

t0815

[华为OD机考 2025C卷 - 评论转换输出 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 最左侧冗余覆盖子串 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 整数编码 \(C++ & Python & JAVA & JS & GO\)_华为od机试 整数编码-CSDN博客](#)

[华为OD机试 2025C卷 - 单核CPU任务调度 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025C卷 - 反转每对括号间的子串 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 统计监控 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025C卷 - 中庸行者 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 网格红绿灯最短路径 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025C卷 - 运输时间 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 计算误码率 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考2025C卷 - 数字螺旋矩阵 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 切割字符串 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 哈夫曼树 \(C++ & Python & JAVA & JS & GO\)_哈弗曼树相关的考试题-CSDN博客](#)

[华为OD机试 2025C卷 - 代码编辑器 \(C++ & Python & JAVA & JS & GO\)_华为代码编辑器-csdn-CSDN博客](#)

[华为OD机试 2025C卷 - 构成正方形的数量 \(C++ & Python & JAVA & JS & GO\)_华为od机试2025a卷 - 构成正方形的数量-CSDN博客](#)

华为OD机考 2025C卷 - 评论转换输出 (C++ & Python & JAVA & JS & GO)-CSDN博客

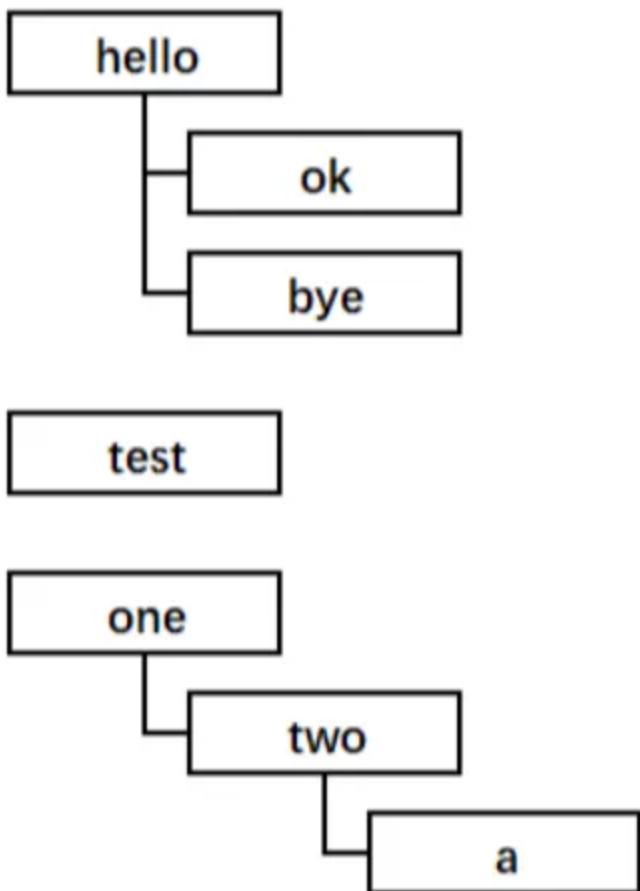
评论转换输出

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

在一个[博客网站](#)上，每篇博客都有评论。每一条评论都是一个非空英文字母字符串。评论具有树状结构，除了根评论外，每个评论都有一个父评论。当评论保存时，使用以下格式：首先是评论的内容；然后是回复当前评论的数量。最后是当前评论的所有子评论。（子评论使用相同的格式嵌套存储）所有元素之间都用单个逗号分隔。例如，如果评论如下：



第一条评论是"hello,2,ok,0,bye,0"，第二条评论是"test,0"，第三条评论是"one,1,two,1,a,0"。
所有评论被保存成"hello,2,ok,0,bye,0,test,0,one,1,two,1,a,0"。

对于上述格式的评论，请以另外一种格式打印：

首先打印评论嵌套的最大深度。然后是打印n行，第 i ($1 \leq i \leq n$) 行对应于嵌套级别为 i 的评论（根评论的嵌套级别为1）。对于第 i 行，嵌套级别的评论按照它们出现的顺序打印，用空格分隔开

输入描述

一行评论。由英文字母、数字和英文逗号组成。

- 保证每个评论都是由英文字符组成的非空字符串。
- 每个评论的数量都是整数（至少由一个数字组成）。
- 整个字符串的长度不超过 10^6 。
- 给定的评论结构保证是合法的

输出描述

按照给定的格式打印评论。对于每一级嵌套，评论应该按照输入中的顺序打印

用例1

输入

```
Plain Text |  
1 hello,2,ok,0,bye,0,test,0,one,1,two,1,a,0
```

输出

```
Plain Text |  
1 3  
2 hello test one  
3 ok bye two  
4 a
```

用例2

输入

```
Plain Text |  
1 A,5,A,0,a,0,A,0,a,0,A,0
```

输出

```
1 2  
2 A  
3 A a A a A
```

Plain Text |

用例3

输入

```
1 A,3,B,2,C,0,D,1,E,0,F,1,G,0,H,1,I,1,J,0,K,1,L,0,M,2,N,0,0,1,P,0
```

Plain Text |

输出

```
1 4  
2 A K M  
3 B F H L N O  
4 C D G I P  
5 E J
```

Plain Text |

题解

思路： **DFS**

1. 题目要求实际根据输入的字符串构建一棵多叉树，并按层级输出每层的节点。
2. 首先对输入字符串按照 **,** 进行切割。切割得到的数组顺序{评论顺序，子评论数目}这样的结构
3. 由于题目输出结果不需要明确知道子评论属于哪个父评论，只需要确定层级即可，直接采用数组进行存储每层评论即可。接下来使用 **DFS** 处理评论层级即可。
4. 按照要求输出构建之后的结果即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<queue>
8 using namespace std;
9
10 // 通用 split 函数
11 vector<string> split(const string& str, const string& delimiter) {
12     vector<string> result;
13     size_t start = 0;
14     size_t end = str.find(delimiter);
15     while (end != string::npos) {
16         result.push_back(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(str.substr(start));
22     return result;
23 }
24
25 // 递归处理嵌套评论
26 void dfs(queue<string> &pq, vector<vector<string>> &res, int level) {
27     if (res.size() < level) {
28         res.push_back({});
29     }
30     // 评论内容
31     string comment = pq.front();
32     pq.pop();
33     // 子评论内容
34     string subComment = pq.front();
35     int subCommentCount = stoi(subComment);
36     pq.pop();
37
38     res[level-1].push_back(comment);
39
40     for (int i = 0; i < subCommentCount; i++) {
41         dfs(pq, res, level+1);
42     }
43 }
44
45 // 打印输出结果
```

```
46 void printResult(vector<vector<string>> ans) {
47     int n = ans.size();
48     cout << n << endl;
49     for (int i = 0; i < ans.size(); i++) {
50         vector<string> tmp = ans[i];
51         for (int i = 0; i < tmp.size(); i++) {
52             cout << tmp[i];
53             if (i != tmp.size() - 1) {
54                 cout << " ";
55             }
56         }
57         cout << endl;
58     }
59 }
60
61
62 int main() {
63     string s;
64     getline(cin, s);
65     vector<string> ans = split(s, ",");
66     vector<vector<string>> res;
67     queue<string> qp;
68     for (int i = 0; i < ans.size(); i++) {
69         qp.push(ans[i]);
70     }
71     // 递归构建结果
72     while (!qp.empty()) {
73         dfs(qp, res, 1);
74     }
75
76     // 格式化输出结果
77     printResult(res);
78
79     return 0;
80 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4
5     // 递归处理嵌套评论
6     static void dfs(Queue<String> pq, List<List<String>> res, int level) {
7         if (res.size() < level) {
8             res.add(new ArrayList<>());
9         }
10        // 评论内容
11        String comment = pq.poll();
12        // 子评论数量
13        int subCommentCount = Integer.parseInt(pq.poll());
14
15        res.get(level - 1).add(comment);
16
17        for (int i = 0; i < subCommentCount; i++) {
18            dfs(pq, res, level + 1);
19        }
20    }
21
22    // 打印输出结果
23    static void printResult(List<List<String>> ans) {
24        System.out.println(ans.size());
25        for (List<String> tmp : ans) {
26            for (int i = 0; i < tmp.size(); i++) {
27                System.out.print(tmp.get(i));
28                if (i != tmp.size() - 1) {
29                    System.out.print(" ");
30                }
31            }
32            System.out.println();
33        }
34    }
35
36    public static void main(String[] args) {
37        Scanner sc = new Scanner(System.in);
38        String s = sc.nextLine();
39        // 分割字符串
40        String[] arr = s.split(",");
41
42        Queue<String> qp = new LinkedList<>();
43        for (String str : arr) {
44            qp.add(str);
45        }
46    }
47}
```

```
46     List<List<String>> res = new ArrayList<>();
47
48     // 递归构建结果
49     while (!qp.isEmpty()) {
50         dfs(qp, res, 1);
51     }
52
53     // 格式化输出
54     printResult(res);
55 }
56 }
```

Python

```
1  from collections import deque
2
3  # 递归处理嵌套评论
4  def dfs(pq, res, level):
5      if len(res) < level:
6          res.append([])
7      # 评论内容
8      comment = pq.popleft()
9      # 子评论数量
10     sub_comment_count = int(pq.popleft())
11
12     res[level - 1].append(comment)
13
14     for _ in range(sub_comment_count):
15         dfs(pq, res, level + 1)
16
17 # 打印输出结果
18 def print_result(ans):
19     print(len(ans))
20     for row in ans:
21         print(" ".join(row))
22
23 if __name__ == "__main__":
24     s = input().strip()
25     # 分割字符串
26     arr = s.split(",")
27
28     qp = deque(arr)
29     res = []
30
31     # 递归构建结果
32     while qp:
33         dfs(qp, res, 1)
34
35     # 格式化输出
36     print_result(res)
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on("line", (line) => {
11     inputLines.push(line.trim());
12 });
13
14 rl.on("close", () => {
15     let s = inputLines[0];
16     let arr = s.split(",");
17
18     // 递归处理嵌套评论
19     function dfs(qp, res, level) {
20         if (res.length < level) {
21             res.push([]);
22         }
23         // 评论内容
24         let comment = qp.shift();
25         // 子评论数量
26         let subCommentCount = parseInt(qp.shift(), 10);
27
28         res[level - 1].push(comment);
29
30         for (let i = 0; i < subCommentCount; i++) {
31             dfs(qp, res, level + 1);
32         }
33     }
34
35     // 打印输出结果
36     function printResult(ans) {
37         console.log(ans.length);
38         for (let row of ans) {
39             console.log(row.join(" "));
40         }
41     }
42
43     let qp = arr.slice();
44     let res = [];
45
```

```
46     // 递归构建结果
47     while (qp.length > 0) {
48         dfs(qp, res, 1);
49     }
50
51     // 格式化输出
52     printResult(res);
53 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 递归处理嵌套评论
12 func dfs(qp *[]string, res *[][]string, level int) {
13     if len(*res) < level {
14         *res = append(*res, []string{})
15     }
16     // 评论内容
17     comment := (*qp)[0]
18     *qp = (*qp)[1:]
19     // 子评论数量
20     subCommentCount, _ := strconv.Atoi((*qp)[0])
21     *qp = (*qp)[1:]
22
23     (*res)[level-1] = append((*res)[level-1], comment)
24
25     for i := 0; i < subCommentCount; i++ {
26         dfs(qp, res, level+1)
27     }
28 }
29
30 // 打印输出结果
31 func printResult(res [][]string) {
32     fmt.Println(len(res))
33     for _, row := range res {
34         fmt.Println(strings.Join(row, " "))
35     }
36 }
37
38 func main() {
39     reader := bufio.NewReader(os.Stdin)
40     line, _ := reader.ReadString('\n')
41     line = strings.TrimSpace(line)
42     arr := strings.Split(line, ",")
43
44     qp := make([]string, len(arr))
45     copy(qp, arr)
```

```
46
47     var res [][]string
48
49     // 递归构建结果
50     for len(qp) > 0 {
51         dfs(&qp, &res, 1)
52     }
53
54     // 格式化输出
55     printResult(res)
56 }
```

| 来自: 华为OD机考 2025C卷 – 评论转换输出 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 最左侧冗余覆盖子串 (C++ & Python & JAVA & JS & GO)-CSDN博客

最左侧冗余覆盖子串

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

给定两个字符串s1和s2和正整数K，其中s1长度为n1，s2长度为n2，在s2中选一个子串，满足：

- 该子串长度为n1+k
- 该子串中包含s1中全部字母，
- 该子串每个字母出现次数不小于s1中对应的字母，

我们称s2以长度k冗余覆盖s1，给定s1, s2, k，求最左侧的s2以长度k冗余覆盖s1的子串的首个元素的下标，如果没有返回-1。

输入描述

输入三行，第一行为s1，第二行为s2，第三行为k，s1和s2只包含小写字母

备注

- $0 \leq \text{len}(s1) \leq 1000000$
- $0 \leq \text{len}(s2) \leq 20000000$
- $0 \leq k \leq 1000$

输出描述

最左侧的s2以长度k冗余覆盖s1的子串首个元素下标，如果没有返回-1

示例1

输入

```
1 ab
2 aabcd
3 1
```

输出

Plain Text |

```
1 0
```

说明

| 子串aab和abc符合要求，由于aab在abc的左侧，因此输出aab的下标：0

示例2

输入

Plain Text |

```
1 abc
2 dfs
3 10
```

输出

Plain Text |

```
1 -1
```

说明

| s2无法覆盖s1，输出 -1

题解

思路：题解涉及算法 滑动窗口，窗口大小为 `s1.length + k`

- 使用`remainingMatches`保存待匹配的字符数量。使用`patternFreq`统计s1中各个字母的出现的数量。使用`windowFreq`统计s2位于窗口中各个字母的出现的数量。
- 具体执行过程一下：当窗口右移时
 - 右边界右移(`right++`)，增加`windowFreq[rightChar - 'a']++`，当`windowFreq[rightChar - 'a'] <= patternFreq[rightChar - 'a']`时，`remainingMatches --`。
 - 左边界右移之前，判断`windowFreq[leftChar - 'a'] <= patternFreq[leftChar - 'a']`如果成立，`remainingMatches++`，`windowFreq[leftChar - 'a']--`，`left++`
- 在窗口移动过程，存在`remainingMatches = 0`时，既满足条件，输出这时的`left`既是结果。

C++

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 using namespace std;
6
7 int main() {
8     string pattern, text;
9     getline(cin, pattern); // 读取模式串
10    getline(cin, text);   // 读取目标文本
11    int maxExtraChars;
12    cin >> maxExtraChars; // 允许的额外字符数
13
14    int startIndex = -1;
15    vector<int> patternFreq(26, 0); // 记录 pattern 中每个字符的出现次数
16    for (char c : pattern) {
17        patternFreq[c - 'a']++;
18    }
19
20    int left = 0, right = 0;
21    // 待需要匹配字符数量
22    int remainingMatches = pattern.length();
23    vector<int> windowFreq(26, 0); // 记录滑动窗口中的字符出现次数
24
25    while (right < text.length()) {
26        char rightChar = text[right];
27        windowFreq[rightChar - 'a']++;
28
29        if (windowFreq[rightChar - 'a'] <= patternFreq[rightChar - 'a']) {
30            remainingMatches--;
31        }
32
33        if (right - left + 1 > pattern.length() + maxExtraChars) {
34            char leftChar = text[left];
35            if (windowFreq[leftChar - 'a'] <= patternFreq[leftChar - 'a']) {
36                remainingMatches++;
37            }
38            windowFreq[leftChar - 'a']--;
39            left++;
40        }
41
42        if (remainingMatches == 0) {
43            startIndex = left;
44            break;
45    }
46}
```

```
45         }
46
47         right++;
48     }
49
50     cout << startIndex;
51
52 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取模式串
8         String pattern = scanner.nextLine();
9         // 读取目标文本
10        String text = scanner.nextLine();
11        // 允许的额外字符数
12        int maxExtraChars = scanner.nextInt();
13        scanner.close();
14
15        int startIndex = -1;
16        int[] patternFreq = new int[26]; // 记录 pattern 中每个字符的出现次数
17        for (char c : pattern.toCharArray()) {
18            patternFreq[c - 'a']++;
19        }
20
21        int left = 0, right = 0;
22        int remainingMatches = pattern.length(); // 待匹配字符数量
23        int[] windowFreq = new int[26]; // 记录滑动窗口中的字符出现次数
24
25        while (right < text.length()) {
26            char rightChar = text.charAt(right);
27            windowFreq[rightChar - 'a']++;
28
29            if (windowFreq[rightChar - 'a'] <= patternFreq[rightChar - 'a']) {
30                remainingMatches--;
31            }
32
33            if (right - left + 1 > pattern.length() + maxExtraChars) {
34                char leftChar = text.charAt(left);
35                if (windowFreq[leftChar - 'a'] <= patternFreq[leftChar - 'a']) {
36                    remainingMatches++;
37                }
38                windowFreq[leftChar - 'a']--;
39                left++;
40            }
41
42            if (remainingMatches == 0) {
43                startIndex = left;
```

```
44             break;
45         }
46
47         right++;
48     }
49
50     System.out.println(startIndex);
51 }
52 }
```

Python

```
1 import sys
2
3 # 读取模式串
4 pattern = sys.stdin.readline().strip()
5 # 读取目标文本
6 text = sys.stdin.readline().strip()
7 # 允许的额外字符数
8 max_extra_chars = int(sys.stdin.readline().strip())
9
10 pattern_freq = [0] * 26 # 记录 pattern 中每个字符的出现次数
11 for c in pattern:
12     pattern_freq[ord(c) - ord('a')] += 1
13
14 left, right = 0, 0
15 remaining_matches = len(pattern) # 待匹配字符数量
16 window_freq = [0] * 26 # 记录滑动窗口中的字符出现次数
17 start_index = -1
18
19 while right < len(text):
20     right_char = text[right]
21     window_freq[ord(right_char) - ord('a')] += 1
22
23     if window_freq[ord(right_char) - ord('a')] <= pattern_freq[ord(right_char) - ord('a')]:
24         remaining_matches -= 1
25
26     if right - left + 1 > len(pattern) + max_extra_chars:
27         left_char = text[left]
28         if window_freq[ord(left_char) - ord('a')] <= pattern_freq[ord(left_char) - ord('a')]:
29             remaining_matches += 1
30             window_freq[ord(left_char) - ord('a')] -= 1
31         left += 1
32
33     if remaining_matches == 0:
34         start_index = left
35         break
36
37     right += 1
38
39 print(start_index)
```

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         rl.close();
13     }
14 });
15
16 rl.on('close', () => {
17     let pattern = inputLines[0]; // 读取模式串
18     let text = inputLines[1]; // 读取目标文本
19     let maxExtraChars = parseInt(inputLines[2]); // 允许的额外字符数
20
21     let patternFreq = Array(26).fill(0); // 记录 pattern 中每个字符的出现次数
22     for (let c of pattern) {
23         patternFreq[c.charCodeAt(0) - 'a'.charCodeAt(0)]++;
24     }
25
26     let left = 0, right = 0, remainingMatches = pattern.length;
27     let windowFreq = Array(26).fill(0); // 记录滑动窗口中的字符出现次数
28     let startIndex = -1;
29
30     while (right < text.length) {
31         let rightChar = text[right];
32         windowFreq[rightChar.charCodeAt(0) - 'a'.charCodeAt(0)]++;
33
34         if (windowFreq[rightChar.charCodeAt(0) - 'a'.charCodeAt(0)] <= patternFreq[rightChar.charCodeAt(0) - 'a'.charCodeAt(0)]) {
35             remainingMatches--;
36         }
37
38         if (right - left + 1 > pattern.length + maxExtraChars) {
39             let leftChar = text[left];
40             if (windowFreq[leftChar.charCodeAt(0) - 'a'.charCodeAt(0)] <= patternFreq[leftChar.charCodeAt(0) - 'a'.charCodeAt(0)]) {
41                 remainingMatches++;
42             }
43             windowFreq[leftChar.charCodeAt(0) - 'a'.charCodeAt(0)]--;
44         }
45     }
46 }
47
48 rl.on('close', () => {
49     console.log(`Remaining matches: ${remainingMatches}`);
50 });
51
52 
```

```
44         left++;
45     }
46
47     if (remainingMatches === 0) {
48         startIndex = left;
49         break;
50     }
51
52     right++;
53 }
54
55 console.log(startIndex);
56});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8 )
9
10 func main() {
11     // 使用 bufio.Scanner 读取大数据
12     scanner := bufio.NewScanner(os.Stdin)
13
14     // 读取模式串
15     scanner.Scan()
16     pattern := scanner.Text()
17
18     // 读取目标文本
19     scanner.Scan()
20     text := scanner.Text()
21
22     // 读取允许的额外字符数 k
23     scanner.Scan()
24     maxExtraChars, _ := strconv.Atoi(scanner.Text())
25
26     // 记录 pattern 中每个字符的出现次数
27     patternFreq := make([]int, 26)
28     for _, c := range pattern {
29         patternFreq[c-'a']++
30     }
31
32     left, right := 0, 0
33     remainingMatches := len(pattern) // 待匹配字符数量
34     windowFreq := make([]int, 26)    // 记录滑动窗口中的字符出现次数
35     startIndex := -1
36
37     for right < len(text) {
38         rightChar := text[right]
39         windowFreq[rightChar-'a']++
40
41         if windowFreq[rightChar-'a'] <= patternFreq[rightChar-'a'] {
42             remainingMatches--
43         }
44
45         if right-left+1 > len(pattern)+maxExtraChars {
```

```

46     leftChar := text[left]
47     if windowFreq[leftChar-'a'] <= patternFreq[leftChar-'a'] {
48         remainingMatches++
49     }
50     windowFreq[leftChar-'a']--
51     left++
52 }
53
54     if remainingMatches == 0 {
55         startIndex = left
56         break
57     }
58
59     right++
60 }
61
62     fmt.Println(startIndex)
63 }
```

最左侧冗余覆盖子串

[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

给定两个字符串s1和s2和正整数K，其中s1长度为n1，s2长度为n2，在s2中选一个子串，满足：

- 该子串长度为n1+k
- 该子串中包含s1中全部字母，
- 该子串每个字母出现次数不小于s1中对应的字母，

我们称s2以长度k冗余覆盖s1，给定s1, s2, k，求最左侧的s2以长度k冗余覆盖s1的子串的首个元素的下标，如果没有返回-1。

输入描述

输入三行，第一行为s1，第二行为s2，第三行为k，s1和s2只包含小写字母

备注

- $0 \leq \text{len}(s1) \leq 1000000$
- $0 \leq \text{len}(s2) \leq 20000000$
- $0 \leq k \leq 1000$

输出描述

最左侧的s2以长度k冗余覆盖s1的子串首个元素下标，如果没有返回-1

示例1

输入

```
Plain Text  
1 ab  
2 aabcd  
3 1
```

输出

```
Plain Text  
1 0
```

说明

| 子串aab和abc符合要求，由于aab在abc的左侧，因此输出aab的下标：0

示例2

输入

```
Plain Text  
1 abc  
2 dfs  
3 10
```

输出

```
Plain Text  
1 -1
```

说明

| s2无法覆盖s1，输出 -1

题解

思路：题解涉及算法 滑动窗口，窗口大小为 `s1.length + k`

- 使用`remainingMatches` 保存待匹配的字符数量。使用 `patternFreq` 统计s1中各个字母的出现的

数量。使用 `windowFreq` 统计s2位于窗口中各个字母的出现的数量。

- 具体执行过程一下：当窗口右移时
 - 右边界右移(`right++`)，增加 `windowFreq[rightChar - 'a']++`；当 `windowFreq[rightChar - 'a'] <= patternFreq[rightChar - 'a']` 时，`remainingMatches--`。
 - 左边界右移之前，判断 `windowFreq[leftChar - 'a'] <= patternFreq[leftChar - 'a']` 如果成立，`remainingMatches++`，`windowFreq[leftChar - 'a']--`，`left++`
- 在窗口移动过程，存在 `remainingMatches = 0` 时，既满足条件，输出这时的`left`既是结果。

C++

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4
5 using namespace std;
6
7 int main() {
8     string pattern, text;
9     getline(cin, pattern); // 读取模式串
10    getline(cin, text);   // 读取目标文本
11    int maxExtraChars;
12    cin >> maxExtraChars; // 允许的额外字符数
13
14    int startIndex = -1;
15    vector<int> patternFreq(26, 0); // 记录 pattern 中每个字符的出现次数
16    for (char c : pattern) {
17        patternFreq[c - 'a']++;
18    }
19
20    int left = 0, right = 0;
21    // 待需要匹配字符数量
22    int remainingMatches = pattern.length();
23    vector<int> windowFreq(26, 0); // 记录滑动窗口中的字符出现次数
24
25    while (right < text.length()) {
26        char rightChar = text[right];
27        windowFreq[rightChar - 'a']++;
28
29        if (windowFreq[rightChar - 'a'] <= patternFreq[rightChar - 'a']) {
30            remainingMatches--;
31        }
32
33        if (right - left + 1 > pattern.length() + maxExtraChars) {
34            char leftChar = text[left];
35            if (windowFreq[leftChar - 'a'] <= patternFreq[leftChar - 'a']) {
36                remainingMatches++;
37            }
38            windowFreq[leftChar - 'a']--;
39            left++;
40        }
41
42        if (remainingMatches == 0) {
43            startIndex = left;
44            break;
45    }
46}
```

```
45         }
46
47         right++;
48     }
49
50     cout << startIndex;
51
52 }
```

Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取模式串
8         String pattern = scanner.nextLine();
9         // 读取目标文本
10        String text = scanner.nextLine();
11        // 允许的额外字符数
12        int maxExtraChars = scanner.nextInt();
13        scanner.close();
14
15        int startIndex = -1;
16        int[] patternFreq = new int[26]; // 记录 pattern 中每个字符的出现次数
17        for (char c : pattern.toCharArray()) {
18            patternFreq[c - 'a']++;
19        }
20
21        int left = 0, right = 0;
22        int remainingMatches = pattern.length(); // 待匹配字符数量
23        int[] windowFreq = new int[26]; // 记录滑动窗口中的字符出现次数
24
25        while (right < text.length()) {
26            char rightChar = text.charAt(right);
27            windowFreq[rightChar - 'a']++;
28
29            if (windowFreq[rightChar - 'a'] <= patternFreq[rightChar - 'a']) {
30                remainingMatches--;
31            }
32
33            if (right - left + 1 > pattern.length() + maxExtraChars) {
34                char leftChar = text.charAt(left);
35                if (windowFreq[leftChar - 'a'] <= patternFreq[leftChar - 'a']) {
36                    remainingMatches++;
37                }
38                windowFreq[leftChar - 'a']--;
39                left++;
40            }
41
42            if (remainingMatches == 0) {
43                startIndex = left;
```

```
44             break;
45         }
46
47         right++;
48     }
49
50     System.out.println(startIndex);
51 }
52 }
```

Python

```
1 import sys
2
3 # 读取模式串
4 pattern = sys.stdin.readline().strip()
5 # 读取目标文本
6 text = sys.stdin.readline().strip()
7 # 允许的额外字符数
8 max_extra_chars = int(sys.stdin.readline().strip())
9
10 pattern_freq = [0] * 26 # 记录 pattern 中每个字符的出现次数
11 for c in pattern:
12     pattern_freq[ord(c) - ord('a')] += 1
13
14 left, right = 0, 0
15 remaining_matches = len(pattern) # 待匹配字符数量
16 window_freq = [0] * 26 # 记录滑动窗口中的字符出现次数
17 start_index = -1
18
19 while right < len(text):
20     right_char = text[right]
21     window_freq[ord(right_char) - ord('a')] += 1
22
23     if window_freq[ord(right_char) - ord('a')] <= pattern_freq[ord(right_char) - ord('a')]:
24         remaining_matches -= 1
25
26     if right - left + 1 > len(pattern) + max_extra_chars:
27         left_char = text[left]
28         if window_freq[ord(left_char) - ord('a')] <= pattern_freq[ord(left_char) - ord('a')]:
29             remaining_matches += 1
30             window_freq[ord(left_char) - ord('a')] -= 1
31         left += 1
32
33     if remaining_matches == 0:
34         start_index = left
35         break
36
37     right += 1
38
39 print(start_index)
```

