无效字符串
31             if (st.empty()) {
32                 cout << 0;
33                 return 0;
34             }
35             char top = st.top();
36             // 无效字符串
37             if (mp[c] != top) {
38                 cout << 0;
39                 return 0;
40             }
41             st.pop();
42         }
43     }
44     // 左右不相同
45     if (!st.empty()) {
```

```
46         cout << 0;
47     return 0;
48 }
49 cout << res;
50 return 0;
51 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         String input = sc.nextLine();
7         int res = 0;
8
9         // 括号映射
10        Map<Character, Character> mp = new HashMap<>();
11        mp.put(')', '(');
12        mp.put(']', '[');
13        mp.put('}', '{');
14
15        Stack<Character> stack = new Stack<>();
16
17        for (int i = 0; i < input.length(); i++) {
18            char c = input.charAt(i);
19
20            // 左括号压栈
21            if (c == '(' || c == '[' || c == '{') {
22                stack.push(c);
23                res = Math.max(res, stack.size());
24            } else {
25                // 无效字符串: 栈空 或者 括号不匹配
26                if (stack.isEmpty() || stack.peek() != mp.get(c)) {
27                    System.out.println(0);
28                    return;
29                }
30                stack.pop();
31            }
32        }
33
34        // 栈未清空, 左右数量不等
35        if (!stack.isEmpty()) {
36            System.out.println(0);
37        } else {
38            System.out.println(res);
39        }
40    }
41 }
```

Python

```
1 def main():
2     import sys
3     input_line = sys.stdin.readline().strip()
4     stack = []
5     res = 0
6
7     # 括号映射
8     mp = {')': '(', ']': '[', '}': '{'}
9
10    for c in input_line:
11        # 左括号压栈
12        if c in '([{':
13            stack.append(c)
14            res = max(res, len(stack))
15        else:
16            # 栈空或不匹配
17            if not stack or stack[-1] != mp.get(c):
18                print(0)
19                return
20            stack.pop()
21
22    # 栈未清空, 左右数量不等
23    if stack:
24        print(0)
25    else:
26        print(res)
27
28 if __name__ == '__main__':
29     main()
```

JavaScript

```
1 const readline = require('readline');
2 const rl = readline.createInterface({ input: process.stdin });
3
4 rl.on('line', function(line) {
5     const input = line.trim();
6     const stack = [];
7     let res = 0;
8     // 括号映射
9     const mp = { ')': '(', ']': '[', '}': '{' };
10
11    for (let c of input) {
12        // // 左括号压栈
13        if (c === '(' || c === '[' || c === '{') {
14            stack.push(c);
15            res = Math.max(res, stack.length);
16        } else {
17            // 无效字符串: 栈空 或者 括号不匹配
18            if (stack.length === 0 || stack[stack.length - 1] !== mp[c]) {
19                console.log(0);
20                return;
21            }
22            stack.pop();
23        }
24    }
25    // 栈未清空, 左右数量不等
26    if (stack.length !== 0) {
27        console.log(0);
28    } else {
29        console.log(res);
30    }
31});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 func main() {
11     reader := bufio.NewReader(os.Stdin)
12     input, _ := reader.ReadString('\n')
13     input = strings.TrimSpace(input) // 自动去除换行和空白
14
15     stack := []rune{}
16     res := 0
17
18     // 括号匹配映射表
19     mp := map[rune]rune{')': '(', ']': '[', '}': '{'}
20
21     for _, c := range input {
22         if c == '(' || c == '[' || c == '{' {
23             stack = append(stack, c)
24             if len(stack) > res {
25                 res = len(stack) // 更新最大嵌套深度
26             }
27         } else {
28             // 无效字符串: 栈空 或者 括号不匹配
29             if len(stack) == 0 || stack[len(stack)-1] != mp[c] {
30                 fmt.Println(0)
31                 return
32             }
33             stack = stack[:len(stack)-1]
34         }
35     }
36     // 栈未清空, 左右数量不等
37     if len(stack) != 0 {
38         fmt.Println(0)
39     } else {
40         fmt.Println(res)
41     }
42 }
```

最大括号深度

华为OD机试2025C卷 100分题型

题目描述

现有一字符串仅由 ‘(‘, ’)’, ‘{‘, ’}’, ‘[‘, ’]’六种括号组成。

若字符串满足以下条件之一，则为无效字符串：

1. 任一类型的左右括号数量不相等；
2. 存在未按正确顺序（先左后右）闭合的括号。

输出括号的最大嵌套深度，若字符串无效则输出0。

0≤字符串长度≤100000

输入描述

一个只包括 ‘(‘, ’)’, ‘{‘, ’}’, ‘[‘, ’]’的字符串

输出描述

一个整数，最大的括号深度

用例1

输入