```

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         rl.close();
13     }
14 });
15
16 rl.on('close', () => {
17     let pattern = inputLines[0]; // 读取模式串
18     let text = inputLines[1]; // 读取目标文本
19     let maxExtraChars = parseInt(inputLines[2]); // 允许的额外字符数
20
21     let patternFreq = Array(26).fill(0); // 记录 pattern 中每个字符的出现次数
22     for (let c of pattern) {
23         patternFreq[c.charCodeAt(0) - 'a'.charCodeAt(0)]++;
24     }
25
26     let left = 0, right = 0, remainingMatches = pattern.length;
27     let windowFreq = Array(26).fill(0); // 记录滑动窗口中的字符出现次数
28     let startIndex = -1;
29
30     while (right < text.length) {
31         let rightChar = text[right];
32         windowFreq[rightChar.charCodeAt(0) - 'a'.charCodeAt(0)]++;
33
34         if (windowFreq[rightChar.charCodeAt(0) - 'a'.charCodeAt(0)] <= patternFreq[rightChar.charCodeAt(0) - 'a'.charCodeAt(0)]) {
35             remainingMatches--;
36         }
37
38         if (right - left + 1 > pattern.length + maxExtraChars) {
39             let leftChar = text[left];
40             if (windowFreq[leftChar.charCodeAt(0) - 'a'.charCodeAt(0)] <= patternFreq[leftChar.charCodeAt(0) - 'a'.charCodeAt(0)]) {
41                 remainingMatches++;
42             }
43             windowFreq[leftChar.charCodeAt(0) - 'a'.charCodeAt(0)]--;
44         }
45     }
46 }
47
48 rl.on('close', () => {
49     console.log(`Remaining matches: ${remainingMatches}`);
50 });
51
52 
```

```
44         left++;
45     }
46
47     if (remainingMatches === 0) {
48         startIndex = left;
49         break;
50     }
51
52     right++;
53 }
54
55 console.log(startIndex);
56});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8 )
9
10 func main() {
11     // 使用 bufio.Scanner 读取大数据
12     scanner := bufio.NewScanner(os.Stdin)
13
14     // 读取模式串
15     scanner.Scan()
16     pattern := scanner.Text()
17
18     // 读取目标文本
19     scanner.Scan()
20     text := scanner.Text()
21
22     // 读取允许的额外字符数 k
23     scanner.Scan()
24     maxExtraChars, _ := strconv.Atoi(scanner.Text())
25
26     // 记录 pattern 中每个字符的出现次数
27     patternFreq := make([]int, 26)
28     for _, c := range pattern {
29         patternFreq[c-'a']++
30     }
31
32     left, right := 0, 0
33     remainingMatches := len(pattern) // 待匹配字符数量
34     windowFreq := make([]int, 26)    // 记录滑动窗口中的字符出现次数
35     startIndex := -1
36
37     for right < len(text) {
38         rightChar := text[right]
39         windowFreq[rightChar-'a']++
40
41         if windowFreq[rightChar-'a'] <= patternFreq[rightChar-'a'] {
42             remainingMatches--
43         }
44
45         if right-left+1 > len(pattern)+maxExtraChars {
```

```
46     leftChar := text[left]
47     if windowFreq[leftChar-'a'] <= patternFreq[leftChar-'a'] {
48         remainingMatches++
49     }
50     windowFreq[leftChar-'a']--
51     left++
52 }
53
54     if remainingMatches == 0 {
55         startIndex = left
56         break
57     }
58
59     right++
60 }
61
62     fmt.Println(startIndex)
63 }
```

| 来自: 华为OD机考 2025C卷 – 最左侧冗余覆盖子串 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考 2025C卷 – 最左侧冗余覆盖子串 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 整数编码 (C++ & Python & JAVA & JS & GO)_华为od机试 整数编码-CSDN博客

整数编码

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

实现一种整数编码方法，使得待编码的数字越小，编码后所占用的字节数越小。

编码规则如下：

1. 编码时7位一组，每个字节的低7位用于存储待编码数字的补码
2. 字节的最高位表示后续是否还有字节，置1表示后面还有更多的字节，置0表示当前字节为最后一个字节。
3. 采用小端序编码，低位和低字节放在低地址上。
4. 编码结果按16进制数的字符格式输出，小写字母需转换为大写字母

输入描述

输入的为一个字符串表示的非负整数。待编码的数字取值范围为[0, $1<<64 - 1$]

输出描述

输出一个字符串，表示整数编码的16进制码流

用例1

输入

▼	Plain Text
1 0	

输出

▼

Plain Text |

1 00

说明

输出的16进制字符，不足两位的前面补0，如00、01、02。

用例2

输入

▼

Plain Text |

1 100

输出

▼

Plain Text |

1 64

说明

100的二进制表示为0110 0100，只需要一个字节进行编码;字节的最高位置0，剩余7位存储数字100的低7位(110 0100)，所以编码后的输出为64。

用例3

输入

▼

Plain Text |

1 1000

输出

▼

Plain Text |

1 E807

说明

1000的二进制表示为0011 1110 1000，至少需要两个字节进行编码;第一个字节最高位置1，剩余的7位存储数字1000的第一个低7位(1101000)，所以第一个字节的二进制为1110 1000，即E8;第二个字节最高位置0，剩余的7位存储数字1000的第二个低7位(0000111)，所以第一个字节的二进制为0000 0111，即07;采用小端序编码，所以低字节E8输出在前，高字节07输出在后。

题解

思路：通过二进制方式采用移位、与运算的方式进行计算。

1. 小端序：低位字节放在内存的低地址，高位字节放在内存的高地址。生活中常见十进制是大端序。
2. 按7位二进制进行组编码：可以采用与操作实现，每次将数n和1111111进行与操作就能得到后7位的值(value)，然后将数n右移7位就能实现。同时字节的最高位表示后续是否还有字节，置1表示后面还有更多的字节，置0表示当前字节为最后一个字节这个判断可以直接判断后7位的值value是否和n相等？，如果相同，则说明当前是最后一个字节不进行处理，否则需要在value前置1(value + 10000000)。
3. 明白1、2之后，接下来就是对每组与之后的value转换为16进制的字符串然后进行拼接，这个就比较简单，可参照下面逻辑实现。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<math.h>
7 #include<algorithm>
8 using namespace std;
9
10 // 十进制转十六进制
11 string tranToSixty(unsigned long long value) {
12     string s = "";
13     do {
14         unsigned long long mod = value % 16;
15         if (mod < 10) {
16             char c = '0' + mod;
17             s = string(1, c) + s;
18         } else {
19             char c = 'A' + (mod - 10);
20             s = string(1, c) + s;
21         }
22
23         value = value / 16;
24     } while (value != 0);
25
26     if (s.size() != 2) {
27         s = "0" + s;
28     }
29     return s;
30 }
31
32
33 int main() {
34     unsigned long long n;
35     cin >> n;
36     //取七位的值
37     unsigned long long mod= pow(2, 7) -1;
38     // 给第8位设置为1的复制值
39     unsigned long long add= 1 << 7;
40     string res = "";
41     do {
42         unsigned long long tmp = n & mod;
43         // 代表还有未编码的值
44         if (tmp != n) {
45             tmp += add;
```

```
46      }
47      res += tranToSixty(tmp);
48      // 移除已编码的值
49      n = n >> 7;
50  } while (n != 0);
51  cout << res;
52  return 0;
53 }
```

JAVA

```
1 import java.util.Scanner;
2
3 public class Main {
4     // 十进制转十六进制
5     public static String tranToSixty(long value) {
6         StringBuilder s = new StringBuilder();
7         do {
8             long mod = value % 16;
9             if (mod < 10) {
10                 char c = (char) ('0' + mod);
11                 s.insert(0, c);
12             } else {
13                 char c = (char) ('A' + (mod - 10));
14                 s.insert(0, c);
15             }
16             value = value / 16;
17         } while (value != 0);
18
19         if (s.length() != 2) {
20             s.insert(0, "0");
21         }
22         return s.toString();
23     }
24
25     public static void main(String[] args) {
26         Scanner scanner = new Scanner(System.in);
27         long n = scanner.nextLong();
28         //取七位的值
29         long mod = (1 << 7) - 1;
30         // 给第8位设置为1的复制值
31         long add = 1 << 7;
32         StringBuilder res = new StringBuilder();
33         do {
34             long tmp = n & mod;
35             // 代表还有未编码的值
36             if (tmp != n) {
37                 tmp += add;
38             }
39             res.append(tranToSixty(tmp));
40             // 移除已编码的值
41             n = n >> 7;
42         } while (n != 0);
43         System.out.println(res.toString());
44     }
45 }
```

Python

Plain Text |

```
1 import math
2
3 # 十进制转十六进制
4 def tran_to_sixty(value):
5     s = ""
6     while value != 0:
7         mod = value % 16
8         if mod < 10:
9             s = chr(ord('0') + mod) + s
10        else:
11            s = chr(ord('A') + (mod - 10)) + s
12        value = value // 16
13    if len(s) != 2:
14        s = "0" + s
15    return s
16
17 if __name__ == "__main__":
18     n = int(input())
19     # 取七位的值
20     mod = 2**7 - 1
21     # 给第8位设置为1的复制值
22     add = 1 << 7
23     res = ""
24     while n != 0:
25         tmp = n & mod
26         # 代表还有未编码的值
27         if tmp != n:
28             tmp += add
29         res += tran_to_sixty(tmp)
30         # 移除已编码的值
31         n = n >> 7
32     print(res)
```

JavaScript

```
1 const readline = require("readline");
2
3 // 创建 readline 接口
4 const rl = readline.createInterface({
5   input: process.stdin,
6   output: process.stdout,
7 });
8
9 // 监听每一行输入
10 rl.on("line", (input) => {
11   console.log(encodeToHex(input.trim())); // 去除输入两端空格后处理
12 });
13
14 // 主函数: 将十进制输入转为编码后的十六进制字符串
15 function encodeToHex(decimalStr) {
16   const binaryStr = BigInt(decimalStr).toString(2); // 转为二进制字符串
17
18   const hexChunks = [];
19   let remainingLength = binaryStr.length;
20
21   // 每7位分组处理, 前缀加1
22   while (remainingLength > 7) {
23     const chunk = "1" + binaryStr.slice(remainingLength - 7, remainingLength);
24     hexChunks.push(binaryToHex(chunk));
25     remainingLength -= 7;
26   }
27
28   // 处理剩余的部分
29   if (remainingLength > 0) {
30     hexChunks.push(binaryToHex(binaryStr.slice(0, remainingLength)));
31   }
32
33   return hexChunks.join(""); // 返回拼接后的十六进制字符串
34 }
35
36 // 辅助函数: 将二进制字符串转为十六进制并补齐两位
37 function binaryToHex(binaryStr) {
38   let hexStr = parseInt(binaryStr, 2).toString(16); // 转换为十六进制字符串
39   if (hexStr.length === 1) {
40     hexStr = "0" + hexStr; // 如果长度为1, 前面补零
41   }
42   return hexStr.toUpperCase(); // 返回大写的十六进制字符串
43 }
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 // 十进制转十六进制
9 func tranToSixty(value uint64) string {
10    s := ""
11    for {
12        mod := value % 16
13        if mod < 10 {
14            c := '0' + mod
15            s = string(c) + s
16        } else {
17            c := 'A' + (mod - 10)
18            s = string(c) + s
19        }
20
21        value /= 16
22        if value == 0 {
23            break
24        }
25    }
26
27    if len(s) != 2 {
28        s = "0" + s
29    }
30    return s
31 }
32
33 func main() {
34     var n uint64
35     fmt.Scan(&n)
36
37     // 取七位的值
38     sevenBitMask := uint64(math.Pow(2, 7) - 1)
39     // 给第8位设置为1的复制值
40     eighthBitFlag := uint64(1 << 7)
41     res := ""
42
43     for {
44         tmp := n & sevenBitMask
45         // 代表还有未编码的值
```

```
46     if tmp != n {
47         tmp += eighthBitFlag
48     }
49     res += tranToSixty(tmp)
50     // 移除已编码的值
51     n >>= 7
52     if n == 0 {
53         break
54     }
55 }
56
57 fmt.Println(res)
58 }
```

来自: 华为OD机考 2025C卷 – 整数编码 (C++ & Python & JAVA & JS & GO)_华为od机试 整数编码–CSDN博客

华为OD机试 2025C卷 - 单核CPU任务调度 (C++ & Python & JAVA & JS & GO)-CSDN博客

单核CPU任务调度

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 200分题型

题目描述

现有一个CPU和一些任务需要处理，已提前获知每个任务的任务ID、优先级、所需执行时间和到达时间。CPU同时只能运行一个任务，请编写一个任务调度程序，采用“可抢占优先权调度”[调度算法](#)进行任务调度，规则如下：

- 如果一个任务到来时，CPU是空闲的，则CPU可以运行该任务直到任务执行完毕。但是如果运行中有一个更高优先级的任务到来，则CPU必须暂停当前任务去运行这个优先级更高的任务；
- 如果一个任务到来时，CPU正在运行一个比他优先级更高的任务时，新到达的任务必须等待；
- 当CPU空闲时，如果还有任务在等待，CPU会从这些任务中选择一个优先级最高的任务执行，相同优先级的任务选择到达时间最早的任务。

输入描述

输入有若干行，每一行有四个数字（均小于 10^8 ），分别为任务ID，任务优先级，执行时间和到达时间。每个任务的任务ID不同，优先级数字越大优先级越高，并且相同优先级的任务不会同时到达。输入的任务已按照到达时间从小到大排序，并且保证在任何时间，处于等待的任务不超过10000个。

输出描述

按照任务执行结束的顺序，输出每个任务的任务ID和对应的结束时间。

用例1

输入

```
1 1 3 5 1
2 2 1 5 10
3 3 2 7 12
4 4 3 2 20
5 5 4 9 21
6 6 4 2 22
```

Plain Text

输出

```
Plain Text |  
1 1 6  
2 3 19  
3 5 30  
4 6 32  
5 4 33  
6 2 35
```

题解

思路： 模拟 + 优先队列

1. 结合时间线 + 优先队列来处理这个问题。
2. 定义 优先队列 排序规则按照题目要求 优先级高的任务位置堆顶，优先级相同的到达时间早的位于堆顶 规则进行排序，用优先队列来 处理当前时间和下一个任务到达之前哪个优先任务执行 的问题。
3. 记录 当前时间 主要是计算下一个任务到达时间之间有多少时间可供当前队列中的任务执行。
4. 讲一下基本逻辑：
 - a. 假设当前时间为 `currentTime`，下一个任务到达时间为 `nextArriveTime`，这中间 `idleTime = nextArriveTime - currentTime` 这段时间，就是可供优先队列中优先级最高任务执行时间。假设当前优先级最高任务为 `currentTask` 处理情况如下： 下一个任务到达之前不用考虑什么下一个任务优先级问题
 - i. `currentTask` 的剩余执行时间小于等于 `idleTime`，说明 `currentTask` 可以在这个时间段内执行结束且空闲时间还有剩余。更新 `idleTime -= currentTask 执行时间`，并更新 `currentTask` 为下一个优先级最高的任务。 注意：这是一个循环迭代的过程
 - ii. `currentTask` 的剩余执行时间大于 `idleTime`，不能在这个空闲结束，但是可以减少 `currentTask` 的剩余执行时间 `currentTask 剩余执行时间 -= idleTime`。
 - b. 当 `currentTime = nextArriveTime` 时，将下一个任务压入堆中即可。重复1 2逻辑即可，所有任务都压入优先队列结束第一阶段处理。 抢占问题 由优先队列统一去进行处理
 - c. 当前所有任务都已经压入优先队列，要不在过程已经完成，要不然任务都存在优先队列。统一处理优先队列剩余任务即可。

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<queue>
10 using namespace std;
11
12 struct Task{
13     int id;
14     int priority;
15     int executeTime;
16     int arriveTime;
17
18     Task(int id, int priority, int executeTime, int arriveTime):id(id),priority(priority),executeTime(executeTime),arriveTime(arriveTime) {}
19 };
20
21
22
23 // 自定义比较函数:
24 struct CompareTask {
25     bool operator()(const Task& a, const Task& b) const {
26         if (a.priority == b.priority) {
27             return a.arriveTime > b.arriveTime; // 到达时间 小的优先
28         }
29         return a.priority < b.priority; // priority 大的优先
30     }
31 };
32
33
34 // 通用 切割函数 函数 将字符串str根据delimiter进行切割
35 vector<int> split(const string& str, const string& delimiter) {
36     vector<int> result;
37     size_t start = 0;
38     size_t end = str.find(delimiter);
39     while (end != string::npos) {
40         result.push_back(stoi(str.substr(start, end - start)));
41         start = end + delimiter.length();
42         end = str.find(delimiter, start);
43     }
44     // 添加最后一个部分
```

```

45     result.push_back(stoi(str.substr(start)));
46     return result;
47 }
48
49 int main() {
50     vector<Task> tasks;
51     string input;
52     while (getline(cin, input)) {
53         if (input.empty()) {
54             break;
55         }
56         vector<int> arr = split(input, " ");
57         tasks.push_back({arr[0], arr[1], arr[2], arr[3]});
58     }
59     // 使用优先队列
60     priority_queue<Task, vector<Task>, CompareTask> pq;
61     int n = tasks.size();
62     int index = 0;
63     pq.push(tasks[index]);
64     index++;
65
66     int currentTime = pq.top().arriveTime;
67     while (index < n) {
68         Task currentTask = pq.top();
69         pq.pop();
70         Task nextTask = tasks[index];
71         index++;
72
73         int currentTaskEndTime = currentTime + currentTask.executeTime;
74         // 当前任务执行结束时间 大于下一个任务到达时间，更新剩余执行时间。优先级抢占
75         等问题统一让优先队列处理
76         if (currentTaskEndTime > nextTask.arriveTime) {
77             currentTask.executeTime -= nextTask.arriveTime - currentTime;
78             currentTime = nextTask.arriveTime;
79             pq.push(currentTask);
80         // 当前栈顶元素可以执行完成
81     } else {
82         cout << currentTask.id << " " << currentTaskEndTime << endl;
83         currentTime = currentTaskEndTime;
84         // 当前执行任务执行完成时间和下一个任务到达时间之前空闲时间，此时优先队
85         列中存在的任务就是等待任务，让其中优先级最高执行，充分利用空闲时间
86         if (nextTask.arriveTime > currentTime) {
87             while (!pq.empty()) {
88                 Task idleTask = pq.top();
89                 pq.pop();
90                 int idleTaskEndTime = currentTime + idleTask.execute
91                 Time;
92             // 同上

```

```
90                     if (idleTaskEndTime > nextTask.arriveTime) {
91                         idleTask.executeTime -= nextTask.arriveTime - cur-
92                         rentTime;
93                         pq.push(idleTask);
94                         break;
95                     // 同上
96                     } else {
97                         cout << idleTask.id << " " << idleTaskEndTime <<
98                         endl;
99                         currentTime = idleTaskEndTime;
100                        }
101                    }
102                }
103                // 入队
104                pq.push(nextTask);
105            }
106            // 剩余未执行任务都在队列中,不存在抢占问题,直接按照优先队列安排输出即可
107            while (!pq.empty()) {
108                Task currentTask = pq.top();
109                pq.pop();
110                int currentTaskEndTime = currentTime + currentTask.executeTime;
111                cout << currentTask.id << " " << currentTaskEndTime << endl;
112                currentTime = currentTaskEndTime;
113            }
114        }
115    }
```

JAVA

```
1 import java.util.*;
2
3 class Task {
4     int id, priority, executeTime, arriveTime;
5
6     Task(int id, int priority, int executeTime, int arriveTime) {
7         this.id = id;
8         this.priority = priority;
9         this.executeTime = executeTime;
10        this.arriveTime = arriveTime;
11    }
12 }
13
14 // 自定义比较器
15 class CompareTask implements Comparator<Task> {
16     public int compare(Task a, Task b) {
17         if (a.priority == b.priority) {
18             return Integer.compare(a.arriveTime, b.arriveTime); // 到达时间
19             小的优先
20         }
21         return Integer.compare(b.priority, a.priority); // 优先级大的优先
22     }
23 }
24
25 public class Main {
26     public static void main(String[] args) {
27         Scanner sc = new Scanner(System.in);
28         List<Task> tasks = new ArrayList<>();
29         // 接收输入
30         while (sc.hasNextLine()) {
31             String line = sc.nextLine().trim();
32             if (line.isEmpty()) break;
33             String[] parts = line.split(" ");
34             tasks.add(new Task(Integer.parseInt(parts[0]), Integer.parseInt(parts[1]),
35                               Integer.parseInt(parts[2]), Integer.parseInt(parts[3])));
36         }
37
38         PriorityQueue<Task> pq = new PriorityQueue<>(new CompareTask());
39         int n = tasks.size(), index = 0;
40
41         pq.add(tasks.get(index++));
42         int currentTime = pq.peek().arriveTime;
```

```

43     while (index < n) {
44         // 当前优先级最高任务
45         Task currentTask = pq.poll();
46         Task nextTask = tasks.get(index++);
47         int currentTaskEndTime = currentTime + currentTask.executeTim
48         e;
49
50         // 当前任务执行结束时间 > 下一个任务到达时间
51         if (currentTaskEndTime > nextTask.arriveTime) {
52             currentTask.executeTime -= (nextTask.arriveTime - currentT
53             ime);
54             currentTime = nextTask.arriveTime;
55             pq.add(currentTask);
56         } else {
57             // 当前任务可以执行完成
58             System.out.println(currentTask.id + " " + currentTaskEndTi
59             me);
56             currentTime = currentTaskEndTime;
60
61             // 如果当前时间 < 下一个任务到达时间, 执行等待队列中的任务
62             while (!pq.isEmpty() && nextTask.arriveTime > currentTim
63             e) {
64                 Task idleTask = pq.poll();
65                 int idleEnd = currentTime + idleTask.executeTime;
66                 if (idleEnd > nextTask.arriveTime) {
67                     idleTask.executeTime -= (nextTask.arriveTime - cur
68                     rentTime);
69                     pq.add(idleTask);
70                     break;
71                 } else {
72                     System.out.println(idleTask.id + " " + idleEnd);
73                     currentTime = idleEnd;
74                 }
75             }
76             currentTime = nextTask.arriveTime;
77         }
78
79         pq.add(nextTask);
80     }
81
82     // 处理剩余任务
83     while (!pq.isEmpty()) {
84         Task t = pq.poll();
85         int end = currentTime + t.executeTime;
86         System.out.println(t.id + " " + end);
87         currentTime = end;
88     }
89 }

```

Python

```

1 import sys
2 import heapq
3
4 # 定义任务类
5 class Task:
6     def __init__(self, id, priority, execute_time, arrive_time):
7         self.id = id
8         self.priority = priority
9         self.execute_time = execute_time
10        self.arrive_time = arrive_time
11
12    # 优先队列比较: 优先级高的先来, 若相同则到达时间小的优先
13    def __lt__(self, other):
14        if self.priority == other.priority:
15            return self.arrive_time < other.arrive_time
16        return self.priority > other.priority # priority 大的优先
17
18 tasks = []
19
20 # 读入任务信息
21 for line in sys.stdin:
22     if line.strip() == "":
23         break
24     arr = list(map(int, line.strip().split()))
25     tasks.append(Task(*arr))
26
27 pq = [] # 优先队列
28 index = 0
29 heapq.heappush(pq, tasks[index]) # 将第一个任务入队
30 index += 1
31 current_time = pq[0].arrive_time # 当前时间从第一个任务到达时间开始
32
33 while index < len(tasks):
34     current_task = heapq.heappop(pq) # 当前执行任务
35     next_task = tasks[index] # 下一个任务
36     index += 1
37
38     current_task_end_time = current_time + current_task.execute_time
39
40     # 当前任务执行结束时间 大于下一个任务到达时间,
41     if current_task_end_time > next_task.arrive_time:
42         current_task.execute_time -= (next_task.arrive_time - current_time)
43         current_time = next_task.arrive_time
44         heapq.heappush(pq, current_task) # 任务未执行完, 重新入队

```

```
45     else:
46         # 当前任务可以完全执行完成
47         print(current_task.id, current_task_end_time)
48         current_time = current_task_end_time
49
50     # 若当前任务完成时间 < 下一个任务到达时间, 可空闲执行其他任务
51     while pq and next_task.arrive_time > current_time:
52         idle_task = heapq.heappop(pq)
53         idle_end = current_time + idle_task.execute_time
54         if idle_end > next_task.arrive_time:
55             idle_task.execute_time -= (next_task.arrive_time - current
56             _time)
57             heapq.heappush(pq, idle_task)
58             break
59         else:
60             print(idle_task.id, idle_end)
61             current_time = idle_end
62
63         current_time = next_task.arrive_time
64
65     # 新任务入队
66     heapq.heappush(pq, next_task)
67
68 # 处理剩余未执行的任务 (无抢占问题)
69 while pq:
70     t = heapq.heappop(pq)
71     end = current_time + t.execute_time
72     print(t.id, end)
73     current_time = end
```

JavaScript

```
1 const readline = require('readline');
2
3 class Task {
4     constructor(id, priority, executeTime, arriveTime) {
5         this.id = id;
6         this.priority = priority;
7         this.executeTime = executeTime;
8         this.arriveTime = arriveTime;
9     }
10 }
11
12 // 自定义最大堆 (priority大 > arriveTime\)
13 class PriorityQueue {
14     constructor() {
15         this.heap = [];
16     }
17
18     compare(a, b) {
19         if (a.priority === b.priority) {
20             return b.id - a.id; // id 小的优先
21         }
22         return a.priority - b.priority; // priority 大的优先
23     }
24
25     push(task) {
26         this.heap.push(task);
27         this._siftUp(this.heap.length - 1);
28     }
29
30     pop() {
31         if (this.isEmpty()) return null;
32         const top = this.heap[0];
33         const end = this.heap.pop();
34         if (this.heap.length) {
35             this.heap[0] = end;
36             this._siftDown(0);
37         }
38         return top;
39     }
40
41     peek() {
42         return this.heap[0];
43     }
44
45     isEmpty() {
```

```

46         return this.heap.length === 0;
47     }
48
49     _siftUp(i) {
50         while (i > 0) {
51             const p = Math.floor((i - 1) / 2);
52             if (this.compare(this.heap[i], this.heap[p]) > 0) {
53                 [this.heap[i], this.heap[p]] = [this.heap[p], this.heap
54 [i]];
55                 i = p;
56             } else break;
57         }
58     }
59
60     _siftDown(i) {
61         const n = this.heap.length;
62         while (true) {
63             let largest = i;
64             const l = 2 * i + 1;
65             const r = 2 * i + 2;
66             if (l < n && this.compare(this.heap[l], this.heap[largest])
67 > 0) largest = l;
68             if (r < n && this.compare(this.heap[r], this.heap[largest])
69 > 0) largest = r;
70             if (largest !== i) {
71                 [this.heap[i], this.heap[largest]] = [this.heap[larges
72 t], this.heap[i]];
73                 i = largest;
74             } else break;
75         }
76     }
77 }
78
79 const rl = readline.createInterface({
80     input: process.stdin,
81     output: process.stdout
82 });
83
84 const tasks = [];
85
86 rl.on('line', (line) => {
87     if (line.trim() === '') {
88         rl.close();
89         return;
90     }
91     const [id, p, t, a] = line.trim().split(' ').map(Number);
92     tasks.push(new Task(id, p, t, a));
93 });

```

```

90
91     rl.on('close', () => {
92         const pq = new PriorityQueue();
93         let index = 0;
94         pq.push(tasks[index++]);
95         let currentTime = pq.peek().arriveTime;
96
97         while (index < tasks.length) {
98             const currentTask = pq.pop();
99             const nextTask = tasks[index++];
100
101            const endTime = currentTime + currentTask.executeTime;
102
103            // 当前任务不能执行完，下一个任务先到
104            if (endTime > nextTask.arriveTime) {
105                currentTask.executeTime -= (nextTask.arriveTime - currentTime);
106
107                currentTime = nextTask.arriveTime;
108                pq.push(currentTask); // 放回队列
109            } else {
110                // 当前任务可以完整执行
111                console.log(`#${currentTask.id} ${endTime}`);
112                currentTime = endTime;
113
114                // 执行等待队列中优先级高的任务，直到新任务到达
115                while (!pq.isEmpty() && currentTime < nextTask.arriveTime) {
116                    const idleTask = pq.pop();
117                    const idleEnd = currentTime + idleTask.executeTime;
118                    if (idleEnd > nextTask.arriveTime) {
119                        idleTask.executeTime -= (nextTask.arriveTime - currentTime);
120
121                        pq.push(idleTask);
122                        break;
123                    } else {
124                        console.log(`#${idleTask.id} ${idleEnd}`);
125                        currentTime = idleEnd;
126                    }
127
128                }
129
130                pq.push(nextTask);
131            }
132
133            // 队列中剩下的任务无抢占，直接执行
134            while (!pq.isEmpty()) {
135                const task = pq.pop();

```

```
136         const end = currentTime + task.executeTime;
137         console.log(`#${task.id} ${end}`);
138         currentTime = end;
139     }
140 );
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8     "strconv"
9     "strings"
10    )
11
12 // 定义任务结构体
13 type Task struct {
14     id         int
15     priority   int
16     executeTime int
17     arriveTime int
18 }
19
20 // 优先队列实现, 基于 container/heap
21 type TaskPQ []*Task
22
23 func (pq TaskPQ) Len() int { return len(pq) }
24
25 // 优先队列排序规则:
26 // priority 大的优先, priority 相等时 id 小的优先
27 func (pq TaskPQ) Less(i, j int) bool {
28     if pq[i].priority == pq[j].priority {
29         return pq[i].id < pq[j].id
30     }
31     return pq[i].priority > pq[j].priority
32 }
33
34 func (pq TaskPQ) Swap(i, j int) {
35     pq[i], pq[j] = pq[j], pq[i]
36 }
37
38 func (pq *TaskPQ) Push(x interface{}) {
39     *pq = append(*pq, x.(*Task))
40 }
41
42 func (pq *TaskPQ) Pop() interface{} {
43     old := *pq
44     n := len(old)
45     item := old[n-1]
```

```
46     *pq = old[:n-1]
47     return item
48 }
49
50 func main() {
51     reader := bufio.NewReader(os.Stdin)
52     var tasks []*Task
53
54     // 读入任务，空行结束
55     for {
56         line, err := reader.ReadString('\n')
57         if err != nil {
58             break
59         }
60         line = strings.TrimSpace(line)
61         if line == "" {
62             break
63         }
64         parts := strings.Fields(line)
65         if len(parts) < 4 {
66             continue
67         }
68         id, _ := strconv.Atoi(parts[0])
69         p, _ := strconv.Atoi(parts[1])
70         t, _ := strconv.Atoi(parts[2])
71         a, _ := strconv.Atoi(parts[3])
72         tasks = append(tasks, &Task{id: id, priority: p, executeTime: t,
73                               arriveTime: a})
74     }
75
76     pq := &TaskPQ{}
77     heap.Init(pq)
78
79     // 入队第一个任务
80     index := 0
81     heap.Push(pq, tasks[index])
82     index++
83     currentTime := (*pq)[0].arriveTime
84
85     // 处理所有任务
86     for index < len(tasks) {
87         currentTask := heap.Pop(pq).(*Task)
88         nextTask := tasks[index]
89         index++
90
91         currentTaskEndTime := currentTime + currentTask.executeTime
92
93         // 当前任务执行结束时间 大于 下一个任务到达时间
```

```

93     if currentTaskEndTime > nextTask.arriveTime {
94         // 扣除已执行时间
95         currentTask.executeTime -= (nextTask.arriveTime - currentTim
96     e)
97         currentTime = nextTask.arriveTime
98         heap.Push(pq, currentTask) // 重新入队剩余任务
99     } else {
100        // 当前任务执行完成
101        fmt.Println(currentTask.id, currentTaskEndTime)
102        currentTime = currentTaskEndTime
103
104        // 利用空闲时间执行等待队列中的任务
105        for pq.Len() > 0 && currentTime < nextTask.arriveTime {
106            idleTask := heap.Pop(pq).(*Task)
107            idleTaskEndTime := currentTime + idleTask.executeTime
108
109            if idleTaskEndTime > nextTask.arriveTime {
110                // 执行一部分, 剩余继续入队
111                idleTask.executeTime -= (nextTask.arriveTime - curren
112                tTime)
113                heap.Push(pq, idleTask)
114                break
115            } else {
116                // 执行完当前任务
117                fmt.Println(idleTask.id, idleTaskEndTime)
118                currentTime = idleTaskEndTime
119            }
120
121            currentTime = nextTask.arriveTime
122        }
123
124        // 新任务入队
125        heap.Push(pq, nextTask)
126    }
127
128    // 处理剩余任务 (无抢占)
129    for pq.Len() > 0 {
130        t := heap.Pop(pq).(*Task)
131        endTime := currentTime + t.executeTime
132        fmt.Println(t.id, endTime)
133        currentTime = endTime
134    }
}

```

单核CPU任务调度

华为OD机试 2025C卷 200分题型

题目描述

现有一个CPU和一些任务需要处理，已提前获知每个任务的任务ID、优先级、所需执行时间和到达时间。CPU同时只能运行一个任务，请编写一个任务调度程序，采用“可抢占优先权调度”[调度算法](#)进行任务调度，规则如下：

- 如果一个任务到来时，CPU是空闲的，则CPU可以运行该任务直到任务执行完毕。但是如果运行中有一个更高优先级的任务到来，则CPU必须暂停当前任务去运行这个优先级更高的任务；
- 如果一个任务到来时，CPU正在运行一个比他优先级更高的任务时，新到达的任务必须等待；
- 当CPU空闲时，如果还有任务在等待，CPU会从这些任务中选择一个优先级最高的任务执行，相同优先级的任务选择到达时间最早的任务。

输入描述

输入有若干行，每一行有四个数字（均小于 10^8 ），分别为任务ID，任务优先级，执行时间和到达时间。每个任务的任务ID不同，优先级数字越大优先级越高，并且相同优先级的任务不会同时到达。输入的任务已按照到达时间从小到大排序，并且保证在任何时间，处于等待的任务不超过10000个。

输出描述

按照任务执行结束的顺序，输出每个任务的任务ID和对应的结束时间。

用例1

输入

```
1 1 3 5 1
2 2 1 5 10
3 3 2 7 12
4 4 3 2 20
5 5 4 9 21
6 6 4 2 22
```

输出

▼

Plain Text |

```
1 1 6
2 3 19
3 5 30
4 6 32
5 4 33
6 2 35
```

题解

思路： 模拟 + 优先队列

1. 结合时间线 + 优先队列来处理这个问题。
2. 定义 优先队列 排序规则按照题目要求 优先级高的任务位置堆顶，优先级相同的到达时间早的位于堆顶 规则进行排序，用优先队列来 处理当前时间和下一个任务到达之前哪个优先任务执行 的问题。
3. 记录 当前时间 主要是计算下一个任务到达时间之间有多少时间可供当前队列中的任务执行。
4. 讲一下基本逻辑：
 - a. 假设当前时间为 `currentTime`，下一个任务到达时间为 `nextArriveTime`，这中间 `idleTime = nextArriveTime - currentTime` 这段时间，就是可供优先队列中优先级最高任务执行时间。假设当前优先级最高任务为 `currentTask` 处理情况如下： 下一个任务到达之前不用考虑什么下一个任务优先级问题
 - i. `currentTask` 的剩余执行时间小于等于 `idleTime`，说明 `currentTask` 可以在这个时间段内执行结束且空闲时间还有剩余。更新 `idleTime -= currentTask` 执行时间，并更新 `currentTask` 为下一个优先级最高的任务。 注意：这是一个循环迭代的过程
 - ii. `currentTask` 的剩余执行时间大于 `idleTime`，不能在这个空闲结束，但是可以减少 `currentTask` 的剩余执行时间 `currentTask剩余执行时间 -= idleTime`。
 - b. 当 `currentTime = nextArriveTime` 时，将下一个任务压入堆中即可。重复1 2逻辑即可，所有任务都压入优先队列结束第一阶段处理。 抢占问题 由优先队列统一去进行处理
 - c. 当前所有任务都已经压入优先队列，要不在过程已经完成，要不然任务都存在优先队列。统一处理优先队列剩余任务即可。

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<queue>
10 using namespace std;
11
12 struct Task{
13     int id;
14     int priority;
15     int executeTime;
16     int arriveTime;
17
18     Task(int id, int priority, int executeTime, int arriveTime):id(id),priority(priority),executeTime(executeTime),arriveTime(arriveTime) {}
19 };
20
21
22
23 // 自定义比较函数:
24 struct CompareTask {
25     bool operator()(const Task& a, const Task& b) const {
26         if (a.priority == b.priority) {
27             return a.arriveTime > b.arriveTime; // 到达时间 小的优先
28         }
29         return a.priority < b.priority; // priority 大的优先
30     }
31 };
32
33
34 // 通用 切割函数 函数 将字符串str根据delimiter进行切割
35 vector<int> split(const string& str, const string& delimiter) {
36     vector<int> result;
37     size_t start = 0;
38     size_t end = str.find(delimiter);
39     while (end != string::npos) {
40         result.push_back(stoi(str.substr(start, end - start)));
41         start = end + delimiter.length();
42         end = str.find(delimiter, start);
43     }
44     // 添加最后一个部分
```

```

45     result.push_back(stoi(str.substr(start)));
46     return result;
47 }
48
49 int main() {
50     vector<Task> tasks;
51     string input;
52     while (getline(cin, input)) {
53         if (input.empty()) {
54             break;
55         }
56         vector<int> arr = split(input, " ");
57         tasks.push_back({arr[0], arr[1], arr[2], arr[3]});
58     }
59     // 使用优先队列
60     priority_queue<Task, vector<Task>, CompareTask> pq;
61     int n = tasks.size();
62     int index = 0;
63     pq.push(tasks[index]);
64     index++;
65
66     int currentTime = pq.top().arriveTime;
67     while (index < n) {
68         Task currentTask = pq.top();
69         pq.pop();
70         Task nextTask = tasks[index];
71         index++;
72
73         int currentTaskEndTime = currentTime + currentTask.executeTime;
74         // 当前任务执行结束时间 大于下一个任务到达时间，更新剩余执行时间。优先级抢占
75         等问题统一让优先队列处理
76         if (currentTaskEndTime > nextTask.arriveTime) {
77             currentTask.executeTime -= nextTask.arriveTime - currentTime;
78             currentTime = nextTask.arriveTime;
79             pq.push(currentTask);
80         // 当前栈顶元素可以执行完成
81     } else {
82         cout << currentTask.id << " " << currentTaskEndTime << endl;
83         currentTime = currentTaskEndTime;
84         // 当前执行任务执行完成时间和下一个任务到达时间之前空闲时间，此时优先队
85         列中存在的任务就是等待任务，让其中优先级最高执行，充分利用空闲时间
86         if (nextTask.arriveTime > currentTime) {
87             while (!pq.empty()) {
88                 Task idleTask = pq.top();
89                 pq.pop();
90                 int idleTaskEndTime = currentTime + idleTask.execute
91                 Time;
92             // 同上

```

```
90                     if (idleTaskEndTime > nextTask.arriveTime) {
91                         idleTask.executeTime -= nextTask.arriveTime - cur-
92                         rentTime;
93                         pq.push(idleTask);
94                         break;
95                     // 同上
96                     } else {
97                         cout << idleTask.id << " " << idleTaskEndTime <<
98                         endl;
99                         currentTime = idleTaskEndTime;
100                        }
101                    }
102                }
103                // 入队
104                pq.push(nextTask);
105            }
106            // 剩余未执行任务都在队列中,不存在抢占问题,直接按照优先队列安排输出即可
107            while (!pq.empty()) {
108                Task currentTask = pq.top();
109                pq.pop();
110                int currentTaskEndTime = currentTime + currentTask.executeTime;
111                cout << currentTask.id << " " << currentTaskEndTime << endl;
112                currentTime = currentTaskEndTime;
113            }
114        return 0;
115    }
```

JAVA

```
1 import java.util.*;
2
3 class Task {
4     int id, priority, executeTime, arriveTime;
5
6     Task(int id, int priority, int executeTime, int arriveTime) {
7         this.id = id;
8         this.priority = priority;
9         this.executeTime = executeTime;
10        this.arriveTime = arriveTime;
11    }
12 }
13
14 // 自定义比较器
15 class CompareTask implements Comparator<Task> {
16     public int compare(Task a, Task b) {
17         if (a.priority == b.priority) {
18             return Integer.compare(a.arriveTime, b.arriveTime); // 到达时间
19             小的优先
20         }
21         return Integer.compare(b.priority, a.priority); // 优先级大的优先
22     }
23 }
24
25 public class Main {
26     public static void main(String[] args) {
27         Scanner sc = new Scanner(System.in);
28         List<Task> tasks = new ArrayList<>();
29         // 接收输入
30         while (sc.hasNextLine()) {
31             String line = sc.nextLine().trim();
32             if (line.isEmpty()) break;
33             String[] parts = line.split(" ");
34             tasks.add(new Task(Integer.parseInt(parts[0]), Integer.parseInt(parts[1]),
35                               Integer.parseInt(parts[2]), Integer.parseInt(parts[3])));
36         }
37
38         PriorityQueue<Task> pq = new PriorityQueue<>(new CompareTask());
39         int n = tasks.size(), index = 0;
40
41         pq.add(tasks.get(index++));
42         int currentTime = pq.peek().arriveTime;
```

```

43     while (index < n) {
44         // 当前优先级最高任务
45         Task currentTask = pq.poll();
46         Task nextTask = tasks.get(index++);
47         int currentTaskEndTime = currentTime + currentTask.executeTim
48         e;
49
50         // 当前任务执行结束时间 > 下一个任务到达时间
51         if (currentTaskEndTime > nextTask.arriveTime) {
52             currentTask.executeTime -= (nextTask.arriveTime - currentT
53             ime);
54             currentTime = nextTask.arriveTime;
55             pq.add(currentTask);
56         } else {
57             // 当前任务可以执行完成
58             System.out.println(currentTask.id + " " + currentTaskEndTi
59             me);
56             currentTime = currentTaskEndTime;
60
61             // 如果当前时间 < 下一个任务到达时间, 执行等待队列中的任务
62             while (!pq.isEmpty() && nextTask.arriveTime > currentTim
63             e) {
64                 Task idleTask = pq.poll();
65                 int idleEnd = currentTime + idleTask.executeTime;
66                 if (idleEnd > nextTask.arriveTime) {
67                     idleTask.executeTime -= (nextTask.arriveTime - cur
68                     rentTime);
69                     pq.add(idleTask);
70                     break;
71                 } else {
72                     System.out.println(idleTask.id + " " + idleEnd);
73                     currentTime = idleEnd;
74                 }
75             }
76             currentTime = nextTask.arriveTime;
77         }
78
79         pq.add(nextTask);
80     }
81
82     // 处理剩余任务
83     while (!pq.isEmpty()) {
84         Task t = pq.poll();
85         int end = currentTime + t.executeTime;
86         System.out.println(t.id + " " + end);
87         currentTime = end;
88     }
89 }

```

86 }

Python

```
1 import sys
2 import heapq
3
4 # 定义任务类
5 class Task:
6     def __init__(self, id, priority, execute_time, arrive_time):
7         self.id = id
8         self.priority = priority
9         self.execute_time = execute_time
10        self.arrive_time = arrive_time
11
12    # 优先队列比较: 优先级高的先来, 若相同则到达时间小的优先
13    def __lt__(self, other):
14        if self.priority == other.priority:
15            return self.arrive_time < other.arrive_time
16        return self.priority > other.priority # priority 大的优先
17
18 tasks = []
19
20 # 读入任务信息
21 for line in sys.stdin:
22     if line.strip() == "":
23         break
24     arr = list(map(int, line.strip().split()))
25     tasks.append(Task(*arr))
26
27 pq = [] # 优先队列
28 index = 0
29 heapq.heappush(pq, tasks[index]) # 将第一个任务入队
30 index += 1
31 current_time = pq[0].arrive_time # 当前时间从第一个任务到达时间开始
32
33 while index < len(tasks):
34     current_task = heapq.heappop(pq) # 当前执行任务
35     next_task = tasks[index] # 下一个任务
36     index += 1
37
38     current_task_end_time = current_time + current_task.execute_time
39
40     # 当前任务执行结束时间 大于下一个任务到达时间,
41     if current_task_end_time > next_task.arrive_time:
42         current_task.execute_time -= (next_task.arrive_time - current_time)
43         # 减少已执行部分
44         current_time = next_task.arrive_time
45         heapq.heappush(pq, current_task) # 任务未执行完, 重新入队
```

```
45     else:
46         # 当前任务可以完全执行完成
47         print(current_task.id, current_task_end_time)
48         current_time = current_task_end_time
49
50     # 若当前任务完成时间 < 下一个任务到达时间, 可空闲执行其他任务
51     while pq and next_task.arrive_time > current_time:
52         idle_task = heapq.heappop(pq)
53         idle_end = current_time + idle_task.execute_time
54         if idle_end > next_task.arrive_time:
55             idle_task.execute_time -= (next_task.arrive_time - current
56             _time)
57             heapq.heappush(pq, idle_task)
58             break
59         else:
60             print(idle_task.id, idle_end)
61             current_time = idle_end
62
63         current_time = next_task.arrive_time
64
65     # 新任务入队
66     heapq.heappush(pq, next_task)
67
68 # 处理剩余未执行的任务 (无抢占问题)
69 while pq:
70     t = heapq.heappop(pq)
71     end = current_time + t.execute_time
72     print(t.id, end)
73     current_time = end
```

JavaScript

```
1 const readline = require('readline');
2
3 class Task {
4     constructor(id, priority, executeTime, arriveTime) {
5         this.id = id;
6         this.priority = priority;
7         this.executeTime = executeTime;
8         this.arriveTime = arriveTime;
9     }
10 }
11
12 // 自定义最大堆 (priority大 > arriveTime\)
13 class PriorityQueue {
14     constructor() {
15         this.heap = [];
16     }
17
18     compare(a, b) {
19         if (a.priority === b.priority) {
20             return b.id - a.id; // id 小的优先
21         }
22         return a.priority - b.priority; // priority 大的优先
23     }
24
25     push(task) {
26         this.heap.push(task);
27         this._siftUp(this.heap.length - 1);
28     }
29
30     pop() {
31         if (this.isEmpty()) return null;
32         const top = this.heap[0];
33         const end = this.heap.pop();
34         if (this.heap.length) {
35             this.heap[0] = end;
36             this._siftDown(0);
37         }
38         return top;
39     }
40
41     peek() {
42         return this.heap[0];
43     }
44
45     isEmpty() {
```

```

46         return this.heap.length === 0;
47     }
48
49     _siftUp(i) {
50         while (i > 0) {
51             const p = Math.floor((i - 1) / 2);
52             if (this.compare(this.heap[i], this.heap[p]) > 0) {
53                 [this.heap[i], this.heap[p]] = [this.heap[p], this.heap
54 [i]];
55                 i = p;
56             } else break;
57         }
58     }
59
60     _siftDown(i) {
61         const n = this.heap.length;
62         while (true) {
63             let largest = i;
64             const l = 2 * i + 1;
65             const r = 2 * i + 2;
66             if (l < n && this.compare(this.heap[l], this.heap[largest])
67 > 0) largest = l;
68             if (r < n && this.compare(this.heap[r], this.heap[largest])
69 > 0) largest = r;
70             if (largest !== i) {
71                 [this.heap[i], this.heap[largest]] = [this.heap[larges
72 t], this.heap[i]];
73                 i = largest;
74             } else break;
75         }
76     }
77 }
78
79 const rl = readline.createInterface({
80     input: process.stdin,
81     output: process.stdout
82 });
83
84 const tasks = [];
85
86 rl.on('line', (line) => {
87     if (line.trim() === '') {
88         rl.close();
89         return;
90     }
91     const [id, p, t, a] = line.trim().split(' ').map(Number);
92     tasks.push(new Task(id, p, t, a));
93 });

```

```

90
91   rl.on('close', () => {
92     const pq = new PriorityQueue();
93     let index = 0;
94     pq.push(tasks[index++]);
95     let currentTime = pq.peek().arriveTime;
96
97     while (index < tasks.length) {
98       const currentTask = pq.pop();
99       const nextTask = tasks[index++];
100
101      const endTime = currentTime + currentTask.executeTime;
102
103      // 当前任务不能执行完，下一个任务先到
104      if (endTime > nextTask.arriveTime) {
105        currentTask.executeTime -= (nextTask.arriveTime - currentTime);
106
107        currentTime = nextTask.arriveTime;
108        pq.push(currentTask); // 放回队列
109      } else {
110        // 当前任务可以完整执行
111        console.log(`#${currentTask.id} ${endTime}`);
112        currentTime = endTime;
113
114        // 执行等待队列中优先级高的任务，直到新任务到达
115        while (!pq.isEmpty() && currentTime < nextTask.arriveTime) {
116          const idleTask = pq.pop();
117          const idleEnd = currentTime + idleTask.executeTime;
118          if (idleEnd > nextTask.arriveTime) {
119            idleTask.executeTime -= (nextTask.arriveTime - currentTime);
120
121            pq.push(idleTask);
122            break;
123          } else {
124            console.log(`#${idleTask.id} ${idleEnd}`);
125            currentTime = idleEnd;
126          }
127
128        }
129
130        pq.push(nextTask);
131      }
132
133      // 队列中剩下的任务无抢占，直接执行
134      while (!pq.isEmpty()) {
135        const task = pq.pop();

```

```
136         const end = currentTime + task.executeTime;
137         console.log(`#${task.id} ${end}`);
138         currentTime = end;
139     }
140 );
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8     "strconv"
9     "strings"
10    )
11
12 // 定义任务结构体
13 type Task struct {
14     id         int
15     priority   int
16     executeTime int
17     arriveTime int
18 }
19
20 // 优先队列实现, 基于 container/heap
21 type TaskPQ []*Task
22
23 func (pq TaskPQ) Len() int { return len(pq) }
24
25 // 优先队列排序规则:
26 // priority 大的优先, priority 相等时 id 小的优先
27 func (pq TaskPQ) Less(i, j int) bool {
28     if pq[i].priority == pq[j].priority {
29         return pq[i].id < pq[j].id
30     }
31     return pq[i].priority > pq[j].priority
32 }
33
34 func (pq TaskPQ) Swap(i, j int) {
35     pq[i], pq[j] = pq[j], pq[i]
36 }
37
38 func (pq *TaskPQ) Push(x interface{}) {
39     *pq = append(*pq, x.(*Task))
40 }
41
42 func (pq *TaskPQ) Pop() interface{} {
43     old := *pq
44     n := len(old)
45     item := old[n-1]
```

```
46     *pq = old[:n-1]
47     return item
48 }
49
50 func main() {
51     reader := bufio.NewReader(os.Stdin)
52     var tasks []*Task
53
54     // 读入任务，空行结束
55     for {
56         line, err := reader.ReadString('\n')
57         if err != nil {
58             break
59         }
60         line = strings.TrimSpace(line)
61         if line == "" {
62             break
63         }
64         parts := strings.Fields(line)
65         if len(parts) < 4 {
66             continue
67         }
68         id, _ := strconv.Atoi(parts[0])
69         p, _ := strconv.Atoi(parts[1])
70         t, _ := strconv.Atoi(parts[2])
71         a, _ := strconv.Atoi(parts[3])
72         tasks = append(tasks, &Task{id: id, priority: p, executeTime: t,
73                               arriveTime: a})
74     }
75
76     pq := &TaskPQ{}
77     heap.Init(pq)
78
79     // 入队第一个任务
80     index := 0
81     heap.Push(pq, tasks[index])
82     index++
83     currentTime := (*pq)[0].arriveTime
84
85     // 处理所有任务
86     for index < len(tasks) {
87         currentTask := heap.Pop(pq).(*Task)
88         nextTask := tasks[index]
89         index++
90
91         currentTaskEndTime := currentTime + currentTask.executeTime
92
93         // 当前任务执行结束时间 大于 下一个任务到达时间
```

```

93     if currentTaskEndTime > nextTask.arriveTime {
94         // 扣除已执行时间
95         currentTask.executeTime -= (nextTask.arriveTime - currentTim
96         e)
97         currentTime = nextTask.arriveTime
98         heap.Push(pq, currentTask) // 重新入队剩余任务
99     } else {
100        // 当前任务执行完成
101        fmt.Println(currentTask.id, currentTaskEndTime)
102        currentTime = currentTaskEndTime
103
104        // 利用空闲时间执行等待队列中的任务
105        for pq.Len() > 0 && currentTime < nextTask.arriveTime {
106            idleTask := heap.Pop(pq).(*Task)
107            idleTaskEndTime := currentTime + idleTask.executeTime
108
109            if idleTaskEndTime > nextTask.arriveTime {
110                // 执行一部分, 剩余继续入队
111                idleTask.executeTime -= (nextTask.arriveTime - curren
112                tTime)
113                heap.Push(pq, idleTask)
114                break
115            } else {
116                // 执行完当前任务
117                fmt.Println(idleTask.id, idleTaskEndTime)
118                currentTime = idleTaskEndTime
119            }
120
121            currentTime = nextTask.arriveTime
122        }
123
124        // 新任务入队
125        heap.Push(pq, nextTask)
126    }
127
128    // 处理剩余任务 (无抢占)
129    for pq.Len() > 0 {
130        t := heap.Pop(pq).(*Task)
131        endTime := currentTime + t.executeTime
132        fmt.Println(t.id, endTime)
133        currentTime = endTime
134    }
}

```

| 来自: 华为OD机试 2025C卷 – 单核CPU任务调度 (C++ & Python & JAVA & JS & GO)-CSDN博客

单核CPU任务调度

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 200分题型

题目描述

现有一个CPU和一些任务需要处理，已提前获知每个任务的任务ID、优先级、所需执行时间和到达时间。CPU同时只能运行一个任务，请编写一个任务调度程序，采用“可抢占优先权调度”[调度算法](#)进行任务调度，规则如下：

- 如果一个任务到来时，CPU是空闲的，则CPU可以运行该任务直到任务执行完毕。但是如果运行中有一个更高优先级的任务到来，则CPU必须暂停当前任务去运行这个优先级更高的任务；
- 如果一个任务到来时，CPU正在运行一个比他优先级更高的任务时，新到达的任务必须等待；
- 当CPU空闲时，如果还有任务在等待，CPU会从这些任务中选择一个优先级最高的任务执行，相同优先级的任务选择到达时间最早的任务。

输入描述

输入有若干行，每一行有四个数字（均小于 10^8 ），分别为任务ID，任务优先级，执行时间和到达时间。

每个任务的任务ID不同，优先级数字越大优先级越高，并且相同优先级的任务不会同时到达。

输入的任务已按照到达时间从小到大排序，并且保证在任何时间，处于等待的任务不超过10000个。

输出描述

按照任务执行结束的顺序，输出每个任务的任务ID和对应的结束时间。

用例1

输入

```
▼ Plain Text |  
1 1 3 5 1  
2 2 1 5 10  
3 3 2 7 12  
4 4 3 2 20  
5 5 4 9 21  
6 6 4 2 22
```

输出

```

1 1 6
2 3 19
3 5 30
4 6 32
5 4 33
6 2 35

```

题解

思路： 模拟 + 优先队列

1. 结合时间线 + 优先队列来处理这个问题。
2. 定义 优先队列 排序规则按照题目要求 优先级高的任务位置堆顶，优先级相同的到达时间早的位于堆顶 规则进行排序，用优先队列来 处理当前时间和下一个任务到达之前哪个优先任务执行 的问题。
3. 记录 当前时间 主要是计算下一个任务到达时间之间有多少时间可供当前队列中的任务执行。
4. 讲一下基本逻辑：
 - a. 假设当前时间为 `currentTime`，下一个任务到达时间为 `nextArriveTime`，这中间 `idleTime = nextArriveTime - currentTime` 这段时间，就是可供优先队列中优先级最高任务执行时间。假设当前优先级最高任务为 `currentTask` 处理情况如下： 下一个任务到达之前不用考虑什么下一个任务优先级问题
 - i. `currentTask` 的剩余执行时间小于等于 `idleTime`，说明 `currentTask` 可以在这个时间段内执行结束且空闲时间还有剩余。更新 `idleTime -= currentTask` 执行时间，并更新 `currentTask` 为下一个优先级最高的任务。 注意：这是一个循环迭代的过程
 - ii. `currentTask` 的剩余执行时间大于 `idleTime`，不能在这个空闲结束，但是可以减少 `currentTask` 的剩余执行时间 `currentTask剩余执行时间 -= idleTime`。
 - b. 当 `currentTime = nextArriveTime` 时，将下一个任务压入堆中即可。重复1 2逻辑即可，所有任务都压入优先队列结束第一阶段处理。 抢占问题 由优先队列统一去进行处理
 - c. 当前所有任务都已经压入优先队列，要不在过程已经完成，要不然任务都存在优先队列。统一处理优先队列剩余任务即可。

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<queue>
10 using namespace std;
11
12 struct Task{
13     int id;
14     int priority;
15     int executeTime;
16     int arriveTime;
17
18     Task(int id, int priority, int executeTime, int arriveTime):id(id),priority(priority),executeTime(executeTime),arriveTime(arriveTime) {}
19 };
20
21
22
23 // 自定义比较函数:
24 struct CompareTask {
25     bool operator()(const Task& a, const Task& b) const {
26         if (a.priority == b.priority) {
27             return a.arriveTime > b.arriveTime; // 到达时间 小的优先
28         }
29         return a.priority < b.priority; // priority 大的优先
30     }
31 };
32
33
34 // 通用 切割函数 函数 将字符串str根据delimiter进行切割
35 vector<int> split(const string& str, const string& delimiter) {
36     vector<int> result;
37     size_t start = 0;
38     size_t end = str.find(delimiter);
39     while (end != string::npos) {
40         result.push_back(stoi(str.substr(start, end - start)));
41         start = end + delimiter.length();
42         end = str.find(delimiter, start);
43     }
44     // 添加最后一个部分
```