```
Plain Text  
1  []
```

输出

```
Plain Text  
1  1
```

用例2

输入

```
Plain Text  
1  ([]{{()}})
```

输出

▼

Plain Text |

1 3

说明

| 有效字符串，最大嵌套深度为3

用例3

输入

▼

Plain Text |

1 ()

输出

▼

Plain Text |

1 0

说明

| 无效字符串，有两种类型的左右括号数量不相等

用例4

输入

▼

Plain Text |

1 ([])

输出

▼

Plain Text |

1 0

说明

| 无效字符串，存在未按正确顺序闭合的括号

题解

思路： 栈 数据结构运用题。使用 栈 来求最大括号深度，处理逻辑如下：

1. 从前往后遍历输入字符串
 - 如果遇到 [{ (之后，直接压入栈。 在输入字符串括号匹配无异常的情况下，最大括号深度其实就为遍历过程中栈的出现过的最大长度
 - 如果遇到] }) ， 判断是否能正确匹配(栈是否为空、栈顶元素是否为对应左括号)，不能正确匹配说明为无效字符串，直接输出0，结束。否则 弹出栈顶元素。
2. 遍历完输入字符串之后，如果栈不为空，说明存在 任一类型的左右括号数量不相等 的情况，直接输出0。如果为空的情况，输出遍历完过程中记录的最大栈长度即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 #include<stack>
10 #include<map>
11 using namespace std;
12
13 int main() {
14     int res = 0;
15     string input;
16     getline(cin, input);
17     // 定义括号对应关系，减少后续进行if判断
18     map<char, char> mp = {{')', '('}, {'[', '['}, {'}', '{'}, {'{', '}'}};
19
20     // 栈
21     stack<char> st;
22     for (int i = 0; i < input.size(); i++) {
23         char c = input[i];
24
25         // 左括号压栈
26         if (c == '(' || c == '{' || c == '[') {
27             st.push(c);
28             res = max(res, (int)st.size());
29         } else {
23         // 无效字符串
31             if (st.empty()) {
32                 cout << 0;
33                 return 0;
34             }
35             char top = st.top();
36             // 无效字符串
37             if (mp[c] != top) {
38                 cout << 0;
39                 return 0;
40             }
41             st.pop();
42         }
43     }
44     // 左右不相同
45     if (!st.empty()) {
```

```
46         cout << 0;
47     return 0;
48 }
49 cout << res;
50 return 0;
51 }
```

JAVA

```

1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         String input = sc.nextLine();
7         int res = 0;
8
9         // 括号映射
10        Map<Character, Character> mp = new HashMap<>();
11        mp.put(')', '(');
12        mp.put(']', '[');
13        mp.put('}', '{');
14
15        Stack<Character> stack = new Stack<>();
16
17        for (int i = 0; i < input.length(); i++) {
18            char c = input.charAt(i);
19
20            // 左括号压栈
21            if (c == '(' || c == '[' || c == '{') {
22                stack.push(c);
23                res = Math.max(res, stack.size());
24            } else {
25                // 无效字符串: 栈空 或者 括号不匹配
26                if (stack.isEmpty() || stack.peek() != mp.get(c)) {
27                    System.out.println(0);
28                    return;
29                }
30                stack.pop();
31            }
32        }
33
34        // 栈未清空, 左右数量不等
35        if (!stack.isEmpty()) {
36            System.out.println(0);
37        } else {
38            System.out.println(res);
39        }
40    }
41 }
```

Python

```
1 def main():
2     import sys
3     input_line = sys.stdin.readline().strip()
4     stack = []
5     res = 0
6
7     # 括号映射
8     mp = {')': '(', ']': '[', '}': '{'}
9
10    for c in input_line:
11        # 左括号压栈
12        if c in '([{':
13            stack.append(c)
14            res = max(res, len(stack))
15        else:
16            # 栈空或不匹配
17            if not stack or stack[-1] != mp.get(c):
18                print(0)
19                return
20            stack.pop()
21
22    # 栈未清空, 左右数量不等
23    if stack:
24        print(0)
25    else:
26        print(res)
27
28 if __name__ == '__main__':
29     main()
```