```

45     result.push_back(stoi(str.substr(start)));
46     return result;
47 }
48
49 int main() {
50     vector<Task> tasks;
51     string input;
52     while (getline(cin, input)) {
53         if (input.empty()) {
54             break;
55         }
56         vector<int> arr = split(input, " ");
57         tasks.push_back({arr[0], arr[1], arr[2], arr[3]});
58     }
59     // 使用优先队列
60     priority_queue<Task, vector<Task>, CompareTask> pq;
61     int n = tasks.size();
62     int index = 0;
63     pq.push(tasks[index]);
64     index++;
65
66     int currentTime = pq.top().arriveTime;
67     while (index < n) {
68         Task currentTask = pq.top();
69         pq.pop();
70         Task nextTask = tasks[index];
71         index++;
72
73         int currentTaskEndTime = currentTime + currentTask.executeTime;
74         // 当前任务执行结束时间 大于下一个任务到达时间，更新剩余执行时间。优先级抢占
75         等问题统一让优先队列处理
76         if (currentTaskEndTime > nextTask.arriveTime) {
77             currentTask.executeTime -= nextTask.arriveTime - currentTime;
78             currentTime = nextTask.arriveTime;
79             pq.push(currentTask);
80         // 当前栈顶元素可以执行完成
81     } else {
82         cout << currentTask.id << " " << currentTaskEndTime << endl;
83         currentTime = currentTaskEndTime;
84         // 当前执行任务执行完成时间和下一个任务到达时间之前空闲时间，此时优先队
85         列中存在的任务就是等待任务，让其中优先级最高执行，充分利用空闲时间
86         if (nextTask.arriveTime > currentTime) {
87             while (!pq.empty()) {
88                 Task idleTask = pq.top();
89                 pq.pop();
90                 int idleTaskEndTime = currentTime + idleTask.execute
91                 Time;
92             // 同上

```

```
90                     if (idleTaskEndTime > nextTask.arriveTime) {
91                         idleTask.executeTime -= nextTask.arriveTime - cur-
92                         rentTime;
93                         pq.push(idleTask);
94                         break;
95                     // 同上
96                     } else {
97                         cout << idleTask.id << " " << idleTaskEndTime <<
98                         endl;
99                         currentTime = idleTaskEndTime;
100                        }
101                    }
102                }
103                // 入队
104                pq.push(nextTask);
105            }
106            // 剩余未执行任务都在队列中,不存在抢占问题,直接按照优先队列安排输出即可
107            while (!pq.empty()) {
108                Task currentTask = pq.top();
109                pq.pop();
110                int currentTaskEndTime = currentTime + currentTask.executeTime;
111                cout << currentTask.id << " " << currentTaskEndTime << endl;
112                currentTime = currentTaskEndTime;
113            }
114        }
115    }
```

JAVA

```
1 import java.util.*;
2
3 class Task {
4     int id, priority, executeTime, arriveTime;
5
6     Task(int id, int priority, int executeTime, int arriveTime) {
7         this.id = id;
8         this.priority = priority;
9         this.executeTime = executeTime;
10        this.arriveTime = arriveTime;
11    }
12 }
13
14 // 自定义比较器
15 class CompareTask implements Comparator<Task> {
16     public int compare(Task a, Task b) {
17         if (a.priority == b.priority) {
18             return Integer.compare(a.arriveTime, b.arriveTime); // 到达时间
19             小的优先
20         }
21         return Integer.compare(b.priority, a.priority); // 优先级大的优先
22     }
23 }
24
25 public class Main {
26     public static void main(String[] args) {
27         Scanner sc = new Scanner(System.in);
28         List<Task> tasks = new ArrayList<>();
29         // 接收输入
30         while (sc.hasNextLine()) {
31             String line = sc.nextLine().trim();
32             if (line.isEmpty()) break;
33             String[] parts = line.split(" ");
34             tasks.add(new Task(Integer.parseInt(parts[0]), Integer.parseInt(parts[1]),
35                               Integer.parseInt(parts[2]), Integer.parseInt(parts[3])));
36         }
37
38         PriorityQueue<Task> pq = new PriorityQueue<>(new CompareTask());
39         int n = tasks.size(), index = 0;
40
41         pq.add(tasks.get(index++));
42         int currentTime = pq.peek().arriveTime;
```

```

43     while (index < n) {
44         // 当前优先级最高任务
45         Task currentTask = pq.poll();
46         Task nextTask = tasks.get(index++);
47         int currentTaskEndTime = currentTime + currentTask.executeTim
48         e;
49
50         // 当前任务执行结束时间 > 下一个任务到达时间
51         if (currentTaskEndTime > nextTask.arriveTime) {
52             currentTask.executeTime -= (nextTask.arriveTime - currentT
53             ime);
54             currentTime = nextTask.arriveTime;
55             pq.add(currentTask);
56         } else {
57             // 当前任务可以执行完成
58             System.out.println(currentTask.id + " " + currentTaskEndTi
59             me);
56             currentTime = currentTaskEndTime;
59
60             // 如果当前时间 < 下一个任务到达时间, 执行等待队列中的任务
61             while (!pq.isEmpty() && nextTask.arriveTime > currentTim
62             e) {
63                 Task idleTask = pq.poll();
64                 int idleEnd = currentTime + idleTask.executeTime;
65                 if (idleEnd > nextTask.arriveTime) {
66                     idleTask.executeTime -= (nextTask.arriveTime - cur
67                     rentTime);
68                     pq.add(idleTask);
69                     break;
70                 } else {
71                     System.out.println(idleTask.id + " " + idleEnd);
72                     currentTime = idleEnd;
73                 }
74             }
75             currentTime = nextTask.arriveTime;
76         }
77
78         pq.add(nextTask);
79     }
80
81     // 处理剩余任务
82     while (!pq.isEmpty()) {
83         Task t = pq.poll();
84         int end = currentTime + t.executeTime;
85         System.out.println(t.id + " " + end);
86         currentTime = end;
87     }

```

Python

```

1 import sys
2 import heapq
3
4 # 定义任务类
5 class Task:
6     def __init__(self, id, priority, execute_time, arrive_time):
7         self.id = id
8         self.priority = priority
9         self.execute_time = execute_time
10        self.arrive_time = arrive_time
11
12    # 优先队列比较: 优先级高的先来, 若相同则到达时间小的优先
13    def __lt__(self, other):
14        if self.priority == other.priority:
15            return self.arrive_time < other.arrive_time
16        return self.priority > other.priority # priority 大的优先
17
18 tasks = []
19
20 # 读入任务信息
21 for line in sys.stdin:
22     if line.strip() == "":
23         break
24     arr = list(map(int, line.strip().split()))
25     tasks.append(Task(*arr))
26
27 pq = [] # 优先队列
28 index = 0
29 heapq.heappush(pq, tasks[index]) # 将第一个任务入队
30 index += 1
31 current_time = pq[0].arrive_time # 当前时间从第一个任务到达时间开始
32
33 while index < len(tasks):
34     current_task = heapq.heappop(pq) # 当前执行任务
35     next_task = tasks[index] # 下一个任务
36     index += 1
37
38     current_task_end_time = current_time + current_task.execute_time
39
40     # 当前任务执行结束时间 大于下一个任务到达时间,
41     if current_task_end_time > next_task.arrive_time:
42         current_task.execute_time -= (next_task.arrive_time - current_time)
43         current_time = next_task.arrive_time
44         heapq.heappush(pq, current_task) # 任务未执行完, 重新入队

```

```
45     else:
46         # 当前任务可以完全执行完成
47         print(current_task.id, current_task_end_time)
48         current_time = current_task_end_time
49
50     # 若当前任务完成时间 < 下一个任务到达时间, 可空闲执行其他任务
51     while pq and next_task.arrive_time > current_time:
52         idle_task = heapq.heappop(pq)
53         idle_end = current_time + idle_task.execute_time
54         if idle_end > next_task.arrive_time:
55             idle_task.execute_time -= (next_task.arrive_time - current
56             _time)
57             heapq.heappush(pq, idle_task)
58             break
59         else:
60             print(idle_task.id, idle_end)
61             current_time = idle_end
62
63         current_time = next_task.arrive_time
64
65     # 新任务入队
66     heapq.heappush(pq, next_task)
67
68 # 处理剩余未执行的任务 (无抢占问题)
69 while pq:
70     t = heapq.heappop(pq)
71     end = current_time + t.execute_time
72     print(t.id, end)
73     current_time = end
```

JavaScript

```
1 const readline = require('readline');
2
3 class Task {
4     constructor(id, priority, executeTime, arriveTime) {
5         this.id = id;
6         this.priority = priority;
7         this.executeTime = executeTime;
8         this.arriveTime = arriveTime;
9     }
10 }
11
12 // 自定义最大堆 (priority大 > arriveTime\)
13 class PriorityQueue {
14     constructor() {
15         this.heap = [];
16     }
17
18     compare(a, b) {
19         if (a.priority === b.priority) {
20             return b.id - a.id; // id 小的优先
21         }
22         return a.priority - b.priority; // priority 大的优先
23     }
24
25     push(task) {
26         this.heap.push(task);
27         this._siftUp(this.heap.length - 1);
28     }
29
30     pop() {
31         if (this.isEmpty()) return null;
32         const top = this.heap[0];
33         const end = this.heap.pop();
34         if (this.heap.length) {
35             this.heap[0] = end;
36             this._siftDown(0);
37         }
38         return top;
39     }
40
41     peek() {
42         return this.heap[0];
43     }
44
45     isEmpty() {
```

```

46         return this.heap.length === 0;
47     }
48
49     _siftUp(i) {
50         while (i > 0) {
51             const p = Math.floor((i - 1) / 2);
52             if (this.compare(this.heap[i], this.heap[p]) > 0) {
53                 [this.heap[i], this.heap[p]] = [this.heap[p], this.heap
54 [i]];
55                 i = p;
56             } else break;
57         }
58     }
59
60     _siftDown(i) {
61         const n = this.heap.length;
62         while (true) {
63             let largest = i;
64             const l = 2 * i + 1;
65             const r = 2 * i + 2;
66             if (l < n && this.compare(this.heap[l], this.heap[largest])
67 > 0) largest = l;
68             if (r < n && this.compare(this.heap[r], this.heap[largest])
69 > 0) largest = r;
70             if (largest !== i) {
71                 [this.heap[i], this.heap[largest]] = [this.heap[larges
72 t], this.heap[i]];
73                 i = largest;
74             } else break;
75         }
76     }
77 }
78
79 const rl = readline.createInterface({
80     input: process.stdin,
81     output: process.stdout
82 });
83
84 const tasks = [];
85
86 rl.on('line', (line) => {
87     if (line.trim() === '') {
88         rl.close();
89         return;
90     }
91     const [id, p, t, a] = line.trim().split(' ').map(Number);
92     tasks.push(new Task(id, p, t, a));
93 });

```

```

90
91     rl.on('close', () => {
92         const pq = new PriorityQueue();
93         let index = 0;
94         pq.push(tasks[index++]);
95         let currentTime = pq.peek().arriveTime;
96
97         while (index < tasks.length) {
98             const currentTask = pq.pop();
99             const nextTask = tasks[index++];
100
101            const endTime = currentTime + currentTask.executeTime;
102
103            // 当前任务不能执行完，下一个任务先到
104            if (endTime > nextTask.arriveTime) {
105                currentTask.executeTime -= (nextTask.arriveTime - currentTime);
106
107                currentTime = nextTask.arriveTime;
108                pq.push(currentTask); // 放回队列
109            } else {
110                // 当前任务可以完整执行
111                console.log(`#${currentTask.id} ${endTime}`);
112                currentTime = endTime;
113
114                // 执行等待队列中优先级高的任务，直到新任务到达
115                while (!pq.isEmpty() && currentTime < nextTask.arriveTime) {
116                    const idleTask = pq.pop();
117                    const idleEnd = currentTime + idleTask.executeTime;
118                    if (idleEnd > nextTask.arriveTime) {
119                        idleTask.executeTime -= (nextTask.arriveTime - currentTime);
120
121                        pq.push(idleTask);
122                        break;
123                    } else {
124                        console.log(`#${idleTask.id} ${idleEnd}`);
125                        currentTime = idleEnd;
126                    }
127
128                }
129
130                pq.push(nextTask);
131            }
132
133            // 队列中剩下的任务无抢占，直接执行
134            while (!pq.isEmpty()) {
135                const task = pq.pop();

```

```
136     const end = currentTime + task.executeTime;
137     console.log(`#${task.id} ${end}`);
138     currentTime = end;
139   }
140 );
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8     "strconv"
9     "strings"
10    )
11
12 // 定义任务结构体
13 type Task struct {
14     id         int
15     priority   int
16     executeTime int
17     arriveTime int
18 }
19
20 // 优先队列实现, 基于 container/heap
21 type TaskPQ []*Task
22
23 func (pq TaskPQ) Len() int { return len(pq) }
24
25 // 优先队列排序规则:
26 // priority 大的优先, priority 相等时 id 小的优先
27 func (pq TaskPQ) Less(i, j int) bool {
28     if pq[i].priority == pq[j].priority {
29         return pq[i].id < pq[j].id
30     }
31     return pq[i].priority > pq[j].priority
32 }
33
34 func (pq TaskPQ) Swap(i, j int) {
35     pq[i], pq[j] = pq[j], pq[i]
36 }
37
38 func (pq *TaskPQ) Push(x interface{}) {
39     *pq = append(*pq, x.(*Task))
40 }
41
42 func (pq *TaskPQ) Pop() interface{} {
43     old := *pq
44     n := len(old)
45     item := old[n-1]
```

```
46     *pq = old[:n-1]
47     return item
48 }
49
50 func main() {
51     reader := bufio.NewReader(os.Stdin)
52     var tasks []*Task
53
54     // 读入任务，空行结束
55     for {
56         line, err := reader.ReadString('\n')
57         if err != nil {
58             break
59         }
60         line = strings.TrimSpace(line)
61         if line == "" {
62             break
63         }
64         parts := strings.Fields(line)
65         if len(parts) < 4 {
66             continue
67         }
68         id, _ := strconv.Atoi(parts[0])
69         p, _ := strconv.Atoi(parts[1])
70         t, _ := strconv.Atoi(parts[2])
71         a, _ := strconv.Atoi(parts[3])
72         tasks = append(tasks, &Task{id: id, priority: p, executeTime: t,
73                               arriveTime: a})
74     }
75
76     pq := &TaskPQ{}
77     heap.Init(pq)
78
79     // 入队第一个任务
80     index := 0
81     heap.Push(pq, tasks[index])
82     index++
83     currentTime := (*pq)[0].arriveTime
84
85     // 处理所有任务
86     for index < len(tasks) {
87         currentTask := heap.Pop(pq).(*Task)
88         nextTask := tasks[index]
89         index++
90
91         currentTaskEndTime := currentTime + currentTask.executeTime
92
93         // 当前任务执行结束时间 大于 下一个任务到达时间
```

```
93     if currentTaskEndTime > nextTask.arriveTime {
94         // 扣除已执行时间
95         currentTask.executeTime -= (nextTask.arriveTime - currentTim
96     e)
97         currentTime = nextTask.arriveTime
98         heap.Push(pq, currentTask) // 重新入队剩余任务
99     } else {
100        // 当前任务执行完成
101        fmt.Println(currentTask.id, currentTaskEndTime)
102        currentTime = currentTaskEndTime
103
104        // 利用空闲时间执行等待队列中的任务
105        for pq.Len() > 0 && currentTime < nextTask.arriveTime {
106            idleTask := heap.Pop(pq).(*Task)
107            idleTaskEndTime := currentTime + idleTask.executeTime
108
109            if idleTaskEndTime > nextTask.arriveTime {
110                // 执行一部分, 剩余继续入队
111                idleTask.executeTime -= (nextTask.arriveTime - curren
112                tTime)
113                heap.Push(pq, idleTask)
114                break
115            } else {
116                // 执行完当前任务
117                fmt.Println(idleTask.id, idleTaskEndTime)
118                currentTime = idleTaskEndTime
119            }
120
121            currentTime = nextTask.arriveTime
122        }
123
124        // 新任务入队
125        heap.Push(pq, nextTask)
126    }
127
128    // 处理剩余任务 (无抢占)
129    for pq.Len() > 0 {
130        t := heap.Pop(pq).(*Task)
131        endTime := currentTime + t.executeTime
132        fmt.Println(t.id, endTime)
133        currentTime = endTime
134    }
}
```

| 来自: 华为OD机试 2025C卷 – 单核CPU任务调度 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机试 2025C卷 – 单核CPU任务调度 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025C卷 - 反转每对括号间的子串 (C++ & Python & JAVA & JS & GO)-CSDN博客

反转每对括号间的子串

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

给出一个字符串s，仅包含小写[英文字母](#)和括号。

请按照从括号内到外的顺序，逐层反转每对匹配括号中的字符串，并返回最终的结果。

输入描述

一个自有小写字母和括号的字符串，括号要成对匹配出现

输出描述

一个字符串

用例1

输入

```
Plain Text |  
1 abcd
```

输出

```
Plain Text |  
1 abcd
```

用例2

输入

Plain Text |

1 (u(love)i)

输出

Plain Text |

1 iloveu

用例3

输入

Plain Text |

1 (abcd)

输出

Plain Text |

1 dcba

题解

思路：栈数据结构应用题

初始定义一个栈，首先往栈顶压入一个空字符串，然后从左至右**遍历字符串**，对于每个字符的处理方式：

1. 小写字母，拼接到当前栈顶的字符串中。
2. (时，压入一个空字符串到栈顶中，表示开启一个括号。
3.) 时，将栈顶元素弹出，表示一个括号的结束。将反转后的字符串拼接在最新栈顶元素后。

遍历完字符串之后，当前栈顶字符串就是结果。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 #include<map>
10 #include<stack>
11 using namespace std;
12
13
14 int main() {
15     string input;
16     getline(cin, input);
17     stack<string> stk;
18
19     stk.push("");
20     for (char c : input) {
21         // 压入一个新字符串，处理括号问题
22         if (c == '(') {
23             stk.push("");
24             // 一个括号结束，反转栈顶元素
25         } else if (c == ')') {
26             string tmp = stk.top();
27             stk.pop();
28             reverse(tmp.begin(), tmp.end());
29
30             string top = stk.top();
31             stk.pop();
32             stk.push(top + tmp);
33             // 直接追加到栈顶元素
34         } else {
35             string top = stk.top();
36             stk.pop();
37             stk.push(top + c);
38         }
39     }
40
41     cout << stk.top();
42
43     return 0;
44 }
```

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         String input = scanner.nextLine(); // 读取输入
7         Stack<String> stk = new Stack<>();
8
9         stk.push("");
10        for (char c : input.toCharArray()) {
11            // 压入一个新字符串，处理括号问题
12            if (c == '(') {
13                stk.push("");
14            } else if (c == ')') {
15                String tmp = stk.pop();
16                tmp = new StringBuilder(tmp).reverse().toString(); // 反
转栈顶元素
17
18                String top = stk.pop();
19                stk.push(top + tmp);
20            } else {
21                // 直接追加到栈顶元素
22                String top = stk.pop();
23                stk.push(top + c);
24            }
25        }
26
27        System.out.println(stk.peek());
28    }
29 }
```

Python

```
1 def main():
2     input_str = input() # 读取输入
3     stk = [""]
4
5     for c in input_str:
6         # 压入一个新字符串，处理括号问题
7         if c == '(':
8             stk.append("")
9         elif c == ')':
10            tmp = stk.pop()
11            tmp = tmp[::-1] # 反转栈顶元素
12
13            top = stk.pop()
14            stk.append(top + tmp)
15        else:
16            # 直接追加到栈顶元素
17            top = stk.pop()
18            stk.append(top + c)
19
20     print(stk[-1])
21
22 if __name__ == "__main__":
23     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 // 创建接口来读取输入
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout
7 });
8
9 function main(input) {
10     const stk = [""];
11
12     for (let c of input) {
13         // 压入一个新字符串，处理括号问题
14         if (c === '(') {
15             stk.push("");
16         } else if (c === ')') {
17             let tmp = stk.pop();
18             tmp = tmp.split("").reverse().join(""); // 反转栈顶元素
19
20             let top = stk.pop();
21             stk.push(top + tmp);
22         } else {
23             // 直接追加到栈顶元素
24             let top = stk.pop();
25             stk.push(top + c);
26         }
27     }
28
29     console.log(stk[stk.length - 1]);
30 }
31
32 // 从控制台读取输入并调用主函数
33 rl.question('', (input) => {
34     main(input);
35     rl.close();
36 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 func main() {
10    scanner := bufio.NewScanner(os.Stdin)
11    scanner.Scan()
12    input := scanner.Text() // 读取输入
13    var stk []string
14
15    stk = append(stk, "")
16    for _, c := range input {
17        // 压入一个新字符串，处理括号问题
18        if c == '(' {
19            stk = append(stk, "")
20        } else if c == ')' {
21            tmp := stk[len(stk)-1]
22            stk = stk[:len(stk)-1]
23            // 反转栈顶元素
24            tmp = reverse(tmp)
25
26            top := stk[len(stk)-1]
27            stk = stk[:len(stk)-1]
28            stk = append(stk, top+tmp)
29        } else {
30            // 直接追加到栈顶元素
31            top := stk[len(stk)-1]
32            stk = stk[:len(stk)-1]
33            stk = append(stk, top+string(c))
34        }
35    }
36
37    fmt.Println(stk[len(stk)-1])
38 }
39
40 // 反转字符串的辅助函数
41 func reverse(s string) string {
42     result := []rune(s)
43     for i, j := 0, len(result)-1; i < j; i, j = i+1, j-1 {
44         result[i], result[j] = result[j], result[i]
45     }
}
```

```
46     return string(result)
47 }
```

来自: 华为OD机试 2025C卷 – 反转每对括号间的子串 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 统计监控 (C++ & Python & JAVA & JS & GO)-CSDN博客

统计监控

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

某长方形停车场，每个车位上方都有对应监控器，当且仅当在当前车位或者前后左右四个方向任意一个车位范围停车时，监控器才需要打开；给出某一时刻停车场的停车分布，请统计最少需要打开多少个监控器；

输入描述

第一行输入m, n表示长宽，满足 $1 < m, n \leq 20$ ；

后面输入m行，每行有n个0或1的整数，整数间使用一个空格隔开，表示该行已停车情况，其中0表示空位，1表示已停；

输出描述

最少需要打开监控器的数量；

用例1

输入

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

输出

```
1 5
```

题解

思路： 模拟

1. 从题目描述中可以得出，一个位置监控器要打开条件是 当前位置停了车或者四周位置停了车。
2. 基于1的规律，可以枚举每个位置监控器判断当前位置/四周位置是否停了车来决定是否要打开。累加 枚举位置过程中需要打开的监控器数量就是结果。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11 int main() {
12     int m,n;
13     cin >> m >> n;
14     vector<vector<int>> grid(m, vector<int>(n));
15     for (int i = 0; i < m; i++) {
16         for (int j = 0; j < n; j++) {
17             cin >> grid[i][j];
18         }
19     }
20
21     int direct[4][2] = {{1,0}, {-1, 0}, {0, 1}, {0, -1}};
22     int count = 0;
23
24     // 枚举每个车位监控, 判断是否需要打开
25     for (int i = 0; i < m; i++) {
26         for (int j = 0; j < n; j++) {
27             if (grid[i][j] == 1) {
28                 count++;
29                 continue;
30             }
31             // 四周是否停车
32             for (int k = 0; k < 4; k++) {
33                 int nx = i + direct[k][0];
34                 int ny = j + direct[k][1];
35                 // 越界
36                 if (nx < 0 || ny < 0 || nx >= m || ny >= n) {
37                     continue;
38                 }
39                 if (grid[nx][ny] == 1) {
40                     count++;
41                     break;
42                 }
43             }
44         }
45     }
}
```

```
46     cout << count;
47
48     return 0;
49 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int m = sc.nextInt(), n = sc.nextInt();
7         int[][] grid = new int[m][n];
8         for (int i = 0; i < m; i++) {
9             for (int j = 0; j < n; j++) {
10                 grid[i][j] = sc.nextInt();
11             }
12         }
13
14         int[][] direct = {{1,0}, {-1,0}, {0,1}, {0,-1}};
15         int count = 0;
16
17         // 枚举每个车位监控, 判断是否需要打开
18         for (int i = 0; i < m; i++) {
19             for (int j = 0; j < n; j++) {
20                 if (grid[i][j] == 1) {
21                     count++;
22                     continue;
23                 }
24                 // 四周是否停车
25                 for (int[] d : direct) {
26                     int nx = i + d[0];
27                     int ny = j + d[1];
28                     // 越界
29                     if (nx < 0 || ny < 0 || nx >= m || ny >= n) {
30                         continue;
31                     }
32                     if (grid[nx][ny] == 1) {
33                         count++;
34                         break;
35                     }
36                 }
37             }
38         }
39
40         System.out.println(count);
41     }
42 }
```

Python

```
1 m, n = map(int, input().split())
2 grid = [list(map(int, input().split())) for _ in range(m)]
3
4 direct = [(1,0), (-1,0), (0,1), (0,-1)]
5 count = 0
6
7 # 枚举每个车位监控，判断是否需要打开
8 for i in range(m):
9     for j in range(n):
10         if grid[i][j] == 1:
11             count += 1
12             continue
13         # 四周是否停车
14         for dx, dy in direct:
15             nx, ny = i + dx, j + dy
16             # 越界
17             if nx < 0 or ny < 0 or nx >= m or ny >= n:
18                 continue
19             if grid[nx][ny] == 1:
20                 count += 1
21                 break
22
23 print(count)
```

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.push(line.trim());
12 }).on("close", () => {
13     let [m, n] = input[0].split(" ").map(Number);
14     let grid = [];
15     for (let i = 1; i <= m; i++) {
16         grid.push(input[i].split(" ").map(Number));
17     }
18
19     let direct = [[1,0], [-1,0], [0,1], [0,-1]];
20     let count = 0;
21
22     // 枚举每个车位监控，判断是否需要打开
23     for (let i = 0; i < m; i++) {
24         for (let j = 0; j < n; j++) {
25             if (grid[i][j] === 1) {
26                 count++;
27                 continue;
28             }
29             // 四周是否停车
30             for (let [dx, dy] of direct) {
31                 let nx = i + dx;
32                 let ny = j + dy;
33                 // 越界
34                 if (nx < 0 || ny < 0 || nx >= m || ny >= n) continue;
35                 if (grid[nx][ny] === 1) {
36                     count++;
37                     break;
38                 }
39             }
40         }
41     }
42
43     console.log(count);
44 });
```

Go

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     var m, n int
9     fmt.Scan(&m, &n)
10    grid := make([][]int, m)
11    for i := range grid {
12        grid[i] = make([]int, n)
13        for j := 0; j < n; j++ {
14            fmt.Scan(&grid[i][j])
15        }
16    }
17
18    direct := [][]int{{1, 0}, {-1, 0}, {0, 1}, {0, -1}}
19    count := 0
20
21    // 枚举每个车位监控，判断是否需要打开
22    for i := 0; i < m; i++ {
23        for j := 0; j < n; j++ {
24            if grid[i][j] == 1 {
25                count++
26                continue
27            }
28            // 四周是否停车
29            for _, d := range direct {
30                nx := i + d[0]
31                ny := j + d[1]
32                // 越界
33                if nx < 0 || ny < 0 || nx >= m || ny >= n {
34                    continue
35                }
36                if grid[nx][ny] == 1 {
37                    count++
38                    break
39                }
40            }
41        }
42    }
43
44    fmt.Println(count)
45 }
```

统计监控

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

某长方形停车场，每个车位上方都有对应监控器，当且仅当在当前车位或者前后左右四个方向任意一个车位范围停车时，监控器才需要打开；给出某一时刻停车场的停车分布，请统计最少需要打开多少个监控器；

输入描述

第一行输入m, n表示长宽，满足 $1 < m, n \leq 20$ ；

后面输入m行，每行有n个0或1的整数，整数间使用一个空格隔开，表示该行已停车情况，其中0表示空位，1表示已停；

输出描述

最少需要打开监控器的数量；

用例1

输入

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

输出

```
1 5
```

题解

思路： [模拟](#)