JavaScript

```
1 const readline = require('readline');
2 const rl = readline.createInterface({ input: process.stdin });
3
4 rl.on('line', function(line) {
5     const input = line.trim();
6     const stack = [];
7     let res = 0;
8     // 括号映射
9     const mp = { ')': '(', ']': '[', '}': '{' };
10
11    for (let c of input) {
12        // // 左括号压栈
13        if (c === '(' || c === '[' || c === '{') {
14            stack.push(c);
15            res = Math.max(res, stack.length);
16        } else {
17            // 无效字符串: 栈空 或者 括号不匹配
18            if (stack.length === 0 || stack[stack.length - 1] !== mp[c]) {
19                console.log(0);
20                return;
21            }
22            stack.pop();
23        }
24    }
25    // 栈未清空, 左右数量不等
26    if (stack.length !== 0) {
27        console.log(0);
28    } else {
29        console.log(res);
30    }
31});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 func main() {
11     reader := bufio.NewReader(os.Stdin)
12     input, _ := reader.ReadString('\n')
13     input = strings.TrimSpace(input) // 自动去除换行和空白
14
15     stack := []rune{}
16     res := 0
17
18     // 括号匹配映射表
19     mp := map[rune]rune{')': '(', ']': '[', '}': '{'}
20
21     for _, c := range input {
22         if c == '(' || c == '[' || c == '{' {
23             stack = append(stack, c)
24             if len(stack) > res {
25                 res = len(stack) // 更新最大嵌套深度
26             }
27         } else {
28             // 无效字符串: 栈空 或者 括号不匹配
29             if len(stack) == 0 || stack[len(stack)-1] != mp[c] {
30                 fmt.Println(0)
31                 return
32             }
33             stack = stack[:len(stack)-1]
34         }
35     }
36     // 栈未清空, 左右数量不等
37     if len(stack) != 0 {
38         fmt.Println(0)
39     } else {
40         fmt.Println(res)
41     }
42 }
```

| 来自: 华为OD机考2025C卷 – 最大括号深度 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考2025C卷 – 最大括号深度 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 流量波峰 (C++ & Python & JAVA & JS & GO)-CSDN博客

流量波峰

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 100分题型

题目描述

一线运维人员在对通话流量进行监控，每一段时间内都是出现流量的高峰，流量有高有低形成一个个波峰波谷，运维人员想找到流量变化最快的波峰，你可以帮助他吗？

给定一个整数数组nums，代表采样点的流量值，请找到满足以下条件的三元组(i,j,k):其中 $i < j < k$, $nums[j] > nums[i]$ 且 $nums[j] > nums[k]$ （即j是峰顶），并找到所有满足条件的三元组中(k-i)的最小值

输入描述

第一行为n个整数，表示数组中的n个元素， $0 \leq n \leq 100000$

输出描述

返回所有满足条件的三元组中(k-i)的最小值。若不存在，返回-1。

用例1

输入

```
1 3 5 4 7 2 1
```

输出

```
1 2
```

说明

▼

Plain Text |

1 满足条件的三元组为 [0, 1, 2], 距离2

用例2

输入

▼

Plain Text |

1 4 3 2 1

输出

▼

Plain Text |

1 -1

说明

▼

Plain Text |

1 无法找到满足条件的三元组，返回-1

题解

思路: 单调栈

1. 题目要想找到(i,j,k)的三元组，要求 `nums[j] > nums[i]` 且 `nums[j] > nums[k]` 的三元组，并且想让 `k-j` 的长度最小。对于j来说 其实就是找左侧离自己最近的小于自己值的位置，右侧离自己最近小于自己值的位置
2. 根据1的思路可以使用 单调栈(维持栈中递增) 分别求出每个点左侧离自己最近的并小于自己的位置、右侧离自己最近并小于自己的位置。使用数组 `left right` 分别保存对应索引。额外需要注意如果左侧或右侧不存在比自己小的，将对应数组设置为-1
3. 循环枚举每个点 j，此时已经直到 `left[j]` 和 `right[j]`，记录其中 `right[j]- left[j]` 最近距离即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11 // 通用 切割函数 函数 将字符串str根据delimiter进行切割
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 int main() {
27     // 记录每一层出现指定关键字的数量
28     vector<map<string, int>> cnts(11);
29     int n;
30     cin >> n;
31     for (int i = 0; i < n; i++) {
32         string url;
33         cin >> url;
34         // 按照这个规律进行切割之后 索引为0的位置为空字符串
35         vector<string> parts = split(url, "/");
36         for (int j = 1; j < parts.size(); j++) {
37             cnts[j][parts[j]] += 1;
38         }
39     }
40
41     string name;
42     int L;
43     cin >> L;
44     cin >> name;
45     cout << cnts[L][name] << endl;
```