1. 从题目描述中可以得出，一个位置监控器要打开条件是 当前位置停了车或者四周位置停了车。
2. 基于1的规律，可以枚举每个位置监控器判断当前位置/四周位置是否停了车来决定是否要打开。累加 枚举位置过程中需要打开的监控器数量就是结果。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11 int main() {
12     int m,n;
13     cin >> m >> n;
14     vector<vector<int>> grid(m, vector<int>(n));
15     for (int i = 0; i < m; i++) {
16         for (int j = 0; j < n; j++) {
17             cin >> grid[i][j];
18         }
19     }
20
21     int direct[4][2] = {{1,0}, {-1, 0}, {0, 1}, {0, -1}};
22     int count = 0;
23
24     // 枚举每个车位监控,判断是否需要打开
25     for (int i = 0; i < m; i++) {
26         for (int j = 0; j < n; j++) {
27             if (grid[i][j] == 1) {
28                 count++;
29                 continue;
30             }
31             // 四周是否停车
32             for (int k = 0; k < 4; k++) {
33                 int nx = i + direct[k][0];
34                 int ny = j + direct[k][1];
35                 // 越界
36                 if (nx < 0 || ny < 0 || nx >= m || ny >= n) {
37                     continue;
38                 }
39                 if (grid[nx][ny] == 1) {
40                     count++;
41                     break;
42                 }
43             }
44         }
45     }
}
```

```
46     cout << count;
47     return 0;
48 }
49 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int m = sc.nextInt(), n = sc.nextInt();
7         int[][] grid = new int[m][n];
8         for (int i = 0; i < m; i++) {
9             for (int j = 0; j < n; j++) {
10                 grid[i][j] = sc.nextInt();
11             }
12         }
13
14         int[][] direct = {{1,0}, {-1,0}, {0,1}, {0,-1}};
15         int count = 0;
16
17         // 枚举每个车位监控, 判断是否需要打开
18         for (int i = 0; i < m; i++) {
19             for (int j = 0; j < n; j++) {
20                 if (grid[i][j] == 1) {
21                     count++;
22                     continue;
23                 }
24                 // 四周是否停车
25                 for (int[] d : direct) {
26                     int nx = i + d[0];
27                     int ny = j + d[1];
28                     // 越界
29                     if (nx < 0 || ny < 0 || nx >= m || ny >= n) {
30                         continue;
31                     }
32                     if (grid[nx][ny] == 1) {
33                         count++;
34                         break;
35                     }
36                 }
37             }
38         }
39
40         System.out.println(count);
41     }
42 }
```

```
1 m, n = map(int, input().split())
2 grid = [list(map(int, input().split())) for _ in range(m)]
3
4 direct = [(1,0), (-1,0), (0,1), (0,-1)]
5 count = 0
6
7 # 枚举每个车位监控，判断是否需要打开
8 for i in range(m):
9     for j in range(n):
10         if grid[i][j] == 1:
11             count += 1
12             continue
13         # 四周是否停车
14         for dx, dy in direct:
15             nx, ny = i + dx, j + dy
16             # 越界
17             if nx < 0 or ny < 0 or nx >= m or ny >= n:
18                 continue
19             if grid[nx][ny] == 1:
20                 count += 1
21                 break
22
23 print(count)
```

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.push(line.trim());
12 }).on("close", () => {
13     let [m, n] = input[0].split(" ").map(Number);
14     let grid = [];
15     for (let i = 1; i <= m; i++) {
16         grid.push(input[i].split(" ").map(Number));
17     }
18
19     let direct = [[1,0], [-1,0], [0,1], [0,-1]];
20     let count = 0;
21
22     // 枚举每个车位监控，判断是否需要打开
23     for (let i = 0; i < m; i++) {
24         for (let j = 0; j < n; j++) {
25             if (grid[i][j] === 1) {
26                 count++;
27                 continue;
28             }
29             // 四周是否停车
30             for (let [dx, dy] of direct) {
31                 let nx = i + dx;
32                 let ny = j + dy;
33                 // 越界
34                 if (nx < 0 || ny < 0 || nx >= m || ny >= n) continue;
35                 if (grid[nx][ny] === 1) {
36                     count++;
37                     break;
38                 }
39             }
40         }
41     }
42
43     console.log(count);
44 });
```

Go

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     var m, n int
9     fmt.Scan(&m, &n)
10    grid := make([][]int, m)
11    for i := range grid {
12        grid[i] = make([]int, n)
13        for j := 0; j < n; j++ {
14            fmt.Scan(&grid[i][j])
15        }
16    }
17
18    direct := [][]int{{1, 0}, {-1, 0}, {0, 1}, {0, -1}}
19    count := 0
20
21    // 枚举每个车位监控，判断是否需要打开
22    for i := 0; i < m; i++ {
23        for j := 0; j < n; j++ {
24            if grid[i][j] == 1 {
25                count++
26                continue
27            }
28            // 四周是否停车
29            for _, d := range direct {
30                nx := i + d[0]
31                ny := j + d[1]
32                // 越界
33                if nx < 0 || ny < 0 || nx >= m || ny >= n {
34                    continue
35                }
36                if grid[nx][ny] == 1 {
37                    count++
38                    break
39                }
40            }
41        }
42    }
43
44    fmt.Println(count)
45 }
```

| 来自: 华为OD机考 2025C卷 – 统计监控 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考 2025C卷 – 统计监控 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025C卷 - 中庸行者 (C++ & Python & JAVA & JS & GO)-CSDN博客

中庸行者

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 $m \times n$ 的整数矩阵作为地图，矩阵数值为地形高度。中庸行者选择地图中的任意一点作为起点，尝试往上、下、左、右四个相邻格子移动；移动时有如下约束：

中庸行者只能上坡或者下坡，不能走到高度相同的点

不允许连续上坡或者连续下坡，需要交替进行

每个位置只能经过一次，不能重复行走

请给出中庸行者在本地图内，能连续移动的最大次数

输入描述

第一行两个数字，分别为行数和每行的列数；

后续数据为矩阵地图内容

矩阵边长范围1–8，地形高度范围0–100000

输出描述

一个整数，代表中庸行者在本地图内，能连续移动的最大次数

用例1

输入

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

输出

Plain Text |

```
1 3
```

用例2

输入

Plain Text |

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

输出

Plain Text |

```
1 4
```

题解

思路： DFS

1. 由于题目数据量比较小 矩阵边长范围1–8，所以可以直接采用将所有位置作为起点尝试进行递归，获取该位置能进行的最大步数。结果就是其中的最大值。
2. 额外需要注意这几个题目限制：
 - a. 中庸行者只能上坡或者下坡，不能走到高度相同的点，不允许连续上坡或者连续下坡，需要交替进行，可以使用一个状态值标志上一个移动是上坡还是下坡。
 - b. 每个位置只能经过一次，不能重复行走：可以使用一个 visited 辅助数组进行限制。

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 using namespace std;
11
12 int direct[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
13
14 // 记录最大步数
15 int res = 0;
16 int m, n;
17
18 // count 已经走过的步数 lastType 0 起点 1 上升 -1下降
19 void dfs(int x, int y, vector<vector<int>>& grid, vector<vector<bool>>& visitd, int count, int lastType) {
20     res = max(res, count);
21     for (int i = 0; i < 4; i++) {
22         int nx = x + direct[i][0];
23         int ny = y + direct[i][1];
24         // 越界 访问过
25         if (nx < 0 || ny < 0 || nx >= m || ny >= n || visitd[nx][ny]) {
26             continue;
27         }
28         // 高度相同
29         if(grid[nx][ny] == grid[x][y]) {
30             continue;
31         }
32
33         int currentType = grid[nx][ny] > grid[x][y] ? 1 : -1;
34         // 不允许连续爬升或下降
35         if (lastType != 0 && lastType == currentType) {
36             continue;
37         }
38         // 递归回溯
39         visitd[nx][ny] = true;
40         dfs(nx, ny, grid, visitd, count + 1, currentType);
41         visitd[nx][ny] = false;
42     }
43 }
44 }
```

```
45 int main() {
46
47     cin >> m >> n;
48     vector<vector<int>> grid(m, vector<int>(n, 0));
49
50     for (int i = 0; i < m; i++) {
51         for (int j = 0; j < n; j++) {
52             cin >> grid[i][j];
53         }
54     }
55
56     for (int i = 0; i < m; i++) {
57         for (int j = 0; j < n; j++) {
58             vector<vector<bool>> visitd(m, vector<bool>(n, false));
59             visitd[i][j] = true;
60             dfs(i, j, grid, visitd, 0, 0);
61         }
62     }
63     cout << res;
64     return 0;
65 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 四个方向移动
5     static int[][] direct = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
6     static int res = 0, m, n;
7
8     /**
9      * 深度优先搜索
10     * @param x 当前 x 坐标
11     * @param y 当前 y 坐标
12     * @param grid 高度矩阵
13     * @param visited 访问标记数组
14     * @param count 当前已经走过的步数
15     * @param lastType 0-起点 1-上升 -1-下降
16     */
17     static void dfs(int x, int y, int[][] grid, boolean[][] visited, int count, int lastType) {
18         res = Math.max(res, count);
19         for (int[] d : direct) {
20             int nx = x + d[0], ny = y + d[1];
21
22             // 越界或已经访问过
23             if (nx < 0 || ny < 0 || nx >= m || ny >= n || visited[nx][ny]) {
24                 continue;
25             }
26             // 高度相同则跳过
27             if (grid[nx][ny] == grid[x][y]) {
28                 continue;
29             }
30
31             int currentType = grid[nx][ny] > grid[x][y] ? 1 : -1;
32             // 不允许连续爬升或下降
33             if (lastType != 0 && lastType == currentType) {
34                 continue;
35             }
36
37             // 递归回溯
38             visited[nx][ny] = true;
39             dfs(nx, ny, grid, visited, count + 1, currentType);
40             visited[nx][ny] = false;
41         }
42     }
43 }
```

```
44     public static void main(String[] args) {
45         Scanner scanner = new Scanner(System.in);
46         // 读取 m 行 n 列
47         m = scanner.nextInt();
48         n = scanner.nextInt();
49         int[][] grid = new int[m][n];
50
51         // 读取网格数据
52         for (int i = 0; i < m; i++) {
53             for (int j = 0; j < n; j++) {
54                 grid[i][j] = scanner.nextInt();
55             }
56         }
57         scanner.close();
58
59         // 遍历所有起点
60         for (int i = 0; i < m; i++) {
61             for (int j = 0; j < n; j++) {
62                 boolean[][] visited = new boolean[m][n];
63                 visited[i][j] = true;
64                 dfs(i, j, grid, visited, 0, 0);
65             }
66         }
67
68         // 输出最大步数
69         System.out.println(res);
70     }
71 }
```

Python

```
1 import sys
2
3 # 方向数组: 上、下、左、右
4 direct = [(-1, 0), (1, 0), (0, -1), (0, 1)]
5 res = 0
6
7 def dfs(x, y, grid, visited, count, last_type):
8     """ 深度优先搜索 """
9     global res
10    res = max(res, count)
11
12    for dx, dy in direct:
13        nx, ny = x + dx, y + dy
14
15        # 越界或已经访问过
16        if nx < 0 or ny < 0 or nx >= m or ny >= n or visited[nx][ny]:
17            continue
18        # 高度相同则跳过
19        if grid[nx][ny] == grid[x][y]:
20            continue
21
22        current_type = 1 if grid[nx][ny] > grid[x][y] else -1
23        # 不允许连续爬升或下降
24        if last_type != 0 and last_type == current_type:
25            continue
26
27        # 递归回溯
28        visited[nx][ny] = True
29        dfs(nx, ny, grid, visited, count + 1, current_type)
30        visited[nx][ny] = False
31
32 if __name__ == "__main__":
33     # 读取输入
34     m, n = map(int, sys.stdin.readline().split())
35     grid = [list(map(int, sys.stdin.readline().split())) for _ in range(m)]
36
37     # 遍历所有起点
38     for i in range(m):
39         for j in range(n):
40             visited = [[False] * n for _ in range(m)]
41             visited[i][j] = True
42             dfs(i, j, grid, visited, 0, 0)
43
44     # 输出最大步数
```

```
45     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 input = [];
9 let m, n, grid;
10 const directions = [[-1, 0], [1, 0], [0, -1], [0, 1]];
11 let res = 0;
12
13 /**
14 * 深度优先搜索 (DFS)
15 * @param {number} x - 当前 x 坐标
16 * @param {number} y - 当前 y 坐标
17 * @param {Array} visited - 访问记录数组
18 * @param {number} count - 已走步数
19 * @param {number} lastType - 上升 (1), 下降 (-1), 初始点 (0)
20 */
21 const dfs = (x, y, visited, count, lastType) => {
22     res = Math.max(res, count);
23
24     for (let [dx, dy] of directions) {
25         let nx = x + dx, ny = y + dy;
26
27         // 越界检查或已经访问
28         if (nx < 0 || ny < 0 || nx >= m || ny >= n || visited[nx][ny]) continue;
29         // 高度相同, 无法前进
30         if (grid[nx][ny] === grid[x][y]) continue;
31
32         let currentType = grid[nx][ny] > grid[x][y] ? 1 : -1;
33         // 不允许连续爬升或下降
34         if (lastType !== 0 && lastType === currentType) continue;
35
36         // 递归回溯
37         visited[nx][ny] = true;
38         dfs(nx, ny, visited, count + 1, currentType);
39         visited[nx][ny] = false;
40     }
41 };
42
43 // 读取输入
44 rl.on('line', (line) => {
```

```
45     input.push(line.trim());
46 }).on('close', () => {
47     // 解析输入数据
48     [m, n] = input[0].split(' ').map(Number);
49     grid = input.slice(1).map(row => row.split(' ').map(Number));
50
51     // 遍历所有起点
52     for (let i = 0; i < m; i++) {
53         for (let j = 0; j < n; j++) {
54             let visited = Array.from({ length: m }, () => Array(n).fill(false));
55             visited[i][j] = true;
56             dfs(i, j, visited, 0, 0);
57         }
58     }
59
60     // 输出最大步数
61     console.log(res);
62 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 方向数组: 上、下、左、右
12 var directions = [][]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}
13 var res, m, n int
14
15 /**
16 * 深度优先搜索 (DFS)
17 * @param x 当前 x 坐标
18 * @param y 当前 y 坐标
19 * @param grid 高度矩阵
20 * @param visited 访问记录数组
21 * @param count 当前已经走过的步数
22 * @param lastType 0-起点 1-上升 -1-下降
23 */
24 func dfs(x, y int, grid [][]int, visited [][]bool, count, lastType int) {
25     res = max(res, count)
26
27     for _, d := range directions {
28         nx, ny := x+d[0], y+d[1]
29
30         // 越界或已访问
31         if nx < 0 || ny < 0 || nx >= m || ny >= n || visited[nx][ny] {
32             continue
33         }
34         // 高度相同则跳过
35         if grid[nx][ny] == grid[x][y] {
36             continue
37         }
38
39         currentType := 1
40         if grid[nx][ny] < grid[x][y] {
41             currentType = -1
42         }
43
44         // 不允许连续爬升或下降
45         if lastType != 0 && lastType == currentType {
```

```

46         continue
47     }
48
49     // 递归回溯
50     visited[nx][ny] = true
51     dfs(nx, ny, grid, visited, count+1, currentType)
52     visited[nx][ny] = false
53   }
54 }
55
56 /**
57 * 读取标准输入
58 * @return int 矩阵行数和列数, 以及网格数据
59 */
60 func main() {
61   // 读取输入
62   scanner := bufio.NewScanner(os.Stdin)
63   scanner.Scan()
64   firstLine := strings.Fields(scanner.Text())
65   m, _ = strconv.Atoi(firstLine[0])
66   n, _ = strconv.Atoi(firstLine[1])
67
68   // 读取网格数据
69   grid := make([][]int, m)
70   for i := 0; i < m; i++ {
71     scanner.Scan()
72     rowStr := strings.Fields(scanner.Text())
73     grid[i] = make([]int, n)
74     for j := 0; j < n; j++ {
75       grid[i][j], _ = strconv.Atoi(rowStr[j])
76     }
77   }
78
79   // 遍历所有起点
80   for i := 0; i < m; i++ {
81     for j := 0; j < n; j++ {
82       visited := make([][]bool, m)
83       for k := range visited {
84         visited[k] = make([]bool, n)
85       }
86       visited[i][j] = true
87       dfs(i, j, grid, visited, 0, 0)
88     }
89   }
90
91   // 输出最大步数
92   fmt.Println(res)
93 }
```

```
94
95     /**
96      * 返回两个整数中的最大值
97      */
98     func max(a, b int) int {
99         if a > b {
100             return a
101         }
102         return b
103     }
```

中庸行者

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 $m \times n$ 的整数矩阵作为地图，矩阵数值为地形高度。中庸行者选择地图中的任意一点作为起点，尝试往上、下、左、右四个相邻格子移动；移动时有如下约束：

中庸行者只能上坡或者下坡，不能走到高度相同的点

不允许连续上坡或者连续下坡，需要交替进行

每个位置只能经过一次，不能重复行走

请给出中庸行者在本地图内，能连续移动的最大次数

输入描述

第一行两个数字，分别为行数和每行的列数；

后续数据为矩阵地图内容

矩阵边长范围1–8，地形高度范围0–100000

输出描述

一个整数，代表中庸行者在本地图内，能连续移动的最大次数

用例1

输入

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

Plain Text

输出

```
Plain Text |  
1 3
```

用例2

输入

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

输出

```
Plain Text |  
1 4
```

题解

思路： **DFS**

1. 由于题目数据量比较小 矩阵边长范围1–8，所以可以直接采用将所有位置作为起点尝试进行递归，获取该位置能进行的最大步数。结果就是其中的最大值。
2. 额外需要注意这几个题目限制：
 - a. 中庸行者只能上坡或者下坡，不能走到高度相同的点，不允许连续上坡或者连续下坡，需要交替进行，可以使用一个状态值标志上一个移动是上坡还是下坡。
 - b. 每个位置只能经过一次，不能重复行走：可以使用一个 `visited` 辅助数组进行限制。

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 using namespace std;
11
12 int direct[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
13
14 // 记录最大步数
15 int res = 0;
16 int m, n;
17
18 // count 已经走过的步数 lastType 0 起点 1 上升 -1下降
19 void dfs(int x, int y, vector<vector<int>>& grid, vector<vector<bool>>& visitd, int count, int lastType) {
20     res = max(res, count);
21     for (int i = 0; i < 4; i++) {
22         int nx = x + direct[i][0];
23         int ny = y + direct[i][1];
24         // 越界 访问过
25         if (nx < 0 || ny < 0 || nx >= m || ny >= n || visitd[nx][ny]) {
26             continue;
27         }
28         // 高度相同
29         if(grid[nx][ny] == grid[x][y]) {
30             continue;
31         }
32
33         int currentType = grid[nx][ny] > grid[x][y] ? 1 : -1;
34         // 不允许连续爬升或下降
35         if (lastType != 0 && lastType == currentType) {
36             continue;
37         }
38         // 递归回溯
39         visitd[nx][ny] = true;
40         dfs(nx, ny, grid, visitd, count + 1, currentType);
41         visitd[nx][ny] = false;
42     }
43 }
44 }
```

```
45 int main() {
46
47     cin >> m >> n;
48     vector<vector<int>> grid(m, vector<int>(n, 0));
49
50     for (int i = 0; i < m; i++) {
51         for (int j = 0; j < n; j++) {
52             cin >> grid[i][j];
53         }
54     }
55
56     for (int i = 0; i < m; i++) {
57         for (int j = 0; j < n; j++) {
58             vector<vector<bool>> visitd(m, vector<bool>(n, false));
59             visitd[i][j] = true;
60             dfs(i, j, grid, visitd, 0, 0);
61         }
62     }
63     cout << res;
64     return 0;
65 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 四个方向移动
5     static int[][] direct = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
6     static int res = 0, m, n;
7
8     /**
9      * 深度优先搜索
10     * @param x 当前 x 坐标
11     * @param y 当前 y 坐标
12     * @param grid 高度矩阵
13     * @param visited 访问标记数组
14     * @param count 当前已经走过的步数
15     * @param lastType 0-起点 1-上升 -1-下降
16     */
17     static void dfs(int x, int y, int[][] grid, boolean[][] visited, int count, int lastType) {
18         res = Math.max(res, count);
19         for (int[] d : direct) {
20             int nx = x + d[0], ny = y + d[1];
21
22             // 越界或已经访问过
23             if (nx < 0 || ny < 0 || nx >= m || ny >= n || visited[nx][ny]) {
24                 continue;
25             }
26             // 高度相同则跳过
27             if (grid[nx][ny] == grid[x][y]) {
28                 continue;
29             }
30
31             int currentType = grid[nx][ny] > grid[x][y] ? 1 : -1;
32             // 不允许连续爬升或下降
33             if (lastType != 0 && lastType == currentType) {
34                 continue;
35             }
36
37             // 递归回溯
38             visited[nx][ny] = true;
39             dfs(nx, ny, grid, visited, count + 1, currentType);
40             visited[nx][ny] = false;
41         }
42     }
43 }
```

```
44     public static void main(String[] args) {
45         Scanner scanner = new Scanner(System.in);
46         // 读取 m 行 n 列
47         m = scanner.nextInt();
48         n = scanner.nextInt();
49         int[][] grid = new int[m][n];
50
51         // 读取网格数据
52         for (int i = 0; i < m; i++) {
53             for (int j = 0; j < n; j++) {
54                 grid[i][j] = scanner.nextInt();
55             }
56         }
57         scanner.close();
58
59         // 遍历所有起点
60         for (int i = 0; i < m; i++) {
61             for (int j = 0; j < n; j++) {
62                 boolean[][] visited = new boolean[m][n];
63                 visited[i][j] = true;
64                 dfs(i, j, grid, visited, 0, 0);
65             }
66         }
67
68         // 输出最大步数
69         System.out.println(res);
70     }
71 }
```

Python

```
1 import sys
2
3 # 方向数组: 上、下、左、右
4 direct = [(-1, 0), (1, 0), (0, -1), (0, 1)]
5 res = 0
6
7 def dfs(x, y, grid, visited, count, last_type):
8     """ 深度优先搜索 """
9     global res
10    res = max(res, count)
11
12    for dx, dy in direct:
13        nx, ny = x + dx, y + dy
14
15        # 越界或已经访问过
16        if nx < 0 or ny < 0 or nx >= m or ny >= n or visited[nx][ny]:
17            continue
18        # 高度相同则跳过
19        if grid[nx][ny] == grid[x][y]:
20            continue
21
22        current_type = 1 if grid[nx][ny] > grid[x][y] else -1
23        # 不允许连续爬升或下降
24        if last_type != 0 and last_type == current_type:
25            continue
26
27        # 递归回溯
28        visited[nx][ny] = True
29        dfs(nx, ny, grid, visited, count + 1, current_type)
30        visited[nx][ny] = False
31
32 if __name__ == "__main__":
33     # 读取输入
34     m, n = map(int, sys.stdin.readline().split())
35     grid = [list(map(int, sys.stdin.readline().split())) for _ in range(m)]
36
37     # 遍历所有起点
38     for i in range(m):
39         for j in range(n):
40             visited = [[False] * n for _ in range(m)]
41             visited[i][j] = True
42             dfs(i, j, grid, visited, 0, 0)
43
44     # 输出最大步数
```

```
45     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 input = [];
9 let m, n, grid;
10 const directions = [[-1, 0], [1, 0], [0, -1], [0, 1]];
11 let res = 0;
12
13 /**
14 * 深度优先搜索 (DFS)
15 * @param {number} x - 当前 x 坐标
16 * @param {number} y - 当前 y 坐标
17 * @param {Array} visited - 访问记录数组
18 * @param {number} count - 已走步数
19 * @param {number} lastType - 上升 (1), 下降 (-1), 初始点 (0)
20 */
21 const dfs = (x, y, visited, count, lastType) => {
22     res = Math.max(res, count);
23
24     for (let [dx, dy] of directions) {
25         let nx = x + dx, ny = y + dy;
26
27         // 越界检查或已经访问
28         if (nx < 0 || ny < 0 || nx >= m || ny >= n || visited[nx][ny]) continue;
29         // 高度相同, 无法前进
30         if (grid[nx][ny] === grid[x][y]) continue;
31
32         let currentType = grid[nx][ny] > grid[x][y] ? 1 : -1;
33         // 不允许连续爬升或下降
34         if (lastType !== 0 && lastType === currentType) continue;
35
36         // 递归回溯
37         visited[nx][ny] = true;
38         dfs(nx, ny, visited, count + 1, currentType);
39         visited[nx][ny] = false;
40     }
41 };
42
43 // 读取输入
44 rl.on('line', (line) => {
```

```
45     input.push(line.trim());
46 }).on('close', () => {
47     // 解析输入数据
48     [m, n] = input[0].split(' ').map(Number);
49     grid = input.slice(1).map(row => row.split(' ').map(Number));
50
51     // 遍历所有起点
52     for (let i = 0; i < m; i++) {
53         for (let j = 0; j < n; j++) {
54             let visited = Array.from({ length: m }, () => Array(n).fill(false));
55             visited[i][j] = true;
56             dfs(i, j, visited, 0, 0);
57         }
58     }
59
60     // 输出最大步数
61     console.log(res);
62 });

});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 方向数组: 上、下、左、右
12 var directions = [][]int{{-1, 0}, {1, 0}, {0, -1}, {0, 1}}
13 var res, m, n int
14
15 /**
16 * 深度优先搜索 (DFS)
17 * @param x 当前 x 坐标
18 * @param y 当前 y 坐标
19 * @param grid 高度矩阵
20 * @param visited 访问记录数组
21 * @param count 当前已经走过的步数
22 * @param lastType 0-起点 1-上升 -1-下降
23 */
24 func dfs(x, y int, grid [][]int, visited [][]bool, count, lastType int) {
25     res = max(res, count)
26
27     for _, d := range directions {
28         nx, ny := x+d[0], y+d[1]
29
30         // 越界或已访问
31         if nx < 0 || ny < 0 || nx >= m || ny >= n || visited[nx][ny] {
32             continue
33         }
34         // 高度相同则跳过
35         if grid[nx][ny] == grid[x][y] {
36             continue
37         }
38
39         currentType := 1
40         if grid[nx][ny] < grid[x][y] {
41             currentType = -1
42         }
43
44         // 不允许连续爬升或下降
45         if lastType != 0 && lastType == currentType {
```

```

46         continue
47     }
48
49     // 递归回溯
50     visited[nx][ny] = true
51     dfs(nx, ny, grid, visited, count+1, currentType)
52     visited[nx][ny] = false
53   }
54 }
55
56 /**
57 * 读取标准输入
58 * @return int 矩阵行数和列数, 以及网格数据
59 */
60 func main() {
61   // 读取输入
62   scanner := bufio.NewScanner(os.Stdin)
63   scanner.Scan()
64   firstLine := strings.Fields(scanner.Text())
65   m, _ = strconv.Atoi(firstLine[0])
66   n, _ = strconv.Atoi(firstLine[1])
67
68   // 读取网格数据
69   grid := make([][]int, m)
70   for i := 0; i < m; i++ {
71     scanner.Scan()
72     rowStr := strings.Fields(scanner.Text())
73     grid[i] = make([]int, n)
74     for j := 0; j < n; j++ {
75       grid[i][j], _ = strconv.Atoi(rowStr[j])
76     }
77   }
78
79   // 遍历所有起点
80   for i := 0; i < m; i++ {
81     for j := 0; j < n; j++ {
82       visited := make([][]bool, m)
83       for k := range visited {
84         visited[k] = make([]bool, n)
85       }
86       visited[i][j] = true
87       dfs(i, j, grid, visited, 0, 0)
88     }
89   }
90
91   // 输出最大步数
92   fmt.Println(res)
93 }
```

```
94
95     /**
96      * 返回两个整数中的最大值
97      */
98     func max(a, b int) int {
99         if a > b {
100             return a
101         }
102         return b
103     }
```

| 来自: 华为OD机试 2025C卷 – 中庸行者 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机试 2025C卷 – 中庸行者 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 网格红绿灯最短路径 (C++ & Python & JAVA & JS & GO)-CSDN博客

网格红绿灯最短路径

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

给定一个二维的m*n网格地图(grids二维数组)，每个单元格0为空，1是障碍物，2是红绿灯；每一步可以在0或者2的单元格移动，每一秒可以走一个单元格；遇到红绿灯想要通过需要等待不同的时间才能通过，大小为x的light数组标注灯的坐标和等待时间，例如(2,2,3),坐标(2,2)红绿灯等待时间3秒，问从左上角(0,0)到右下角(m-1,n-1)所需的最短时间。

输入描述

grids二维数组，内部数据只有0, 1, 2, 1<m,n<=100

lights红绿灯数组，1<x<=m*n

输出描述

从坐标(0,0)到(m-1,n-1)坐标所需的最短时间，如果没有路径，则返回最短时间为-1.

示例1

输入