```
46  
47  
48     return 0;  
49 }
```

JAVA

```
1 import java.util.*;
2 import java.io.*;
3
4 public class Main {
5     // 找到左边小于本身高度的最近点
6     static int[] findLeftSmaller(int[] nums) {
7         int n = nums.length;
8         int[] left = new int[n];
9         Arrays.fill(left, -1);
10        // 单调栈 维持一个单调递增
11        Stack<Integer> stack = new Stack<>();
12        for (int i = 0; i < n; i++) {
13            while (!stack.isEmpty() && nums[stack.peek()] >= nums[i]) {
14                stack.pop();
15            }
16            if (!stack.isEmpty()) {
17                left[i] = stack.peek();
18            }
19            stack.push(i);
20        }
21        return left;
22    }
23
24    // 找到右边小于本身高度的最近点
25    static int[] findRightSmaller(int[] nums) {
26        int n = nums.length;
27        int[] right = new int[n];
28        Arrays.fill(right, -1);
29        // 单调栈 维持单调递增
30        Stack<Integer> stack = new Stack<>();
31        for (int i = n - 1; i >= 0; i--) {
32            while (!stack.isEmpty() && nums[stack.peek()] >= nums[i]) {
33                stack.pop();
34            }
35            if (!stack.isEmpty()) {
36                right[i] = stack.peek();
37            }
38            stack.push(i);
39        }
40        return right;
41    }
42
43    public static void main(String[] args) throws IOException {
44        BufferedReader br = new BufferedReader(new InputStreamReader(Syste
m.in));

```

```
45     String line = br.readLine();
46     String[] parts = line.trim().split(" ");
47     int n = parts.length;
48     int[] heights = new int[n];
49     for (int i = 0; i < n; i++) {
50         heights[i] = Integer.parseInt(parts[i]);
51     }
52
53     // 数量小于三肯定不存在指定元组
54     if (n < 3) {
55         System.out.println(-1);
56         return;
57     }
58
59     int[] left = findLeftSmall(heights);
60     int[] right = findRightSmaller(heights);
61
62     int res = Integer.MAX_VALUE;
63     for (int j = 0; j < n; j++) {
64         // 左侧小于自己最近的点
65         int i = left[j];
66         int k = right[j];
67         // 左侧或右侧不存在这样的点
68         if (i == -1 || k == -1) continue;
69         res = Math.min(res, k - i);
70     }
71
72     // 输出结果
73     if (res == Integer.MAX_VALUE) {
74         System.out.println(-1);
75     } else {
76         System.out.println(res);
77     }
78 }
79 }
```

Python

```
1 import sys
2
3 # 找到左边小于本身高度的最近点
4 def find_left_small(nums):
5     n = len(nums)
6     left = [-1] * n
7     # 单调栈 维持一个单调递增
8     stack = []
9     for i in range(n):
10         while stack and nums[stack[-1]] >= nums[i]:
11             stack.pop()
12         if stack:
13             left[i] = stack[-1]
14         stack.append(i)
15     return left
16
17 # 找到右边小于本身高度的最近点
18 def find_right_smaller(nums):
19     n = len(nums)
20     right = [-1] * n
21     # 单调栈 维持单调递增
22     stack = []
23     for i in range(n - 1, -1, -1):
24         while stack and nums[stack[-1]] >= nums[i]:
25             stack.pop()
26         if stack:
27             right[i] = stack[-1]
28         stack.append(i)
29     return right
30
31 def main():
32     line = sys.stdin.readline().strip()
33     heights = list(map(int, line.split()))
34     n = len(heights)
35
36     # 数量小于三肯定不存在指定元组
37     if n < 3:
38         print(-1)
39         return
40
41     left = find_left_small(heights)
42     right = find_right_smaller(heights)
43
44     res = float("inf")
45     for j in range(n):
```