```
1  [[0,1,0],[0,2,1],[0,0,0]]  
2  [[1,1,3]]
```

输出

```
1  4
```

题解

思路： 单点最短路问题

1. 本题可以转换为 **单点最短路问题**，可以把矩阵看作有向图，默认每个位置和四周相连，边的距离默认为1。额外特殊处理点如果某个位置值为1，它与四周不可达/四周的点与它也不可达。对于位置为2的红绿灯位置，四周的点到它的位置边距离为1，它到四周的点的边距额外需要加上红绿灯等待时间。
2. 明白1的转换过程之后，就可以使用单点最短路 **Dijkstra** 算法求解进行处理了。下面代码逻辑使用 **优先队列** 数据结构来模拟算法实现。简单说说这个算法的逻辑：
 - dijkstra基于贪心的思维，每次找到当前离起点最短位置并且之前未访问的的点去更新与它相连的点的最短距离。为了记录访问状态，一般会定义 **visited** 数组去进行记录。为了快速找到离起点最短距离位置可以使用 **优先队列** 加速。
 - 定义 **dp[]**，其中 **dp[i]** 表示i离起点的最短距离，初始一般会设置为一个较大值，注意下面代码将 **visited** 和 **dp** 数组复用了。初始将 **dp[起点索引] = 0** 并将起点加入到优先队列中。
 - 每一轮循环中都会取出优先队列堆顶元素，更新与它相连点的最短距离，例如当前点x为优先队列堆顶元素，与它相连的点为y，如果 **dp[i] + grid[x][y] < dp[y]**，则更新 **dp[y] = dp[x] + grid[x][y]**，并将 **y** 加入到优先队列中。重复循环操作，直到优先队列为空结束。
3. 通过2的，最终可以得到起点和终点的距离 **dp[m*n-1]**，就可得出结果。

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<queue>
10 #include<climits>
11 using namespace std;
12
13 // 通用 切割函数 函数 将字符串str根据delimiter进行切割
14 vector<string> split(const string& str, const string& delimiter) {
15     vector<string> result;
16     size_t start = 0;
17     size_t end = str.find(delimiter);
18     while (end != string::npos) {
19         result.push_back(str.substr(start, end - start));
20         start = end + delimiter.length();
21         end = str.find(delimiter, start);
22     }
23     // 添加最后一个部分
24     result.push_back(str.substr(start));
25     return result;
26 }
27
28
29 struct cmp {
30     bool operator()(const pair<int, int>& a, const pair<int, int>& b) const {
31         return a.first < b.first;
32     }
33 };
34
35 // 解析字符串函数 将[[0,1,0],[0,2,1],[0,0,1]] 转化为二维数字数组
36 vector<vector<int>> parseInputStr(string input) {
37     vector<vector<int>> result;
38     if (input.empty()) {
39         return result;
40     }
41     // [[0,1,0],[0,2,1],[0,0,1]] => 0,1,0],[0,2,1],[0,0,1
42     input = input.substr(2, input.size() - 4);
43     vector<string> gridRow = split(input, "],[" );
44     for (int i = 0; i < gridRow.size(); i++) {
```

```

45     vector<string> tmpStr = split(gridRow[i], ",");
46     vector<int> tmp;
47     for (int j = 0 ; j < tmpStr.size();j++) {
48         tmp.push_back(stoi(tmpStr[j]));
49     }
50     result.push_back(tmp);
51 }
52 return result;
53 }

55 // 最短路算法
56 int BFS(vector<vector<int>>& grid, vector<vector<int>>& lights) {
57     // 四个方向
58     vector<vector<int>> directed = {{0,1}, {0, -1}, {1, 0}, {-1, 0}};
59     int m = grid.size();
60     int n = grid[0].size();
61     // 起点/终点就是障碍物
62     if (grid[0][0] == 1 || grid[m-1][n-1] == 1) {
63         return -1;
64     }
65

66

67     // 使用哈希存储每个红绿灯的等待时间，便于后续查找
68     map<int, int> lightsColWaitMap;
69     for (int i = 0; i < lights.size(); i++) {
70         int x = lights[i][0];
71         int y = lights[i][1];
72         int time = lights[i][2];
73         int pos = x * n + y;
74         lightsColWaitMap[pos] = time;
75     }
76     // 防止重复访问
77     vector<int> visited(m*n ,INT_MAX);
78     // 使用优先队列模拟bfs {所用时间,位置}
79     priority_queue<pair<int, int>, vector<pair<int,int>>, cmp> pq;
80     pq.push({0, 0});
81     visited[0] = 0;
82

83     while (!pq.empty()) {
84         pair<int,int> top = pq.top();
85         pq.pop();
86         int pos = top.second;
87         int time = top.first;
88         int x = pos/ n;
89         int y = pos % n;
90         // 一秒走一格
91         int nextStepTime = time + 1;
92         // 额外判断是否为红绿灯

```

```

93         if (grid[x][y] == 2) {
94             nextStepTime += lightsColWaitMap[pos];
95         }
96         // 向四个方向遍历
97         for (int i = 0; i < 4; i++) {
98             int nextX = x + directed[i][0];
99             int nextY = y + directed[i][1];
100            int nextPos = nextX * n + nextY;
101            // 越界或者先前有更小步数到达或者为障碍物
102            if (nextX < 0 || nextX >= m || nextY < 0 || nextY >= n || vis-
103                ited[nextPos] <= nextStepTime || grid[nextX][nextY] == 1) {
104                    continue;
105                }
106
107                visited[nextPos] = nextStepTime;
108                pq.push({nextStepTime, nextPos});
109            }
110        }
111
112    return visited[m * n - 1] == INT_MAX ? -1 : visited[m * n - 1];
113 }
114
115 int main() {
116     string gridStr;
117     getline(cin, gridStr);
118     string lightsStr;
119     getline(cin, lightsStr);
120
121     vector<vector<int>> grid = parseInputStr(gridStr);
122     vector<vector<int>> lights = parseInputStr(lightsStr);
123     int res = BFS(grid, lights);
124     cout << res;
125     return 0;
126 }
```

Java

```

1 import java.util.*;
2
3 public class Main {
4
5     // 解析字符串函数
6     static List<List<Integer>> parseInput(String input) {
7         List<List<Integer>> result = new ArrayList<>();
8         input = input.trim();
9         if (input.length() <= 4) return result; // 包括 [[ 和 ]]
10        input = input.substring(2, input.length() - 2); // 去除外层 [[ 和
11    ]
12    String[] rows = input.split("],\\\["); // 正确: 用双反斜杠
13    for (String row : rows) {
14        List<Integer> nums = new ArrayList<>();
15        for (String num : row.split(",,")) {
16            nums.add(Integer.parseInt(num));
17        }
18        result.add(nums);
19    }
20    return result;
21 }
22
23     // 最短路算法
24     static int bfs(List<List<Integer>> grid, List<List<Integer>> lights) {
25         int m = grid.size();
26         int n = grid.get(0).size();
27         int[][] directions = {{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
28
29         // 起点或终点为障碍物
30         if (grid.get(0).get(0) == 1 || grid.get(m - 1).get(n - 1) == 1) {
31             return -1;
32         }
33
34         // 使用 HashMap 存储红绿灯等待时间
35         Map<Integer, Integer> lightMap = new HashMap<>();
36         for (List<Integer> light : lights) {
37             int x = light.get(0), y = light.get(1), t = light.get(2);
38             int pos = x * n + y;
39             lightMap.put(pos, t);
40         }
41
42         int[] visited = new int[m * n];
43         Arrays.fill(visited, Integer.MAX_VALUE);
44         PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> Integer.co
mpare(a[0], b[0])); // (time, pos)

```

```

44
45     pq.offer(new int[]{0, 0});
46     visited[0] = 0;
47
48     while (!pq.isEmpty()) {
49         int[] top = pq.poll();
50         int time = top[0], pos = top[1];
51         int x = pos / n, y = pos % n;
52
53         int nextTime = time + 1; // 每次移动耗时 1
54
55         // 当前是红绿灯
56         if (grid.get(x).get(y) == 2) {
57             nextTime += lightMap.getOrDefault(pos, 0);
58         }
59
60         for (int[] dir : directions) {
61             int nx = x + dir[0];
62             int ny = y + dir[1];
63             int npos = nx * n + ny;
64
65             // 越界、障碍或已有更优解
66             if (nx < 0 || nx >= m || ny < 0 || ny >= n) continue;
67             if (grid.get(nx).get(ny) == 1 || visited[npos] <= nextTim
e) continue;
68
69             visited[npos] = nextTime;
70             pq.offer(new int[]{nextTime, npos});
71         }
72     }
73
74     return visited[m * n - 1] == Integer.MAX_VALUE ? -1 : visited[m *
n - 1];
75 }
76
77     public static void main(String[] args) {
78         Scanner sc = new Scanner(System.in);
79         String gridStr = sc.nextLine();
80         String lightsStr = sc.nextLine();
81
82         List<List<Integer>> grid = parseInput(gridStr);
83         List<List<Integer>> lights = parseInput(lightsStr);
84
85         System.out.println(bfs(grid, lights));
86     }
87 }
```

Python

```
1 # Python 版本 (ACM 模式, 保留注释)
2 import sys
3 import heapq
4
5 # 将字符串解析为二维数字数组, 如 "[[0,1,0],[0,2,1],[0,0,1]]"
6 def parse_input(s):
7     s = s.strip()[2:-2] # 去掉外部的 [[ 和 ]]
8     if not s:
9         return []
10    rows = s.split("], [")
11    return [list(map(int, row.split(","))) for row in rows]
12 # 最短路算法
13 def bfs(grid, lights):
14     m, n = len(grid), len(grid[0])
15     directions = [(0,1), (0,-1), (1,0), (-1,0)]
16
17     # 起点或终点为障碍物
18     if grid[0][0] == 1 or grid[m-1][n-1] == 1:
19         return -1
20
21     # 构建红绿灯等待时间映射表, key 用 pos = x * n + y 编码
22     light_map = {}
23     for x, y, t in lights:
24         light_map[x * n + y] = t
25
26     visited = [float('inf')] * (m * n)
27     pq = [(0, 0)] # 最小堆模拟优先队列 (time, position)
28     visited[0] = 0
29
30     while pq:
31         time, pos = heapq.heappop(pq)
32         x, y = divmod(pos, n)
33
34         next_time = time + 1 # 每次移动耗时 1
35
36         # 当前是红绿灯格子
37         if grid[x][y] == 2:
38             next_time += light_map.get(pos, 0)
39
40         for dx, dy in directions:
41             nx, ny = x + dx, y + dy
42             npos = nx * n + ny
43
44             # 越界或障碍物或已有更优路径
45             if not (0 <= nx < m and 0 <= ny < n):
```

```
46         continue
47     if grid[nx][ny] == 1 or visited[npos] <= next_time:
48         continue
49
50     visited[npos] = next_time
51     heapq.heappush(pq, (next_time, npos))
52
53     return -1 if visited[m*n-1] == float('inf') else visited[m*n-1]
54
55 if __name__ == '__main__':
56     grid_str = sys.stdin.readline()
57     lights_str = sys.stdin.readline()
58     grid = parse_input(grid_str)
59     lights = parse_input(lights_str)
60     print(bfs(grid, lights))
```

JavaScript

```
1 // 使用数组模拟优先队列代码
2 const readline = require('readline');
3
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout
7 });
8
9 const lines = [];
10 rl.on('line', (line) => {
11     lines.push(line.trim());
12     if (lines.length === 2) {
13         const grid = parseInput(lines[0]);
14         const lights = parseInput(lines[1]);
15         const res = bfs(grid, lights);
16         console.log(res);
17         process.exit(0);
18     }
19 });
20
21 // 将字符串转换为二维数字数组, 如 "[[0,1,0],[0,2,1],[0,0,1]]"
22 function parseInput(input) {
23     input = input.trim();
24     if (input === "[]") return [];
25     input = input.slice(2, -2); // 去除最外层 [[ 和 ]]
26     const rows = input.split("], [");
27     return rows.map(row => row.split(',') .map(Number));
28 }
29
30 // 最短路算法
31 function bfs(grid, lights) {
32     const m = grid.length;
33     const n = grid[0].length;
34
35     // 起点或终点为障碍物
36     if (grid[0][0] === 1 || grid[m - 1][n - 1] === 1) {
37         return -1;
38     }
39
40     const directions = [[0,1], [0,-1], [1,0], [-1,0]];
41
42     // 哈希表存储红绿灯等待时间, key = x * n + y
43     const lightMap = new Map();
44     for (const [x, y, t] of lights) {
45         lightMap.set(x * n + y, t);
```

```

46     }
47
48     // 最短时间记录数组
49     const visited = Array(m * n).fill(Infinity);
50     const pq = [[0, 0]]; // [time, pos]
51     visited[0] = 0;
52
53     while (pq.length > 0) {
54         // 模拟优先队列，弹出最短时间节点
55         pq.sort((a, b) => a[0] - b[0]);
56         const [time, pos] = pq.shift();
57         const x = Math.floor(pos / n);
58         const y = pos % n;
59
60         let nextTime = time + 1;
61
62         // 红绿灯等待处理
63         if (grid[x][y] === 2) {
64             nextTime += lightMap.get(pos) || 0;
65         }
66
67         for (const [dx, dy] of directions) {
68             const nx = x + dx;
69             const ny = y + dy;
70             const npos = nx * n + ny;
71
72             // 越界或为障碍或已经访问过更优路径
73             if (
74                 nx < 0 || nx >= m || ny < 0 || ny >= n ||
75                 grid[nx][ny] === 1 || visited[npos] <= nextTime
76             ) {
77                 continue;
78             }
79
80             visited[npos] = nextTime;
81             pq.push([nextTime, npos]);
82         }
83     }
84
85     const result = visited[m * n - 1];
86     return result === Infinity ? -1 : result;
87 }

```

```
1 // 自定义实现优先队列写法
2 const readline = require('readline');
3
4 const rl = readline.createInterface({
5     input: process.stdin,
6     output: process.stdout
7 });
8
9 const lines = [];
10 rl.on('line', (line) => {
11     lines.push(line.trim());
12     if (lines.length === 2) {
13         const grid = parseInput(lines[0]);
14         const lights = parseInput(lines[1]);
15         const res = bfs(grid, lights);
16         console.log(res);
17         process.exit(0);
18     }
19 });
20
21 // 将字符串转换为二维数字数组, 如 "[[0,1,0],[0,2,1],[0,0,1]]"
22 function parseInput(input) {
23     input = input.trim();
24     if (input === "") return [];
25     input = input.slice(2, -2); // 去除最外层 [[ 和 ]]
26     const rows = input.split("],[");
27     return rows.map(row => row.split(',').map(Number));
28 }
29
30 // 自定义最小堆实现的优先队列
31 class PriorityQueue {
32     constructor() {
33         this.heap = [];
34     }
35
36     push(item) {
37         this.heap.push(item);
38         this._siftUp();
39     }
40
41     pop() {
42         if (this.size() === 0) return null;
43         const top = this.heap[0];
44         const last = this.heap.pop();
45         if (this.size() !== 0) {
```

```

46         this.heap[0] = last;
47         this._siftDown();
48     }
49     return top;
50 }
51
52 size() {
53     return this.heap.length;
54 }
55
56 _siftUp() {
57     let i = this.heap.length - 1;
58     const item = this.heap[i];
59     while (i > 0) {
60         let parent = Math.floor((i - 1) / 2);
61         if (this.heap[parent][0] <= item[0]) break;
62         this.heap[i] = this.heap[parent];
63         i = parent;
64     }
65     this.heap[i] = item;
66 }
67
68 _siftDown() {
69     let i = 0;
70     const item = this.heap[i];
71     const n = this.heap.length;
72     while (true) {
73         let left = 2 * i + 1;
74         let right = 2 * i + 2;
75         let smallest = i;
76         if (left < n && this.heap[left][0] < this.heap[smallest][0])
77             smallest = left;
78         if (right < n && this.heap[right][0] < this.heap[smallest]
79             [0]) smallest = right;
80         if (smallest === i) break;
81         this.heap[i] = this.heap[smallest];
82         i = smallest;
83     }
84     this.heap[i] = item;
85 }
86
87 // 最短路算法
88 function bfs(grid, lights) {
89     const m = grid.length;
90     const n = grid[0].length;
91
// 起点或终点为障碍物

```

```

92     if (grid[0][0] === 1 || grid[m - 1][n - 1] === 1) {
93         return -1;
94     }
95
96     const directions = [[0,1], [0,-1], [1,0], [-1,0]];
97
98     // 哈希表存储红绿灯等待时间, key = x * n + y
99     const lightMap = new Map();
100    for (const [x, y, t] of lights) {
101        lightMap.set(x * n + y, t);
102    }
103
104    // 最短时间记录数组
105    const visited = Array(m * n).fill(Infinity);
106    const pq = new PriorityQueue(); // 使用自定义最小堆
107    pq.push([0, 0]); // [time, pos]
108    visited[0] = 0;
109
110    while (pq.size() > 0) {
111        const [time, pos] = pq.pop();
112        const x = Math.floor(pos / n);
113        const y = pos % n;
114
115        let nextTime = time + 1;
116
117        // 红绿灯等待处理
118        if (grid[x][y] === 2) {
119            nextTime += lightMap.get(pos) || 0;
120        }
121
122        for (const [dx, dy] of directions) {
123            const nx = x + dx;
124            const ny = y + dy;
125            const npos = nx * n + ny;
126
127            // 越界或为障碍或已有更优路径
128            if (
129                nx < 0 || nx >= m || ny < 0 || ny >= n ||
130                grid[nx][ny] === 1 || visited[npos] <= nextTime
131            ) {
132                continue;
133            }
134
135            visited[npos] = nextTime;
136            pq.push([nextTime, npos]);
137        }
138    }
139

```

```
140     const result = visited[m * n - 1];
141     return result === Infinity ? -1 : result;
142 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8     "strconv"
9     "strings"
10    )
11
12 // 解析输入格式 "[[0,1,0],[0,2,1],[0,0,1]]"
13 func parseInput(s string) [][]int {
14     s = strings.TrimSpace(s)
15     if len(s) <= 4 {
16         return [][]int{}
17     }
18     s = s[2 : len(s)-2]
19     rows := strings.Split(s, "],[")
20     result := [][]int{}
21     for _, row := range rows {
22         nums := strings.Split(row, ",")
23         temp := []int{}
24         for _, num := range nums {
25             n, _ := strconv.Atoi(num)
26             temp = append(temp, n)
27         }
28         result = append(result, temp)
29     }
30     return result
31 }
32
33 // 自定义最小堆，元素为 (time, position)
34 type Item struct {
35     time int
36     pos   int
37 }
38 type PriorityQueue []Item
39
40 func (pq PriorityQueue) Len() int           { return len(pq) }
41 func (pq PriorityQueue) Less(i, j int) bool { return pq[i].time < pq[j].time }
42 func (pq PriorityQueue) Swap(i, j int)      { pq[i], pq[j] = pq[j], pq[i] }
```

```

44     func (pq *PriorityQueue) Push(x any) {
45         *pq = append(*pq, x.(Item))
46     }
47     func (pq *PriorityQueue) Pop() any {
48         old := *pq
49         n := len(old)
50         item := old[n-1]
51         *pq = old[:n-1]
52         return item
53     }
54
55     // 最短路算法
56     func bfs(grid, lights [][]int) int {
57         m, n := len(grid), len(grid[0])
58         if grid[0][0] == 1 || grid[m-1][n-1] == 1 {
59             return -1
60         }
61         directions := [][]int{{0, 1}, {0, -1}, {1, 0}, {-1, 0}}
62
63         // 哈希表存储红绿灯时间
64         lightMap := map[int]int{}
65         for _, light := range lights {
66             x, y, t := light[0], light[1], light[2]
67             pos := x*n + y
68             lightMap[pos] = t
69         }
70
71         visited := make([]int, m*n)
72         for i := range visited {
73             visited[i] = 1<<31 - 1
74         }
75         pq := &PriorityQueue{}
76         heap.Init(pq)
77         heap.Push(pq, Item{0, 0})
78         visited[0] = 0
79
80         for pq.Len() > 0 {
81             item := heap.Pop(pq).(Item)
82             t, pos := item.time, item.pos
83             x, y := pos/n, pos%n
84             nextTime := t + 1
85
86             // 红绿灯等待时间
87             if grid[x][y] == 2 {
88                 nextTime += lightMap[pos]
89             }
90
91             for _, d := range directions {

```

```
92     nx, ny := x+d[0], y+d[1]
93     npos := nx*n + ny
94     // 越界 障碍物 之前到达过并且距离更小
95     if nx < 0 || nx >= m || ny < 0 || ny >= n || grid[nx][ny] == 1 || v
96     isited[npos] <= nextTime {
97         continue
98     }
99     visited[npos] = nextTime
100    heap.Push(pq, Item{nextTime, npos})
101 }
102 }

103 res := visited[m*n-1]
104 if res == 1<<31-1 {
105     return -1
106 }
107 return res
108 }
109

110 func main() {
111     reader := bufio.NewReader(os.Stdin)
112     gridStr, _ := reader.ReadString('\n')
113     lightStr, _ := reader.ReadString('\n')
114     grid := parseInput(gridStr)
115     lights := parseInput(lightStr)
116     fmt.Println(bfs(grid, lights))
117 }
```

| 来自: 华为OD机考 2025C卷 – 网格红绿灯最短路径 (C++ & Python & JAVA & JS & GO)–CSDN博客

华为OD机试 2025C卷 - 运输时间 (C++ & Python & JAVA & JS & GO)-CSDN博客

运输时间

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

M ($1 \leq M \leq 20$) 辆车需要在一条不能超车的单行道到达终点，起点到终点的距离为 N ($1 \leq N \leq 400$)。

速度快的车追上前车后，只能以前车的速度继续行驶，求最后一辆车到达目的地花费的时间。

注：每辆车固定间隔 1 小时出发，比如第一辆车 0 时出发，第二辆车 1 时出发，依次类推

输入描述

第一行两个数字： M N ，分别代表车辆数和到终点的距离，以空格分隔

接下来 M 行，每行一个数字 S ，代表每辆车的速度。 $0 < S < 30$

输出描述

最后一辆车到达目的地花费的时间

用例1

输入

```
1 2 11
2 3
3 2
```

输出

```
1 5.5
```

说明

1 2辆车，距离11，0时出发的车速度快，1时出发的车，到达目的地花费5.5

题解

思路: 模拟

理解两个概念就比较容易了:

- 如果后车正常行驶情况下, 比前车更早到达, 则会被前车阻碍, 此时后车到达终点时刻, 和前车一致。
- 如果后车正常行驶情况下, 比前车更晚到达, 则不会被前车阻碍, 此时后车到达终点时刻, 就是自己正常行驶到达终点的时刻

转换之后本题要求的是: 最后一台车到达目的地花费的时间 = 到达时刻 - 出发时刻。

| 结果最多保留三位小数, 需要去除小数点后的0.

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include <iomanip>
8 using namespace std;
9
10 int main() {
11     int m,n;
12     cin >> m >> n;
13     // 后车不能早于前车
14     double res = 0;
15     for (int i = 0; i < m; i++) {
16         double speed;
17         cin >> speed;
18         // (n / speed + i) 以自己速度到终点的时间 + 出发时间
19         res = max(res, n / speed + i);
20     }
21
22     // 最后一台车的时间
23     res = res - (m - 1);
24
25     auto formatOutput = [] (double num) {
26         ostringstream oss;
27         oss << fixed << setprecision(3) << num; // 先保留 3 位小数
28         string result = oss.str();
29
30         // 自动去掉多余的 .000、.100 之类的
31         result.erase(result.find_last_not_of('0') + 1, string::npos);
32         if (result.back() == '.') result.pop_back(); // 去掉末尾的点
33
34         return result;
35     };
36
37     cout << formatOutput(res) << endl;
38
39     return 0;
40 }
```

JAVA

```
1 import java.util.Scanner;
2 import java.text.NumberFormat;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner scanner = new Scanner(System.in);
7
8         int m = scanner.nextInt();
9         int n = scanner.nextInt();
10
11         double res = 0;
12
13         for (int i = 0; i < m; i++) {
14             // 当前车到达终点的时刻
15             // * 当前车如果比前车更早到达，则被前车阻碍，按前车到达时间算
16             // * 当前车如果比前车更晚到达，则不被前车阻碍，按后车到达时间算
17             double speed = scanner.nextDouble();
18             res = Math.max(res, n / speed + i);
19         }
20
21         // 最后一台车的时间
22         res = res - (m - 1);
23
24         // 格式化打印小数
25         NumberFormat nf = NumberFormat.getInstance();
26         nf.setMinimumFractionDigits(0); // 没有小数位则不留
27         nf.setMaximumFractionDigits(3); // 有小数位则至多保留3位
28
29         System.out.println(nf.format(res));
30         scanner.close();
31     }
32
33 }
34 }
```

Python

```
1 def format_output(num):
2     formatted = f"{num:.3f}" # 保留 3 位小数
3     return formatted.rstrip('0').rstrip('.') # 去掉多余的 0 和小数点
4
5 def main():
6     m, n = map(int, input().split())
7
8     res = 0
9
10    for i in range(m):
11        speed = float(input())
12        res = max(res, n / speed + i)
13
14    # 最后一台车的时间
15    res -= (m - 1)
16
17    print(format_output(res))
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 let inputLines = [];
9 rl.on('line', (line) => {
10     inputLines.push(line);
11 }).on('close', () => {
12     const [m, n] = inputLines[0].split(' ').map(Number);
13     let res = 0;
14
15     for (let i = 1; i <= m; i++) {
16         let speed = parseFloat(inputLines[i]);
17         res = Math.max(res, n / speed + (i - 1));
18     }
19
20     // 最后一台车的时间
21     res -= (m - 1);
22
23     console.log(formatOutput(res));
24 });
25
26 function formatOutput(num) {
27     return parseFloat(num.toFixed(3)).toString(); // 保留最多3位小数，去掉多余的0
28 }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 创建标准输入读取器
13     scanner := bufio.NewScanner(os.Stdin)
14     scanner.Scan()
15
16     // 读取第一行，解析 m 和 n
17     firstLine := strings.Split(scanner.Text(), " ")
18     m, _ := strconv.Atoi(firstLine[0])
19     n, _ := strconv.Atoi(firstLine[1])
20
21     res := 0.0
22
23     // 读取每一行速度
24     for i := 0; i < m; i++ {
25         scanner.Scan()
26         speed, _ := strconv.ParseFloat(scanner.Text(), 64)
27         res = max(res, float64(n)/speed+float64(i))
28     }
29
30     // 最后一台车的时间
31     res -= float64(m - 1)
32
33     // 输出格式化结果（最多 3 位小数，去掉无意义的 .000）
34     fmt.Println(formatOutput(res))
35 }
36
37 // 计算最大值
38 func max(a, b float64) float64 {
39     if a > b {
40         return a
41     }
42     return b
43 }
44
45 // 格式化输出，最多保留 3 位小数，去掉无意义的 .000
```

```
46 func formatOutput(num float64) string {
47     str := fmt.Sprintf("%.3f", num)
48     str = strings.TrimRight(str, "0") // 去掉末尾的 0
49     if strings.HasSuffix(str, ".") { // 去掉小数点
50         str = str[:len(str)-1]
51     }
52     return str
53 }
```

| 来自: 华为OD机试 2025C卷 – 运输时间 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 计算误码率 (C++ & Python & JAVA & JS & GO)-CSDN博客

计算误码率

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

误码率是最常用的[数据通信](#)传输质量指标。它可以理解为“在多少位数据中出现一位差错”。

移动通信网络中的误码率主要是指比特误码率，其计算公式如下：比特误码率=错误比特数/传输总比特数，

为了简单，我们使用字符串来标识通信的信息，一个字符错误了，就认为出现了一个误码

输入一个标准的字符串，和一个传输后的字符串，计算误码率

字符串会被压缩，

例：“2A3B4D5X1Z”表示“AABBBD。。。XXXXZ”

用例会保证两个输入字符串解压后长度一致，解压前的长度不一定一致

每个生成后的字符串长度<100000000。

输入描述

两行，分别为两种字符串的压缩形式。

每行字符串（压缩后的）长度<100000

输出描述

一行，错误的字等数量/展开后的总长度

备注

注意：展开后的字符串不含数字

用例1

输入

```
▼ Plain Text |  
1 3A3B  
2 2A4B
```

输出

```
Plain Text |  
1 1/6
```

用例2

输入

```
Plain Text |  
1 5Y5Z  
2 5Y5Z
```

输出

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

用例3

输入