```
46     # 左侧小于自己最近的点
47     i = left[j]
48     k = right[j]
49     # 左侧或右侧不存在这样的点
50     if i == -1 or k == -1:
51         continue
52     res = min(res, k - i)
53
54     # 输出结果
55     print(-1 if res == float("inf") else res)
56
57 if __name__ == "__main__":
58     main()
```

JavaScript

```
1 const readline = require("readline");
2
3 function findLeftSmall(nums) {
4     let n = nums.length;
5     let left = new Array(n).fill(-1);
6     // 单调栈 维持一个单调递增
7     let stack = [];
8     for (let i = 0; i < n; i++) {
9         while (stack.length && nums[stack[stack.length - 1]] >= nums[i]) {
10             stack.pop();
11         }
12         if (stack.length) {
13             left[i] = stack[stack.length - 1];
14         }
15         stack.push(i);
16     }
17     return left;
18 }
19
20 function findRightSmaller(nums) {
21     let n = nums.length;
22     let right = new Array(n).fill(-1);
23     // 单调栈 维持单调递增
24     let stack = [];
25     for (let i = n - 1; i >= 0; i--) {
26         while (stack.length && nums[stack[stack.length - 1]] >= nums[i]) {
27             stack.pop();
28         }
29         if (stack.length) {
30             right[i] = stack[stack.length - 1];
31         }
32         stack.push(i);
33     }
34     return right;
35 }
36
37 const rl = readline.createInterface({
38     input: process.stdin,
39     output: process.stdout,
40 });
41
42 let input = "";
43 rl.on("line", (line) => {
44     input = line.trim();
45     let heights = input.split(" ").map(Number);
```

```
46     let n = heights.length;
47
48     // 数量小于三肯定不存在指定元组
49     if (n < 3) {
50         console.log(-1);
51         rl.close();
52         return;
53     }
54
55     let left = findLeftSmall(heights);
56     let right = findRightSmaller(heights);
57
58     let res = Infinity;
59     for (let j = 0; j < n; j++) {
60         // 左侧小于自己最近的点
61         let i = left[j];
62         let k = right[j];
63         // 左侧或右侧不存在这样的点
64         if (i === -1 || k === -1) continue;
65         res = Math.min(res, k - i);
66     }
67
68     // 输出结果
69     console.log(res === Infinity ? -1 : res);
70     rl.close();
71 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8     "strconv"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13     writer := bufio.NewWriter(os.Stdout)
14     defer writer.Flush()
15
16     // 读取第一行, 表示有多少个 url
17     var n int
18     fmt.Fscanln(reader, &n)
19
20     // cnts[i] 表示第 i 层的关键词出现次数
21     cnts := make([]map[string]int, 11)
22     for i := range cnts {
23         cnts[i] = make(map[string]int)
24     }
25
26     // 逐行读取 url 并统计每一层的关键词
27     for i := 0; i < n; i++ {
28         line, _ := reader.ReadString('\n')
29         url := strings.TrimSpace(line)
30         parts := strings.Split(url, "/")
31         for j := 1; j < len(parts); j++ {
32             cnts[j][parts[j]]++
33         }
34     }
35
36     // 读取最后一行: 格式为 L key
37     lastLine, _ := reader.ReadString('\n')
38     tokens := strings.Fields(lastLine)
39     L, _ := strconv.Atoi(tokens[0])
40     key := tokens[1]
41
42     fmt.Fprintln(writer, cnts[L][key])
43 }
```

| 来自: 华为OD机试2025C卷 - 流量波峰 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 模拟目录管理 (C++ & Python & JAVA & JS & GO)-CSDN博客

模拟目录管理

华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 200分题型

题目描述

实现一个模拟目录管理功能的软件，输入一个命令序列，输出最后一条命令运行结果。

支持命令：

- 创建目录命令：mkdir 目录名称，如 mkdir abc 为在当前目录创建abc目录，如果已存在同名目录则不执行任何操作。此命令无输出。
- 进入目录命令：cd 目录名称，如 cd abc 为进入abc目录，特别地，cd ... 为返回上级目录，如果目录不存在则不执行任何操作。此命令无输出。
- 查看当前所在路径命令：pwd，输出当前路径字符串。

约束：

- 目录名称仅支持小写字母；mkdir 和 cd 命令的参数仅支持单个目录，如：mkdir abc 和 cd abc；不支持嵌套路径和绝对路径，如 mkdir abc/efg，cd abc/efg，mkdir /abc/efg，cd /abc/efg 是不支持的。
- 目录符号为/，根目录/作为初始目录。
- 任何不符合上述定义的无效命令不做任何处理并且无输出。

输入描述

输入 N 行字符串，每一行字符串是一条命令。

命令行数限制100行以内，目录名称限制10个字符以内。

输出描述

输出最后一条命令运行结果字符串。

示例1

输入