```
Plain Text |  
1 4Y5Z  
2 9Y
```

输出

```
Plain Text |  
1 5/9
```

题解

思路： 模拟 / 双指针

- 将输入的字符串按照 `字符 : 数量` 格式按顺序分割并保存。推荐使用链表、列表的容器数据结构保存，便于头插，效率高。如果使用语言没自带链表的容器，可以使用数组保存后续使用双指针进行遍历。

2. 循环 / 双指针遍历两个数组，统计相同字符数量和不同字符数量
3. 输出误码率。

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 // 解析字符串
14 list<pair<char, int>> splitStr(string s) {
15     list<pair<char, int>> res;
16     string num = "";
17     for (char c : s) {
18         if ('0' <= c && c <= '9') {
19             num += c;
20         } else {
21             res.push_back({c, stoi(num)});
22             num = "";
23         }
24     }
25     return res;
26 }
27
28 int main() {
29     string s1,s2;
30     getline(cin, s1);
31     getline(cin,s2);
32
33     list<pair<char, int>> list1 = splitStr(s1);
34     list<pair<char, int>> list2 = splitStr(s2);
35
36     int diff = 0;
37     int same = 0;
38     while (!list1.empty()) {
39         pair<char, int> p1 = list1.front();
40         list1.pop_front();
41         pair<char, int> p2 = list2.front();
42         list2.pop_front();
43         // 本次比较的数量
44         int compareNum = min(p1.second, p2.second);
45     }
}
```

```
46     if (p1.first == p2.first) {
47         same += compareNum;
48     } else {
49         diff += compareNum;
50     }
51     // 多余数量重新插入下次再进行比较
52     if (p1.second > compareNum) {
53         list1.push_front({p1.first, p1.second - compareNum});
54     }
55
56     if (p2.second > compareNum) {
57         list2.push_front({p2.first, p2.second - compareNum});
58     }
59 }
60 cout << to_string(diff) + "/" + to_string(diff + same);
61
62     return 0;
63 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 分割字符串
5     static LinkedList<Pair> splitStr(String s) {
6         LinkedList<Pair> res = new LinkedList<>();
7         StringBuilder num = new StringBuilder();
8         for (char c : s.toCharArray()) {
9             if (Character.isDigit(c)) {
10                 num.append(c);
11             } else {
12                 res.add(new Pair(c, Integer.parseInt(num.toString())));
13                 num.setLength(0);
14             }
15         }
16         return res;
17     }
18
19     static class Pair {
20         char ch;
21         int count;
22         Pair(char ch, int count) {
23             this.ch = ch;
24             this.count = count;
25         }
26     }
27
28     public static void main(String[] args) {
29         Scanner sc = new Scanner(System.in);
30         String s1 = sc.nextLine();
31         String s2 = sc.nextLine();
32
33         LinkedList<Pair> list1 = splitStr(s1);
34         LinkedList<Pair> list2 = splitStr(s2);
35
36         int diff = 0;
37         int same = 0;
38         // 循环对比
39         while (!list1.isEmpty()) {
40             Pair p1 = list1.pollFirst();
41             Pair p2 = list2.pollFirst();
42             int compareNum = Math.min(p1.count, p2.count);
43
44             if (p1.ch == p2.ch) {
45                 same += compareNum;
```

```
46         } else {
47             diff += compareNum;
48         }
49
50         if (p1.count > compareNum) {
51             list1.addFirst(new Pair(p1.ch, p1.count - compareNum));
52         }
53         if (p2.count > compareNum) {
54             list2.addFirst(new Pair(p2.ch, p2.count - compareNum));
55         }
56     }
57
58     System.out.println(diff + "/" + (diff + same));
59 }
60 }
```

Python

```
1 from collections import deque
2 import sys
3
4 # 解析压缩字符串，返回 deque 结构的（字符， 次数）
5 def parse_string(s):
6     res = deque()
7     num = ""
8     for c in s:
9         if c.isdigit():
10             num += c
11         else:
12             res.append((c, int(num)))
13             num = ""
14     return res
15
16 s1 = sys.stdin.readline().strip()
17 s2 = sys.stdin.readline().strip()
18
19 list1 = parse_string(s1)
20 list2 = parse_string(s2)
21
22 diff = 0
23 same = 0
24
25 while list1:
26     ch1, cnt1 = list1.popleft()
27     ch2, cnt2 = list2.popleft()
28     compare_num = min(cnt1, cnt2)
29
30     if ch1 == ch2:
31         same += compare_num
32     else:
33         diff += compare_num
34
35     # 将剩余字符重新插入队头
36     if cnt1 > compare_num:
37         list1.appendleft((ch1, cnt1 - compare_num))
38     if cnt2 > compare_num:
39         list2.appendleft((ch2, cnt2 - compare_num))
40
41 print(f"{diff}/{diff + same}")
```

JavaScript

```
1 const readline = require('readline');
2 // 解析压缩字符串， 按顺序存储(字符， 次数)结构
3 function parseString(s) {
4     let res = [];
5     let num = '';
6     for (let c of s) {
7         if (/^\d/.test(c)) {
8             num += c;
9         } else {
10            res.push([c, parseInt(num)]);
11            num = '';
12        }
13    }
14    return res;
15 }
16
17 const rl = readline.createInterface({ input: process.stdin, output: process.stdout });
18 const lines = [];
19
20 rl.on('line', line => {
21     lines.push(line);
22     if (lines.length === 2) {
23         const list1 = parseString(lines[0]);
24         const list2 = parseString(lines[1]);
25
26         let i = 0, j = 0;
27         let cnt1 = list1[0][1], cnt2 = list2[0][1];
28         let ch1 = list1[0][0], ch2 = list2[0][0];
29         let same = 0, diff = 0;
30         // 双指针循环对比
31         while (i < list1.length && j < list2.length) {
32             const compareNum = Math.min(cnt1, cnt2);
33
34             if (ch1 === ch2) same += compareNum;
35             else diff += compareNum;
36
37             cnt1 -= compareNum;
38             cnt2 -= compareNum;
39
40             if (cnt1 === 0) {
41                 i++;
42                 if (i < list1.length) {
43                     ch1 = list1[i][0];
44                     cnt1 = list1[i][1];
45                 }
46             }
47         }
48     }
49 }
50
51 rl.on('close', () => {
52     console.log(`same: ${same}, diff: ${diff}`);
53     process.exit();
54 })
```

```
45         }
46     }
47     if (cnt2 === 0) {
48         j++;
49         if (j < list2.length) {
50             ch2 = list2[j][0];
51             cnt2 = list2[j][1];
52         }
53     }
54 }
55 // 计算误码率
56 console.log(` ${diff}/${diff + same}`);
57 rl.close();
58 }
59});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "unicode"
9 )
10
11 // 解析压缩字符串，返回 [字符, 次数] 的切片
12 func parse(s string) []interface{} {
13     var res []interface{}
14     num := ""
15     for _, c := range s {
16         if unicode.IsDigit(c) {
17             num += string(c)
18         } else {
19             n, _ := strconv.Atoi(num)
20             res = append(res, interface{}{c, n})
21             num = ""
22         }
23     }
24     return res
25 }
26
27 func main() {
28     reader := bufio.NewReader(os.Stdin)
29     s1, _ := reader.ReadString('\n')
30     s2, _ := reader.ReadString('\n')
31
32     s1 = s1[:len(s1)-1]
33     s2 = s2[:len(s2)-1]
34
35     list1 := parse(s1)
36     list2 := parse(s2)
37
38     i, j := 0, 0
39     cnt1 := list1[0][1].(int)
40     cnt2 := list2[0][1].(int)
41     ch1 := list1[0][0].(rune)
42     ch2 := list2[0][0].(rune)
43     same, diff := 0, 0
44     // 双指针循环对比
45     for i < len(list1) && j < len(list2) {
```

```

46     compare := min(cnt1, cnt2)
47     if ch1 == ch2 {
48         same += compare
49     } else {
50         diff += compare
51     }
52     cnt1 -= compare
53     cnt2 -= compare
54
55     if cnt1 == 0 {
56         i++
57         if i < len(list1) {
58             ch1 = list1[i][0].(rune)
59             cnt1 = list1[i][1].(int)
60         }
61     }
62     if cnt2 == 0 {
63         j++
64         if j < len(list2) {
65             ch2 = list2[j][0].(rune)
66             cnt2 = list2[j][1].(int)
67         }
68     }
69 }
70 // 计算误码率
71 fmt.Printf("%d/%d\n", diff, diff+same)
72 }
73
74 func min(a, b int) int {
75     if a < b {
76         return a
77     }
78     return b
79 }
```

计算误码率

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

误码率是最常用的**数据通信**传输质量指标。它可以理解为“在多少位数据中出现一位差错”。

移动通信网络中的误码率主要是指**比特误码率**，其计算公式如下：**比特误码率=错误比特数/传输总比特数**，

为了简单，我们使用字符串来标识通信的信息，一个字符错误了，就认为出现了一个误码

输入一个标准的字符串，和一个传输后的字符串，计算误码率

字符串会被压缩，

例：“2A3B4D5X1Z”表示“AABBDDDDXXXXZ”

用例会保证两个输入字符串解压后长度一致，解压前的长度不一定一致

每个生成后的字符串长度<100000000。

输入描述

两行，分别为两种字符串的压缩形式。

每行字符串（压缩后的）长度<100000

输出描述

一行，错误的字等数量/展开后的总长度

备注

注意：展开后的字符串不含数字

用例1

输入

▼ Plain Text |

1	3A3B
2	2A4B

输出

▼ Plain Text |

1	1/6
---	-----

用例2

输入

▼ Plain Text |

1	5Y5Z
2	5Y5Z

输出

Plain Text |

1 0/10

用例3

输入

Plain Text |

1 4Y5Z
2 9Y

输出

Plain Text |

1 5/9

题解

思路： 模拟 / 双指针

1. 将输入的字符串按照 `字符 : 数量` 格式按顺序分割并保存。推荐使用链表、列表的容器数据结构保存，便于头插，效率高。如果使用语言没自带链表的容器，可以使用数组保存后续使用双指针进行遍历。
2. 循环/ 双指针遍历两个数组，统计相同字符数量和不同字符数量
3. 输出误码率。

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 // 解析字符串
14 list<pair<char, int>> splitStr(string s) {
15     list<pair<char, int>> res;
16     string num = "";
17     for (char c : s) {
18         if ('0' <= c && c <= '9') {
19             num += c;
20         } else {
21             res.push_back({c, stoi(num)});
22             num = "";
23         }
24     }
25     return res;
26 }
27
28 int main() {
29     string s1,s2;
30     getline(cin, s1);
31     getline(cin,s2);
32
33     list<pair<char, int>> list1 = splitStr(s1);
34     list<pair<char, int>> list2 = splitStr(s2);
35
36     int diff = 0;
37     int same = 0;
38     while (!list1.empty()) {
39         pair<char, int> p1 = list1.front();
40         list1.pop_front();
41         pair<char, int> p2 = list2.front();
42         list2.pop_front();
43         // 本次比较的数量
44         int compareNum = min(p1.second, p2.second);
45     }
}
```

```
46     if (p1.first == p2.first) {
47         same += compareNum;
48     } else {
49         diff += compareNum;
50     }
51     // 多余数量重新插入下次再进行比较
52     if (p1.second > compareNum) {
53         list1.push_front({p1.first, p1.second - compareNum});
54     }
55
56     if (p2.second > compareNum) {
57         list2.push_front({p2.first, p2.second - compareNum});
58     }
59 }
60 cout << to_string(diff) + "/" + to_string(diff + same);
61
62     return 0;
63 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 分割字符串
5     static LinkedList<Pair> splitStr(String s) {
6         LinkedList<Pair> res = new LinkedList<>();
7         StringBuilder num = new StringBuilder();
8         for (char c : s.toCharArray()) {
9             if (Character.isDigit(c)) {
10                 num.append(c);
11             } else {
12                 res.add(new Pair(c, Integer.parseInt(num.toString())));
13                 num.setLength(0);
14             }
15         }
16         return res;
17     }
18
19     static class Pair {
20         char ch;
21         int count;
22         Pair(char ch, int count) {
23             this.ch = ch;
24             this.count = count;
25         }
26     }
27
28     public static void main(String[] args) {
29         Scanner sc = new Scanner(System.in);
30         String s1 = sc.nextLine();
31         String s2 = sc.nextLine();
32
33         LinkedList<Pair> list1 = splitStr(s1);
34         LinkedList<Pair> list2 = splitStr(s2);
35
36         int diff = 0;
37         int same = 0;
38         // 循环对比
39         while (!list1.isEmpty()) {
40             Pair p1 = list1.pollFirst();
41             Pair p2 = list2.pollFirst();
42             int compareNum = Math.min(p1.count, p2.count);
43
44             if (p1.ch == p2.ch) {
45                 same += compareNum;
```

```
46         } else {
47             diff += compareNum;
48         }
49
50         if (p1.count > compareNum) {
51             list1.addFirst(new Pair(p1.ch, p1.count - compareNum));
52         }
53         if (p2.count > compareNum) {
54             list2.addFirst(new Pair(p2.ch, p2.count - compareNum));
55         }
56     }
57
58     System.out.println(diff + "/" + (diff + same));
59 }
60 }
```

Python

```
1 from collections import deque
2 import sys
3
4 # 解析压缩字符串，返回 deque 结构的（字符， 次数）
5 def parse_string(s):
6     res = deque()
7     num = ""
8     for c in s:
9         if c.isdigit():
10             num += c
11         else:
12             res.append((c, int(num)))
13             num = ""
14     return res
15
16 s1 = sys.stdin.readline().strip()
17 s2 = sys.stdin.readline().strip()
18
19 list1 = parse_string(s1)
20 list2 = parse_string(s2)
21
22 diff = 0
23 same = 0
24
25 while list1:
26     ch1, cnt1 = list1.popleft()
27     ch2, cnt2 = list2.popleft()
28     compare_num = min(cnt1, cnt2)
29
30     if ch1 == ch2:
31         same += compare_num
32     else:
33         diff += compare_num
34
35     # 将剩余字符重新插入队头
36     if cnt1 > compare_num:
37         list1.appendleft((ch1, cnt1 - compare_num))
38     if cnt2 > compare_num:
39         list2.appendleft((ch2, cnt2 - compare_num))
40
41 print(f"{diff}/{diff + same}")
```

JavaScript

```
1 const readline = require('readline');
2 // 解析压缩字符串， 按顺序存储(字符， 次数)结构
3 function parseString(s) {
4     let res = [];
5     let num = '';
6     for (let c of s) {
7         if (/^\d/.test(c)) {
8             num += c;
9         } else {
10            res.push([c, parseInt(num)]);
11            num = '';
12        }
13    }
14    return res;
15 }
16
17 const rl = readline.createInterface({ input: process.stdin, output: process.stdout });
18 const lines = [];
19
20 rl.on('line', line => {
21     lines.push(line);
22     if (lines.length === 2) {
23         const list1 = parseString(lines[0]);
24         const list2 = parseString(lines[1]);
25
26         let i = 0, j = 0;
27         let cnt1 = list1[0][1], cnt2 = list2[0][1];
28         let ch1 = list1[0][0], ch2 = list2[0][0];
29         let same = 0, diff = 0;
30         // 双指针循环对比
31         while (i < list1.length && j < list2.length) {
32             const compareNum = Math.min(cnt1, cnt2);
33
34             if (ch1 === ch2) same += compareNum;
35             else diff += compareNum;
36
37             cnt1 -= compareNum;
38             cnt2 -= compareNum;
39
40             if (cnt1 === 0) {
41                 i++;
42                 if (i < list1.length) {
43                     ch1 = list1[i][0];
44                     cnt1 = list1[i][1];
45                 }
46             }
47         }
48     }
49 }
50
51 rl.on('close', () => {
52     console.log(`same: ${same}, diff: ${diff}`);
53     process.exit();
54 })
55
56 rl.on('error', (err) => {
57     console.error(err);
58     process.exit();
59 })
```

```
45         }
46     }
47     if (cnt2 === 0) {
48         j++;
49         if (j < list2.length) {
50             ch2 = list2[j][0];
51             cnt2 = list2[j][1];
52         }
53     }
54 }
55 // 计算误码率
56 console.log(` ${diff}/${diff + same}`);
57 rl.close();
58 }
59});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "unicode"
9 )
10
11 // 解析压缩字符串，返回 [字符, 次数] 的切片
12 func parse(s string) []interface{} {
13     var res []interface{}
14     num := ""
15     for _, c := range s {
16         if unicode.IsDigit(c) {
17             num += string(c)
18         } else {
19             n, _ := strconv.Atoi(num)
20             res = append(res, interface{}{c, n})
21             num = ""
22         }
23     }
24     return res
25 }
26
27 func main() {
28     reader := bufio.NewReader(os.Stdin)
29     s1, _ := reader.ReadString('\n')
30     s2, _ := reader.ReadString('\n')
31
32     s1 = s1[:len(s1)-1]
33     s2 = s2[:len(s2)-1]
34
35     list1 := parse(s1)
36     list2 := parse(s2)
37
38     i, j := 0, 0
39     cnt1 := list1[0][1].(int)
40     cnt2 := list2[0][1].(int)
41     ch1 := list1[0][0].(rune)
42     ch2 := list2[0][0].(rune)
43     same, diff := 0, 0
44     // 双指针循环对比
45     for i < len(list1) && j < len(list2) {
```

```
46     compare := min(cnt1, cnt2)
47     if ch1 == ch2 {
48         same += compare
49     } else {
50         diff += compare
51     }
52     cnt1 -= compare
53     cnt2 -= compare
54
55     if cnt1 == 0 {
56         i++
57         if i < len(list1) {
58             ch1 = list1[i][0].(rune)
59             cnt1 = list1[i][1].(int)
60         }
61     }
62     if cnt2 == 0 {
63         j++
64         if j < len(list2) {
65             ch2 = list2[j][0].(rune)
66             cnt2 = list2[j][1].(int)
67         }
68     }
69 }
70 // 计算误码率
71 fmt.Printf("%d/%d\n", diff, diff+same)
72 }
73
74 func min(a, b int) int {
75     if a < b {
76         return a
77     }
78     return b
79 }
```

| 来自: 华为OD机考 2025C卷 – 计算误码率 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考 2025C卷 – 计算误码率 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 数字螺旋矩阵 (C++ & Python & JAVA & JS & GO)-CSDN博客

数字螺旋矩阵

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

疫情期间，小明隔离在家，百无聊赖，在纸上写数字玩。他发明了一种写法：

给出数字个数n和行数m ($0 < n \leq 999$, $0 < m \leq 999$)，从左上角的1开始，按照顺时针螺旋向内写方式，依次写出2,3...n，最终形成一个m行矩阵。

小明对这个矩阵有些要求：

- 每行数字的个数一样多
- 列的数量尽可能少
- 填充数字时优先填充外部
- 数字不够时，使用单个*号占位

输入描述

输入一行，两个整数，空格隔开，依次表示n、m

输出描述

符合要求的唯一矩阵

示例1

输入

```
▼ Plain Text |  
1 9 4
```

输出

▼

Plain Text |

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

说明

| 9个数字写成4行，最少需要3列

示例2

输入

```
▼
Plain Text |
1 3 5
```

输出

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

说明

| 3个数字写5行，只有一列，数字不够用*号填充

示例3

输入

```
▼
Plain Text |
1 120 7
```

输出

```

1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
2 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 19
3 45 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 63 20
4 44 83 114 115 116 117 118 119 120 * * * * * 99 64 21
5 43 82 113 112 111 110 109 108 107 106 105 104 103 102 101 100 65 22
6 42 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 23
7 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24

```

题解

思路： 模拟

- 根据输入的 `n m` 来计算出需要的列 `col = (num + m - 1) / m`, 向上取整。
- 定义个 `matrix[m][col]` 数组初始的话可以将数组所有元素默认设置为0, 然后按照螺旋的方式将数字添入数组中就行, 一直循环直到填充完num个数。具体模拟逻辑如下:
 - 可以定义 `top bottom left right` 代表本次循环的四个边界。
 - 根据规律螺旋过程对应四次转弯, 发生在碰到对应边界时, 并更新对应需要的边界范围。
 - 初始从左至右, 碰到右边界, 更新上边界范围。转向从上到下
 - 碰到下边界时, 更新右边界, 转向从右至左。
 - 碰到左边界时, 更新下边界, 转向从下到上。
 - 碰到上边界时, 更新左边界, 转向从左至右。
 - 重复上述逻辑直到将n个数全部填充完毕为止。
- 执行完上述填充操作之后, 按行进行输出填充后的数组, 对应位置为0输出 `*`

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <sstream>
4
5 int main() {
6     int n, m;
7     std::cin >> n >> m; // 读取 n 和 m
8     int cols = (n + m - 1) / m; // 计算列数 向上取整
9
10    std::vector<std::vector<int>> matrix(m, std::vector<int>(cols)); // 定义矩阵
11    int num = 1, top = 0, bottom = m - 1, left = 0, right = cols - 1;
12
13    while (num <= n) {
14        for (int i = left; i <= right && num <= n; i++) matrix[top][i] = num++;
15        top++;
16        for (int i = top; i <= bottom && num <= n; i++) matrix[i][right] = num++;
17        right--;
18        for (int i = right; i >= left && num <= n; i--) matrix[bottom][i] = num++;
19        bottom--;
20        for (int i = bottom; i >= top && num <= n; i--) matrix[i][left] = num++;
21        left++;
22    }
23
24    std::ostringstream output;
25    for (int i = 0; i < m; i++) {
26        for (int j = 0; j < cols; j++) {
27            if (matrix[i][j] == 0) {
28                output << '*';
29            } else {
30                output << matrix[i][j];
31            }
32            if (j < cols - 1) output << ' ';
33        }
34        output << '\n'; // 换行
35    }
36
37    std::cout << output.str(); // 一次性输出, 提高性能
38    return 0;
39 }
```

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(); // 读取 n
7         int m = scanner.nextInt(); // 读取 m
8         scanner.close(); // 关闭输入流
9
10        int cols = (n + m - 1) / m; // 计算列数, 向上取整
11        int[][] matrix = new int[m][cols]; // 定义矩阵
12        // 初始值
13        for (int i = 0; i < m; i++) {
14            for (int j = 0; j < cols; j++) {
15                matrix[i][j] = -1;
16            }
17        }
18
19        int num = 1, top = 0, bottom = m - 1, left = 0, right = cols - 1;
20
21        while (num <= n) {
22            for (int i = left; i <= right && num <= n; i++) matrix[top]
23 [i] = num++; // 从左到右
24            top++;
25            for (int i = top; i <= bottom && num <= n; i++) matrix[i][right]
26 = num++; // 从上到下
27            right--;
28            for (int i = right; i >= left && num <= n; i--) matrix[bottom]
29 [i] = num++; // 从右到左
30            bottom--;
31            for (int i = bottom; i >= top && num <= n; i--) matrix[i][lef
32 t] = num++; // 从下到上
33            left++;
34        }
35
36        // 输出矩阵
37        StringBuilder output = new StringBuilder();
38        for (int i = 0; i < m; i++) {
39            for (int j = 0; j < cols; j++) {
40                output.append(matrix[i][j] == -1 ? "*" : matrix[i][j]);
41                if (j < cols - 1) output.append(" ");
42            }
43            output.append("\n");
44        }
45    }
46}
```

格

```
41      }
42      System.out.print(output); // 一次性输出，提高性能
43  }
44 }
```

Python

```
1 import sys
2 import math
3
4 # 读取输入
5 n, m = map(int, sys.stdin.readline().split())
6
7 cols = (n + m - 1) // m # 计算列数, 向上取整
8 matrix = [[0] * cols for _ in range(m)] # 定义矩阵
9
10 num, top, bottom, left, right = 1, 0, m - 1, 0, cols - 1
11
12 while num <= n:
13     for i in range(left, right + 1):
14         if num > n: break
15         matrix[top][i] = num
16         num += 1
17     top += 1
18     for i in range(top, bottom + 1):
19         if num > n: break
20         matrix[i][right] = num
21         num += 1
22     right -= 1
23     for i in range(right, left - 1, -1):
24         if num > n: break
25         matrix[bottom][i] = num
26         num += 1
27     bottom -= 1
28     for i in range(bottom, top - 1, -1):
29         if num > n: break
30         matrix[i][left] = num
31         num += 1
32     left += 1
33
34 # 输出矩阵
35 output = []
36 for row in matrix:
37     output.append(" ".join(str(cell) if cell != 0 else '*' for cell in row))
38 sys.stdout.write("\n".join(output) + "\n")
```

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", (input) => {
9     const [n, m] = input.split(" ").map(Number);
10    const cols = Math.ceil(n / m); // 计算列数, 向上取整
11    let matrix = Array.from({ length: m }, () => Array(cols).fill(0)); // 定义矩阵
12
13    let num = 1, top = 0, bottom = m - 1, left = 0, right = cols - 1;
14
15    while (num <= n) {
16        for (let i = left; i <= right && num <= n; i++) matrix[top][i] = num++;
17        top++;
18        for (let i = top; i <= bottom && num <= n; i++) matrix[i][right] = num++;
19        right--;
20        for (let i = right; i >= left && num <= n; i--) matrix[bottom][i] = num++;
21        bottom--;
22        for (let i = bottom; i >= top && num <= n; i--) matrix[i][left] = num++;
23        left++;
24    }
25
26    let output = matrix.map(row => row.map(cell => cell === 0 ? '*' : cell).join(" ")).join("\n");
27    console.log(output);
28
29    rl.close();
30});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 读取输入
13     reader := bufio.NewReader(os.Stdin)
14     line, _ := reader.ReadString('\n')
15     fields := strings.Fields(line)
16     n, _ := strconv.Atoi(fields[0])
17     m, _ := strconv.Atoi(fields[1])
18
19     cols := (n + m - 1) / m // 计算列数，向上取整
20     matrix := make([][]int, m)
21     for i := range matrix {
22         matrix[i] = make([]int, cols)
23     }
24
25     num, top, bottom, left, right := 1, 0, m-1, 0, cols-1
26
27     for num <= n {
28         // 右移
29         for i := left; i <= right && num <= n; i++ {
30             matrix[top][i] = num
31             num++
32         }
33         top++
34         // 下移
35         for i := top; i <= bottom && num <= n; i++ {
36             matrix[i][right] = num
37             num++
38         }
39         // 左移
40         right--
41         for i := right; i >= left && num <= n; i-- {
42             matrix[bottom][i] = num
43             num++
44         }
45         bottom--
}
```

```

46     // 上移
47     for i := bottom; i >= top && num <= n; i-- {
48         matrix[i][left] = num
49         num++
50     }
51     left++
52 }
53
54 // 输出矩阵
55 var output strings.Builder
56 for i := 0; i < m; i++ {
57     for j := 0; j < cols; j++ {
58         if matrix[i][j] == 0 {
59             output.WriteString("*")
60         } else {
61             output.WriteString(strconv.Itoa(matrix[i][j]))
62         }
63         if j < cols-1 {
64             output.WriteString(" ") // 避免末尾空格
65         }
66     }
67     output.WriteString("\n")
68 }
69 fmt.Println(output.String()) // 一次性输出，提高性能
70 }

```

数字螺旋矩阵

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

疫情期间，小明隔离在家，百无聊赖，在纸上写数字玩。他发明了一种写法：

给出数字个数n和行数m ($0 < n \leq 999$, $0 < m \leq 999$)，从左上角的1开始，按照顺时针螺旋向内写方式，依次写出2,3...n，最终形成一个m行矩阵。

小明对这个矩阵有些要求：

- 每行数字的个数一样多
- 列的数量尽可能少
- 填充数字时优先填充外部
- 数字不够时，使用单个*号占位

输入描述

输入一行，两个整数，空格隔开，依次表示n、m

输出描述

符合要求的唯一矩阵

示例1

输入

▼ Plain Text |

```
1 9 4
```

输出

▼ Plain Text |

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

说明

| 9个数字写成4行，最少需要3列

示例2

输入

▼ Plain Text |

```
1 3 5
```

输出

▼ Plain Text |

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

说明

3个数字写5行，只有一列，数字不够用*号填充

示例3

输入

```
Plain Text  
1 120 7
```

输出

```
Plain Text  
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18  
2 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 19  
3 45 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 63 20  
4 44 83 114 115 116 117 118 119 120 * * * * * 99 64 21  
5 43 82 113 112 111 110 109 108 107 106 105 104 103 102 101 100 65 22  
6 42 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 23  
7 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24
```

题解

思路： 模拟

- 根据输入的 `n m` 来计算出需要的列 `col = (num + m - 1) / m`，向上取整。
- 定义个 `matrix[m][col]` 数组初始的话可以将数组所有元素默认设置为0，然后按照螺旋的方式将数字添入数组中就行，一直循环直到填充完num个数。具体模拟逻辑如下：
 - 可以定义 `top bottom left right` 代表本次循环的四个边界。
 - 根据规律螺旋过程对应四次转弯，发生在碰到对应边界时，并更新对应需要的边界范围。
 - 初始从左至右，碰到右边界，更新上边界范围。转向从上到下
 - 碰到下边界时，更新右边界，转向从右至左。
 - 碰到左边界时，更新下边界，转向从下到上。
 - 碰到上边界时，更新左边界，转向从左至右。
 - 重复上述逻辑直到将n个数全部填充完毕为止。
- 执行完上述填充操作之后，按行进行输出填充后的数组，对应位置为0输出 *