```
Plain Text |  
1 mkdir abc  
2 cd abc  
3 pwd
```

输出

```
Plain Text |  
1 /abc/
```

说明

在根目录创建一个abc的目录并进入abc目录中查看当前目录路径，输出当前路径/abc/。

题解

思路： 模拟

1. 目录结构本身就类似 树 的结构，可以定义一个类或者结构来表示树的节点。
2. 每个树节点中包含节点路径名称，下级目录(文件系统中有个父节点为...也可以当作下级目录处理)。
3. 定义好树节点结构之后，初始定义一个根目录，并使用 `currentDirectory` 保存当前目录，初始将 `currentDirectory` 设置为根目录。接下来按照题目要求进行模拟即可，具体逻辑可参照下面代码。

C++

```
1 #include <iostream>
2 #include <string>
3 #include <unordered_map>
4 #include <sstream>
5
6 using namespace std;
7
8 // 表示文件系统中每个目录的类
9 class Node {
10 public:
11     string path; // 目录的路径
12     unordered_map<string, Node*> next; // 当前目录下的子目录映射，键为目录名，值
13     // 为Node指针
14     Node(string path, Node* parent) : path(path) {
15         // 如果存在父目录，则在子目录映射中添加指向父目录的条目
16         if (parent != nullptr) {
17             this->next[".."] = parent;
18         }
19     }
20 };
21
22 // 检查目录名是否有效，目录名只能包含小写字母
23 bool isValidDirectoryName(const string& name) {
24     for (char c : name) {
25         if (c < 'a' || c > 'z') {
26             return false; // 目录名包含非小写字母字符
27         }
28     }
29     return true; // 目录名有效
30 }
31
32 // 检查是否可以切换到指定的目录
33 bool isValidChangeDirectory(const string& name) {
34     return name == ".." || isValidDirectoryName(name); // 目录名为".."或者有
35     // 效的目录名
36 }
37
38 int main() {
39     Node* root = new Node("/", nullptr); // 根目录，没有父目录
40     Node* currentDirectory = root; // 初始化当前目录为根目录
41     string lastOutput; // 用于存储最后输出的路径
42
43     string input;
44     while (getline(cin, input)) {
```

```
44     istringstream iss(input);
45     string command, arg;
46     iss >> command;
47
48     if (command == "mkdir") {
49         iss >> arg;
50         if (isValidDirectoryName(arg)) {
51             // 创建一个新目录并添加到当前目录的子目录映射中
52             if (currentDirectory->next.find(arg) == currentDirectory->
53                 next.end()) {
54                 currentDirectory->next[arg] = new Node(currentDirec-
55                 tory->path + arg + "/", currentDirectory);
56             }
57         }
58     } else if (command == "cd") {
59         iss >> arg;
60         if (isValidChangeDirectory(arg)) {
61             // 切换当前目录
62             auto it = currentDirectory->next.find(arg);
63             if (it != currentDirectory->next.end()) {
64                 currentDirectory = it->second; // 切换到目标目录
65             }
66         }
67     } else if (command == "pwd") {
68         // 输出当前目录的路径
69         lastOutput = currentDirectory->path;
70     }
71     cout << lastOutput << endl; // 打印最后保存的路径
72
73 }
```

Java

```
1 import java.util.HashMap;
2 import java.util.Map;
3 import java.util.Scanner;
4
5 class Main {
6     // 表示文件系统中的每个目录
7     static class Node {
8         String path; // 目录的路径
9         Map<String, Node> next = new HashMap<>(); // 当前目录下的子目录映射
10
11     Node(String path, Node parent) {
12         this.path = path;
13         // 如果存在父目录，则在子目录映射中添加指向父目录的条目
14         if (parent != null) {
15             this.next.put("../", parent);
16         }
17     }
18 }
19
20 // 检查目录名是否有效，目录名只能包含小写字母
21 static boolean isValidDirectoryName(String name) {
22     for (char c : name.toCharArray()) {
23         if (c < 'a' || c > 'z') {
24             return false; // 目录名包含非小写字母字符
25         }
26     }
27     return true; // 目录名有效
28 }
29
30 // 检查是否可以切换到指定的目录
31 static boolean isValidChangeDirectory(String name) {
32     return name.equals("../") || isValidDirectoryName(name); // 目录名为".."或者有效的目录名
33 }
34
35 public static void main(String[] args) {
36     Scanner scanner = new Scanner(System.in);
37     Node root = new Node("/", null); // 根目录，没有父目录
38     Node currentDirectory = root; // 初始化当前目录为根目录
39     String lastOutput = ""; // 用于存储最后输出的路径
40
41     while (scanner.hasNextLine()) {
42         String input = scanner.nextLine();
43         String[] parts = input.split(" ");
44         String command = parts[0];
```

```
45     String arg = parts.length > 1 ? parts[1] : "";
46
47     if (command.equals("mkdir")) {
48         if (isValidDirectoryName(arg)) {
49             // 创建一个新目录并添加到当前目录的子目录映射中
50             if (!currentDirectory.next.containsKey(arg)) {
51                 currentDirectory.next.put(arg, new Node(currentDir
52                     ectory.path + arg + "/", currentDirectory));
53             }
54         }
55     } else if (command.equals("cd")) {
56         if (isValidChangeDirectory(arg)) {
57             // 切换当前目录
58             Node nextNode = currentDirectory.next.get(arg);
59             if (nextNode != null) {
60                 currentDirectory = nextNode; // 切换到目标目录
61             }
62         }
63     } else if (command.equals("pwd")) {
64         // 输出当前目录的路径
65         lastOutput = currentDirectory.path;
66     }
67 }
68 System.out.println(lastOutput); // 打印最后保存的路径
69 }
70 }
```

Python

```
1 class FileSystem:
2     # 表示文件系统中的每个目录
3     class Node:
4         def __init__(self, path, parent=None):
5             self.path = path # 目录的路径
6             self.next = {} # 当前目录下的子目录映射
7             if parent is not None:
8                 self.next[".."] = parent # 如果存在父目录, 添加指向父目录的条目
9
10    # 检查目录名是否有效, 目录名只能包含小写字母
11    @staticmethod
12    def is_valid_directory_name(name):
13        return name.islower() and name.isalpha()
14
15    # 检查是否可以切换到指定的目录
16    @staticmethod
17    def is_valid_change_directory(name):
18        return name == ".." or FileSystem.is_valid_directory_name(name)
19
20    def __init__(self):
21        self.root = FileSystem.Node("/") # 根目录
22        self.current_directory = self.root # 初始化当前目录为根目录
23        self.last_output = ""
24
25    def process_input(self):
26        import sys
27        for input_line in sys.stdin:
28            input_line = input_line.strip()
29            parts = input_line.split()
30            command = parts[0]
31            arg = parts[1] if len(parts) > 1 else ""
32
33            if command == "mkdir":
34                if self.is_valid_directory_name(arg):
35                    # 创建一个新目录并添加到当前目录的子目录映射中
36                    if arg not in self.current_directory.next:
37                        self.current_directory.next[arg] = FileSystem.Node(
38                            self.current_directory.path + arg + "/", self.current_directory)
39
39            elif command == "cd":
40                if self.is_valid_change_directory(arg):
41                    # 切换当前目录
42                    if arg in self.current_directory.next:
43                        self.current_directory = self.current_directory.ne
xt[arg]
```

```
44
45     elif command == "pwd":
46         # 输出当前目录的路径
47         self.last_output = self.current_directory.path
48
49     print(self.last_output) # 打印最后保存的路径
50
51
52 if __name__ == "__main__":
53     fs = FileSystem()
54     fs.process_input()
```

JavaScript

```
1 class Node {
2     constructor(path, parent = null) {
3         this.path = path; // 目录的路径
4         this.next = {}; // 当前目录下的子目录映射
5         if (parent !== null) {
6             this.next[".."] = parent; // 如果存在父目录，则添加指向父目录的条目
7         }
8     }
9 }
10
11 class FileSystem {
12     // 检查目录名是否有效，目录名只能包含小写字母
13     static isValidDirectoryName(name) {
14         return /^[a-z]+$/i.test(name);
15     }
16
17     // 检查是否可以切换到指定的目录
18     static isValidChangeDirectory(name) {
19         return name === ".." || this.isValidDirectoryName(name);
20     }
21
22     constructor() {
23         this.root = new Node("/"); // 根目录
24         this.currentDirectory = this.root; // 初始化当前目录为根目录
25         this.lastOutput = "";
26     }
27
28     processInput() {
29         const readline = require("readline");
30         const rl = readline.createInterface({
31             input: process.stdin,
32             output: process.stdout
33         });
34
35         rl.on("line", (input) => {
36             let [command, arg] = input.split(" ");
37             if (command === "mkdir") {
38                 if (FileSystem.isValidDirectoryName(arg)) {
39                     // 创建一个新目录并添加到当前目录的子目录映射中
40                     if (!(arg in this.currentDirectory.next)) {
41                         this.currentDirectory.next[arg] = new Node(this.currentDirectory.path + arg + "/", this.currentDirectory);
42                     }
43                 }
44             } else if (command === "cd") {
```

```
45         if (FileSystem.isValidChangeDirectory(arg)) {
46             // 切换当前目录
47             if (arg in this.currentDirectory.next) {
48                 this.currentDirectory = this.currentDirectory.next
49             }
50         }
51     } else if (command === "pwd") {
52         // 输出当前目录的路径
53         this.lastOutput = this.currentDirectory.path;
54     }
55 });
56
57 rl.on("close", () => {
58     console.log(this.lastOutput); // 打印最后保存的路径
59 });
60 }
61 }
62
63 const fs = new FileSystem();
64 fs.processInput();
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 表示文件系统中的每个目录
11 type Node struct {
12     Path string           // 目录的路径
13     Next map[string]*Node // 当前目录下的子目录映射
14 }
15
16 func NewNode(path string, parent *Node) *Node {
17     node := &Node{Path: path, Next: make(map[string]*Node)}
18     if parent != nil {
19         node.Next[".."] = parent // 如果存在父目录，添加指向父目录的条目
20     }
21     return node
22 }
23
24 // 检查目录名是否有效，目录名只能包含小写字母
25 func isValidDirectoryName(name string) bool {
26     for _, c := range name {
27         if c < 'a' || c > 'z' {
28             return false // 目录名包含非小写字母字符
29         }
30     }
31     return true // 目录名有效
32 }
33
34 // 检查是否可以切换到指定的目录
35 func isValidChangeDirectory(name string) bool {
36     return name == ".." || isValidDirectoryName(name)
37 }
38
39 func main() {
40     root := NewNode("/", nil) // 根目录，没有父目录
41     currentDirectory := root // 初始化当前目录为根目录
42     var lastOutput string    // 用于存储最后输出的路径
43
44     scanner := bufio.NewScanner(os.Stdin)
45     for scanner.Scan() {
```

```
46     input := scanner.Text()
47     parts := strings.Fields(input)
48     command := parts[0]
49     arg := ""
50     if len(parts) > 1 {
51         arg = parts[1]
52     }
53
54     if command == "mkdir" {
55         if isValidDirectoryName(arg) {
56             // 创建一个新目录并添加到当前目录的子目录映射中
57             if _, exists := currentDirectory.Next[arg]; !exists {
58                 currentDirectory.Next[arg] = NewNode(currentDirectory.Path+arg
59 +"/", currentDirectory)
60             }
61         }
62     } else if command == "cd" {
63         if isValidChangeDirectory(arg) {
64             // 切换当前目录
65             if nextNode, exists := currentDirectory.Next[arg]; exists {
66                 currentDirectory = nextNode
67             }
68         }
69     } else if command == "pwd" {
70         // 输出当前目录的路径
71         lastOutput = currentDirectory.Path
72     }
73 }
74 fmt.Println(lastOutput) // 打印最后保存的路径
75 }
```

| 来自: 华为OD机考2025C卷 – 模拟目录管理 (C++ & Python & JAVA & JS & GO)-CSDN博客