C++

```
1 #include <iostream>
2 #include <vector>
3 #include <sstream>
4
5 int main() {
6     int n, m;
7     std::cin >> n >> m; // 读取 n 和 m
8     int cols = (n + m - 1) / m; // 计算列数 向上取整
9
10    std::vector<std::vector<int>> matrix(m, std::vector<int>(cols)); // 定义矩阵
11    int num = 1, top = 0, bottom = m - 1, left = 0, right = cols - 1;
12
13    while (num <= n) {
14        for (int i = left; i <= right && num <= n; i++) matrix[top][i] = num++;
15        top++;
16        for (int i = top; i <= bottom && num <= n; i++) matrix[i][right] = num++;
17        right--;
18        for (int i = right; i >= left && num <= n; i--) matrix[bottom][i] = num++;
19        bottom--;
20        for (int i = bottom; i >= top && num <= n; i--) matrix[i][left] = num++;
21        left++;
22    }
23
24    std::ostringstream output;
25    for (int i = 0; i < m; i++) {
26        for (int j = 0; j < cols; j++) {
27            if (matrix[i][j] == 0) {
28                output << '*';
29            } else {
30                output << matrix[i][j];
31            }
32            if (j < cols - 1) output << ' ';
33        }
34        output << '\n'; // 换行
35    }
36
37    std::cout << output.str(); // 一次性输出, 提高性能
38    return 0;
39 }
```

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(); // 读取 n
7         int m = scanner.nextInt(); // 读取 m
8         scanner.close(); // 关闭输入流
9
10        int cols = (n + m - 1) / m; // 计算列数, 向上取整
11        int[][] matrix = new int[m][cols]; // 定义矩阵
12        // 初始值
13        for (int i = 0; i < m; i++) {
14            for (int j = 0; j < cols; j++) {
15                matrix[i][j] = -1;
16            }
17        }
18
19        int num = 1, top = 0, bottom = m - 1, left = 0, right = cols - 1;
20
21        while (num <= n) {
22            for (int i = left; i <= right && num <= n; i++) matrix[top]
23 [i] = num++; // 从左到右
24            top++;
25            for (int i = top; i <= bottom && num <= n; i++) matrix[i][right]
26 = num++; // 从上到下
27            right--;
28            for (int i = right; i >= left && num <= n; i--) matrix[bottom]
29 [i] = num++; // 从右到左
30            bottom--;
31            for (int i = bottom; i >= top && num <= n; i--) matrix[i][lef
32 t] = num++; // 从下到上
33            left++;
34        }
35
36        // 输出矩阵
37        StringBuilder output = new StringBuilder();
38        for (int i = 0; i < m; i++) {
39            for (int j = 0; j < cols; j++) {
40                output.append(matrix[i][j] == -1 ? "*" : matrix[i][j]);
41                if (j < cols - 1) output.append(" ");
42            }
43            output.append("\n");
44        }
45    }
46}
```

格

```
41      }
42      System.out.print(output); // 一次性输出，提高性能
43  }
44 }
```

Python

```
1 import sys
2 import math
3
4 # 读取输入
5 n, m = map(int, sys.stdin.readline().split())
6
7 cols = (n + m - 1) // m # 计算列数, 向上取整
8 matrix = [[0] * cols for _ in range(m)] # 定义矩阵
9
10 num, top, bottom, left, right = 1, 0, m - 1, 0, cols - 1
11
12 while num <= n:
13     for i in range(left, right + 1):
14         if num > n: break
15         matrix[top][i] = num
16         num += 1
17     top += 1
18     for i in range(top, bottom + 1):
19         if num > n: break
20         matrix[i][right] = num
21         num += 1
22     right -= 1
23     for i in range(right, left - 1, -1):
24         if num > n: break
25         matrix[bottom][i] = num
26         num += 1
27     bottom -= 1
28     for i in range(bottom, top - 1, -1):
29         if num > n: break
30         matrix[i][left] = num
31         num += 1
32     left += 1
33
34 # 输出矩阵
35 output = []
36 for row in matrix:
37     output.append(" ".join(str(cell) if cell != 0 else '*' for cell in row))
38 sys.stdout.write("\n".join(output) + "\n")
```

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", (input) => {
9     const [n, m] = input.split(" ").map(Number);
10    const cols = Math.ceil(n / m); // 计算列数, 向上取整
11    let matrix = Array.from({ length: m }, () => Array(cols).fill(0)); // 定义矩阵
12
13    let num = 1, top = 0, bottom = m - 1, left = 0, right = cols - 1;
14
15    while (num <= n) {
16        for (let i = left; i <= right && num <= n; i++) matrix[top][i] = num++;
17        top++;
18        for (let i = top; i <= bottom && num <= n; i++) matrix[i][right] = num++;
19        right--;
20        for (let i = right; i >= left && num <= n; i--) matrix[bottom][i] = num++;
21        bottom--;
22        for (let i = bottom; i >= top && num <= n; i--) matrix[i][left] = num++;
23        left++;
24    }
25
26    let output = matrix.map(row => row.map(cell => cell === 0 ? '*' : cell).join(" ")).join("\n");
27    console.log(output);
28
29    rl.close();
30});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 读取输入
13     reader := bufio.NewReader(os.Stdin)
14     line, _ := reader.ReadString('\n')
15     fields := strings.Fields(line)
16     n, _ := strconv.Atoi(fields[0])
17     m, _ := strconv.Atoi(fields[1])
18
19     cols := (n + m - 1) / m // 计算列数，向上取整
20     matrix := make([][]int, m)
21     for i := range matrix {
22         matrix[i] = make([]int, cols)
23     }
24
25     num, top, bottom, left, right := 1, 0, m-1, 0, cols-1
26
27     for num <= n {
28         // 右移
29         for i := left; i <= right && num <= n; i++ {
30             matrix[top][i] = num
31             num++
32         }
33         top++
34         // 下移
35         for i := top; i <= bottom && num <= n; i++ {
36             matrix[i][right] = num
37             num++
38         }
39         // 左移
40         right--
41         for i := right; i >= left && num <= n; i-- {
42             matrix[bottom][i] = num
43             num++
44         }
45         bottom--
}
```

```
46     // 上移
47     for i := bottom; i >= top && num <= n; i-- {
48         matrix[i][left] = num
49         num++
50     }
51     left++
52 }
53
54 // 输出矩阵
55 var output strings.Builder
56 for i := 0; i < m; i++ {
57     for j := 0; j < cols; j++ {
58         if matrix[i][j] == 0 {
59             output.WriteString("*")
60         } else {
61             output.WriteString(strconv.Itoa(matrix[i][j]))
62         }
63         if j < cols-1 {
64             output.WriteString(" ") // 避免末尾空格
65         }
66     }
67     output.WriteString("\n")
68 }
69 fmt.Println(output.String()) // 一次性输出，提高性能
70 }
```

| 来自: 华为OD机考2025C卷 – 数字螺旋矩阵 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考2025C卷 – 数字螺旋矩阵 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 切割字符串 (C++ & Python & JAVA & JS & GO)-CSDN博客

切割字符串

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

给定非空字符串s，将该字符串分割成一些子串，使每个子串的ASCII码值的和均为水仙花数。

- 1、若分割不成功，则返回0；
- 2、若分割成功且分割结果不唯一，则返回-1；
- 3、若分割成功且分割结果唯一，则返回分割后子串的数目。

备注

"水仙花数"是指一个三位数，每位上数字的立方和等于该数字本身，如 371 是'水仙花数'，因为 $371=3^3+7^3+1^3$

输入描述

输入字符串的最大长度为200

输出描述

根据题目描述中情况，返回相应的结果。

用例1

输入

```
1 abc
```

输出

```
1 0
```

说明

| 分割不成功

用例2

输入

▼ Plain Text |

```
1 f3@d5a8
```

输出

▼ Plain Text |

```
1 -1
```

说明

| 分割成功但分割结果不唯一，可以分割为两组，一组'f3'和'@d5a8'，另外一组'f3@d5'和'a8'

用例2

输入

▼ Plain Text |

```
1 AXdddF
```

输出

▼ Plain Text |

```
1 2
```

说明

| 分割成功且分割结果唯一，可以分割'AX'(153)和'dddF'(370)成两个子串

题解

思路： 前缀和 + 递归回溯

1. 预处理计算前缀和方便后续快速计算区间和的值。

2. 使用递归回溯尝试进行切割，使用 `res` 数组存储每种切割方案划分的子串数量。递归的逻辑如下：
 - a. 从当前位置作为起点，向右枚举子串终点，如果[起点，终点]的和能组成水仙花数。子串数量+1，继续向下递归。
 - b. 递归的结束条件为：将字符串全部切割完毕，此时可以获得这种切割方案的子串数量，存入`res`。
3. 输出结果，根据 `res` 数组的长度来进行对应输出即可。

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 // 判断是否为水仙花数
12 bool judge(int num) {
13     if (num < 100 || num > 999) {
14         return false;
15     }
16     int sum = 0;
17     int tmp = num;
18     while (tmp != 0) {
19         int mod = tmp % 10;
20         sum += mod * mod * mod;
21         tmp /= 10;
22     }
23     return sum == num;
24 }
25
26 // 递归回溯
27 void DFS(vector<int>& preSum, int index, int n, int count, vector<int>& res) {
28     // 全部分割完成
29     if (index == n) {
30         res.push_back(count);
31         return;
32     }
33     for (int i = index + 1; i <= n; i++) {
34         int sum = preSum[i] - preSum[index];
35         if (judge(sum)) {
36             DFS(preSum, i, n, count + 1, res);
37         }
38     }
39 }
40
41 int main() {
42     string input;
43     getline(cin, input);
44     int n = input.size();
```

```
45     if (n == 0) {
46         cout << 0;
47     }
48     // 预计算前缀和 快速求区间和
49     vector<int> preSum(n+1, 0);
50     for (int i = 1; i <= n; i++) {
51         preSum[i] = preSum[i-1] + input[i-1];
52     }
53     // 存储分割方案的子串数目
54     vector<int> res;
55     DFS(preSum, 0, n, 0, res);
56     if (res.empty()) {
57         cout << 0;
58     } else if (res.size() > 1) {
59         cout << -1;
60     } else {
61         cout << res[0];
62     }
63     return 0;
64 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为水仙花数
5     public static boolean judge(int num) {
6         if (num < 100 || num > 999) return false;
7         int sum = 0, tmp = num;
8         while (tmp != 0) {
9             int mod = tmp % 10;
10            sum += mod * mod * mod;
11            tmp /= 10;
12        }
13        return sum == num;
14    }
15
16    // 递归回溯
17    public static void DFS(int[] preSum, int index, int n, int count, List
<Integer> res) {
18        // 全部分割完成
19        if (index == n) {
20            res.add(count);
21            return;
22        }
23        for (int i = index + 1; i <= n; i++) {
24            int sum = preSum[i] - preSum[index];
25            if (judge(sum)) {
26                DFS(preSum, i, n, count + 1, res);
27            }
28        }
29    }
30
31    public static void main(String[] args) {
32        Scanner scanner = new Scanner(System.in);
33        if (!scanner.hasNextLine()) {
34            System.out.println(0);
35            return;
36        }
37        String input = scanner.nextLine();
38        int n = input.length();
39        if (n == 0) {
40            System.out.println(0);
41            return;
42        }
43        // 预计算前缀和 快速求区间和
```

```
45     int[] preSum = new int[n + 1];
46     for (int i = 1; i <= n; i++) {
47         preSum[i] = preSum[i - 1] + input.charAt(i - 1);
48     }
49
50     List<Integer> res = new ArrayList<>();
51     DFS(preSum, 0, n, 0, res);
52
53     if (res.isEmpty()) {
54         System.out.println(0);
55     } else if (res.size() > 1) {
56         System.out.println(-1);
57     } else {
58         System.out.println(res.get(0));
59     }
60 }
61 }
```

Python

```
1 def judge(num):
2     # 判断是否为水仙花数 (Narcissistic number)
3     if num < 100 or num > 999:
4         return False
5     sum_ = 0
6     tmp = num
7     while tmp != 0:
8         mod = tmp % 10
9         sum_ += mod ** 3
10        tmp //= 10
11    return sum_ == num
12
13 # 递归回溯
14 def DFS(preSum, index, n, count, res):
15     # 全部分割完成
16     if index == n:
17         res.append(count)
18         return
19     for i in range(index + 1, n + 1):
20         sum_ = preSum[i] - preSum[index]
21         if judge(sum_):
22             DFS(preSum, i, n, count + 1, res)
23
24 def main():
25     try:
26         input_str = input().strip()
27     except:
28         print(0)
29         return
30
31     n = len(input_str)
32     if n == 0:
33         print(0)
34         return
35
36     # 预计算前缀和 快速求区间和
37     preSum = [0] * (n + 1)
38     for i in range(1, n + 1):
39         preSum[i] = preSum[i - 1] + ord(input_str[i - 1])
40
41     # 存储分割方案的子串数目
42     res = []
43     DFS(preSum, 0, n, 0, res)
44
45     if not res:
```

```
46         print(0)
47     elif len(res) > 1:
48         print(-1)
49     else:
50         print(res[0])
51
52 if __name__ == "__main__":
53     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 function judge(num) {
9     // 判断是否为水仙花数
10    if (num < 100 || num > 999) return false;
11    let sum = 0, tmp = num;
12    while (tmp !== 0) {
13        let mod = tmp % 10;
14        sum += mod ** 3;
15        tmp = Math.floor(tmp / 10);
16    }
17    return sum === num;
18 }
19
20 // 递归回溯
21 function DFS(preSum, index, n, count, res) {
22     // 全部分割完成
23     if (index === n) {
24         res.push(count);
25         return;
26     }
27     for (let i = index + 1; i <= n; i++) {
28         const sum = preSum[i] - preSum[index];
29         if (judge(sum)) {
30             DFS(preSum, i, n, count + 1, res);
31         }
32     }
33 }
34
35 rl.on('line', function (input) {
36     const n = input.length;
37     if (n === 0) {
38         console.log(0);
39         process.exit();
40     }
41
42     // 预计算前缀和 快速求区间和
43     const preSum = new Array(n + 1).fill(0);
44     for (let i = 1; i <= n; i++) {
45         preSum[i] = preSum[i - 1] + input.charCodeAt(i - 1);
```

```
46     }
47
48     const res = [];
49     DFS(preSum, 0, n, 0, res);
50
51     if (res.length === 0) {
52         console.log(0);
53     } else if (res.length > 1) {
54         console.log(-1);
55     } else {
56         console.log(res[0]);
57     }
58
59     process.exit();
60 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 // 判断是否为水仙花数
10 func judge(num int) bool {
11     if num < 100 || num > 999 {
12         return false
13     }
14     sum, tmp := 0, num
15     for tmp != 0 {
16         mod := tmp % 10
17         sum += mod * mod * mod
18         tmp /= 10
19     }
20     return sum == num
21 }
22
23 // 递归回溯
24 func DFS(preSum []int, index, n, count int, res *[]int) {
25     if index == n {
26         *res = append(*res, count)
27         return
28     }
29     for i := index + 1; i <= n; i++ {
30         sum := preSum[i] - preSum[index]
31         if judge(sum) {
32             DFS(preSum, i, n, count+1, res)
33         }
34     }
35 }
36
37 func main() {
38     scanner := bufio.NewScanner(os.Stdin)
39     if !scanner.Scan() {
40         fmt.Println(0)
41         return
42     }
43     input := scanner.Text()
44     n := len(input)
45     if n == 0 {
```

```
46     fmt.Println(0)
47     return
48 }
49
50 // 预计算前缀和 快速求区间和
51 preSum := make([]int, n+1)
52 for i := 1; i <= n; i++ {
53     preSum[i] = preSum[i-1] + int(input[i-1])
54 }
55
56 var res []int
57 DFS(preSum, 0, n, 0, &res)
58
59 if len(res) == 0 {
60     fmt.Println(0)
61 } else if len(res) > 1 {
62     fmt.Println(-1)
63 } else {
64     fmt.Println(res[0])
65 }
66 }
```

切割字符串

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

给定非空字符串s，将该字符串分割成一些子串，使每个子串的ASCII码值的和均为水仙花数。

- 1、若分割不成功，则返回0；
- 2、若分割成功且分割结果不唯一，则返回-1；
- 3、若分割成功且分割结果唯一，则返回分割后子串的数目。

备注

"水仙花数"是指一个三位数，每位上数字的立方和等于该数字本身，如 371 是'水仙花数'，因为 $371=3^3+7^3+1^3$

输入描述

输入字符串的最大长度为200

输出描述

根据题目描述中情况，返回相应的结果。

用例1

输入

```
▼ Plain Text |  
1 abc
```

输出

```
▼ Plain Text |  
1 0
```

说明

| 分割不成功

用例2

输入

```
▼ Plain Text |  
1 f3@d5a8
```

输出

```
▼ Plain Text |  
1 -1
```

说明

| 分割成功但分割结果不唯一，可以分割为两组，一组'f3'和'@d5a8'，另外一组'f3@d5'和'a8'

用例2

输入

```
▼ Plain Text |  
1 AXdddF
```

输出

```
1 2
```

说明

分割成功且分割结果唯一，可以分割'AX'(153)和'ddF'(370)成两个子串

题解

思路： 前缀和 + 递归回溯

1. 预处理计算前缀和方便后续快速计算区间和的值。
2. 使用递归回溯尝试进行切割，使用 `res` 数组存储每种切割方案划分的子串数量。递归的逻辑如下：
 - a. 从当前位置作为起点，向右枚举子串终点，如果[起点，终点]的和能组成水仙花数。子串数量+1，继续向下递归。
 - b. 递归的结束条件为：将字符串全部切割完毕，此时可以获得这种切割方案的子串数量，存入`res`。
3. 输出结果，根据 `res` 数组的长度来进行对应输出即可。

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 // 判断是否为水仙花数
12 bool judge(int num) {
13     if (num < 100 || num > 999) {
14         return false;
15     }
16     int sum = 0;
17     int tmp = num;
18     while (tmp != 0) {
19         int mod = tmp % 10;
20         sum += mod * mod * mod;
21         tmp /= 10;
22     }
23     return sum == num;
24 }
25
26 // 递归回溯
27 void DFS(vector<int>& preSum, int index, int n, int count, vector<int>& res) {
28     // 全部分割完成
29     if (index == n) {
30         res.push_back(count);
31         return;
32     }
33     for (int i = index + 1; i <= n; i++) {
34         int sum = preSum[i] - preSum[index];
35         if (judge(sum)) {
36             DFS(preSum, i, n, count + 1, res);
37         }
38     }
39 }
40
41 int main() {
42     string input;
43     getline(cin, input);
44     int n = input.size();
```

```
45     if (n == 0) {
46         cout << 0;
47     }
48     // 预计算前缀和 快速求区间和
49     vector<int> preSum(n+1, 0);
50     for (int i = 1; i <= n; i++) {
51         preSum[i] = preSum[i-1] + input[i-1];
52     }
53     // 存储分割方案的子串数目
54     vector<int> res;
55     DFS(preSum, 0, n, 0, res);
56     if (res.empty()) {
57         cout << 0;
58     } else if (res.size() > 1) {
59         cout << -1;
60     } else {
61         cout << res[0];
62     }
63     return 0;
64 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为水仙花数
5     public static boolean judge(int num) {
6         if (num < 100 || num > 999) return false;
7         int sum = 0, tmp = num;
8         while (tmp != 0) {
9             int mod = tmp % 10;
10            sum += mod * mod * mod;
11            tmp /= 10;
12        }
13        return sum == num;
14    }
15
16    // 递归回溯
17    public static void DFS(int[] preSum, int index, int n, int count, List
<Integer> res) {
18        // 全部分割完成
19        if (index == n) {
20            res.add(count);
21            return;
22        }
23        for (int i = index + 1; i <= n; i++) {
24            int sum = preSum[i] - preSum[index];
25            if (judge(sum)) {
26                DFS(preSum, i, n, count + 1, res);
27            }
28        }
29    }
30
31    public static void main(String[] args) {
32        Scanner scanner = new Scanner(System.in);
33        if (!scanner.hasNextLine()) {
34            System.out.println(0);
35            return;
36        }
37        String input = scanner.nextLine();
38        int n = input.length();
39        if (n == 0) {
40            System.out.println(0);
41            return;
42        }
43        // 预计算前缀和 快速求区间和
```

```
45     int[] preSum = new int[n + 1];
46     for (int i = 1; i <= n; i++) {
47         preSum[i] = preSum[i - 1] + input.charAt(i - 1);
48     }
49
50     List<Integer> res = new ArrayList<>();
51     DFS(preSum, 0, n, 0, res);
52
53     if (res.isEmpty()) {
54         System.out.println(0);
55     } else if (res.size() > 1) {
56         System.out.println(-1);
57     } else {
58         System.out.println(res.get(0));
59     }
60 }
61 }
```

Python

```
1 def judge(num):
2     # 判断是否为水仙花数 (Narcissistic number)
3     if num < 100 or num > 999:
4         return False
5     sum_ = 0
6     tmp = num
7     while tmp != 0:
8         mod = tmp % 10
9         sum_ += mod ** 3
10        tmp //= 10
11    return sum_ == num
12
13 # 递归回溯
14 def DFS(preSum, index, n, count, res):
15     # 全部分割完成
16     if index == n:
17         res.append(count)
18         return
19     for i in range(index + 1, n + 1):
20         sum_ = preSum[i] - preSum[index]
21         if judge(sum_):
22             DFS(preSum, i, n, count + 1, res)
23
24 def main():
25     try:
26         input_str = input().strip()
27     except:
28         print(0)
29         return
30
31     n = len(input_str)
32     if n == 0:
33         print(0)
34         return
35
36     # 预计算前缀和 快速求区间和
37     preSum = [0] * (n + 1)
38     for i in range(1, n + 1):
39         preSum[i] = preSum[i - 1] + ord(input_str[i - 1])
40
41     # 存储分割方案的子串数目
42     res = []
43     DFS(preSum, 0, n, 0, res)
44
45     if not res:
```

```
46         print(0)
47     elif len(res) > 1:
48         print(-1)
49     else:
50         print(res[0])
51
52 if __name__ == "__main__":
53     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 function judge(num) {
9     // 判断是否为水仙花数
10    if (num < 100 || num > 999) return false;
11    let sum = 0, tmp = num;
12    while (tmp !== 0) {
13        let mod = tmp % 10;
14        sum += mod ** 3;
15        tmp = Math.floor(tmp / 10);
16    }
17    return sum === num;
18 }
19
20 // 递归回溯
21 function DFS(preSum, index, n, count, res) {
22     // 全部分割完成
23     if (index === n) {
24         res.push(count);
25         return;
26     }
27     for (let i = index + 1; i <= n; i++) {
28         const sum = preSum[i] - preSum[index];
29         if (judge(sum)) {
30             DFS(preSum, i, n, count + 1, res);
31         }
32     }
33 }
34
35 rl.on('line', function (input) {
36     const n = input.length;
37     if (n === 0) {
38         console.log(0);
39         process.exit();
40     }
41
42     // 预计算前缀和 快速求区间和
43     const preSum = new Array(n + 1).fill(0);
44     for (let i = 1; i <= n; i++) {
45         preSum[i] = preSum[i - 1] + input.charCodeAt(i - 1);
```

```
46     }
47
48     const res = [];
49     DFS(preSum, 0, n, 0, res);
50
51     if (res.length === 0) {
52         console.log(0);
53     } else if (res.length > 1) {
54         console.log(-1);
55     } else {
56         console.log(res[0]);
57     }
58
59     process.exit();
60 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 // 判断是否为水仙花数
10 func judge(num int) bool {
11     if num < 100 || num > 999 {
12         return false
13     }
14     sum, tmp := 0, num
15     for tmp != 0 {
16         mod := tmp % 10
17         sum += mod * mod * mod
18         tmp /= 10
19     }
20     return sum == num
21 }
22
23 // 递归回溯
24 func DFS(preSum []int, index, n, count int, res *[]int) {
25     if index == n {
26         *res = append(*res, count)
27         return
28     }
29     for i := index + 1; i <= n; i++ {
30         sum := preSum[i] - preSum[index]
31         if judge(sum) {
32             DFS(preSum, i, n, count+1, res)
33         }
34     }
35 }
36
37 func main() {
38     scanner := bufio.NewScanner(os.Stdin)
39     if !scanner.Scan() {
40         fmt.Println(0)
41         return
42     }
43     input := scanner.Text()
44     n := len(input)
45     if n == 0 {
```

```

46     fmt.Println(0)
47     return
48 }
49
50 // 预计算前缀和 快速求区间和
51 preSum := make([]int, n+1)
52 for i := 1; i <= n; i++ {
53     preSum[i] = preSum[i-1] + int(input[i-1])
54 }
55
56 var res []int
57 DFS(preSum, 0, n, 0, &res)
58
59 if len(res) == 0 {
60     fmt.Println(0)
61 } else if len(res) > 1 {
62     fmt.Println(-1)
63 } else {
64     fmt.Println(res[0])
65 }
66 }

```

| 来自: 华为OD机考 2025C卷 – 切割字符串 (C++ & Python & JAVA & JS & GO)-CSDN博客

切割字符串

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

给定非空字符串s，将该字符串分割成一些子串，使每个子串的ASCII码值的和均为水仙花数。

- 1、若分割不成功，则返回0；
- 2、若分割成功且分割结果不唯一，则返回-1；
- 3、若分割成功且分割结果唯一，则返回分割后子串的数目。

备注

"水仙花数"是指一个三位数，每位上数字的立方和等于该数字本身，如 371 是'水仙花数'，因为 $3^3 + 7^3 + 1^3 = 371$

输入描述

输入字符串的最大长度为200

输出描述

根据题目描述中情况，返回相应结果。

用例1

输入

Plain Text |

```
1 abc
```

输出

Plain Text |

```
1 0
```

说明

| 分割不成功

用例2

输入

Plain Text |

```
1 f3@d5a8
```

输出

Plain Text |

```
1 -1
```

说明

| 分割成功但分割结果不唯一，可以分割为两组，一组'f3'和'@d5a8'，另外一组'f3@d5'和'a8'

用例2

输入

Plain Text |

1 AXdddF

输出

Plain Text |

1 2

说明

| 分割成功且分割结果唯一，可以分割'AX'(153)和'dddF'(370)成两个子串

题解

思路： 前缀和 + 递归回溯

1. 预处理计算前缀和方便后续快速计算区间和的值。
2. 使用递归回溯尝试进行切割，使用 `res` 数组存储每种切割方案划分的子串数量。递归的逻辑如下：
 - a. 从当前位置作为起点，向右枚举子串终点，如果[起点，终点]的和能组成水仙花数。子串数量+1，继续向下递归。
 - b. 递归的结束条件为：将字符串全部切割完毕，此时可以获得这种切割方案的子串数量，存入`res`。
3. 输出结果，根据 `res` 数组的长度来进行对应输出即可。

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 // 判断是否为水仙花数
12 bool judge(int num) {
13     if (num < 100 || num > 999) {
14         return false;
15     }
16     int sum = 0;
17     int tmp = num;
18     while (tmp != 0) {
19         int mod = tmp % 10;
20         sum += mod * mod * mod;
21         tmp /= 10;
22     }
23     return sum == num;
24 }
25
26 // 递归回溯
27 void DFS(vector<int>& preSum, int index, int n, int count, vector<int>& res) {
28     // 全部分割完成
29     if (index == n) {
30         res.push_back(count);
31         return;
32     }
33     for (int i = index + 1; i <= n; i++) {
34         int sum = preSum[i] - preSum[index];
35         if (judge(sum)) {
36             DFS(preSum, i, n, count + 1, res);
37         }
38     }
39 }
40
41 int main() {
42     string input;
43     getline(cin, input);
44     int n = input.size();
```

```
45     if (n == 0) {
46         cout << 0;
47     }
48     // 预计算前缀和 快速求区间和
49     vector<int> preSum(n+1, 0);
50     for (int i = 1; i <= n; i++) {
51         preSum[i] = preSum[i-1] + input[i-1];
52     }
53     // 存储分割方案的子串数目
54     vector<int> res;
55     DFS(preSum, 0, n, 0, res);
56     if (res.empty()) {
57         cout << 0;
58     } else if (res.size() > 1) {
59         cout << -1;
60     } else {
61         cout << res[0];
62     }
63     return 0;
64 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为水仙花数
5     public static boolean judge(int num) {
6         if (num < 100 || num > 999) return false;
7         int sum = 0, tmp = num;
8         while (tmp != 0) {
9             int mod = tmp % 10;
10            sum += mod * mod * mod;
11            tmp /= 10;
12        }
13        return sum == num;
14    }
15
16    // 递归回溯
17    public static void DFS(int[] preSum, int index, int n, int count, List
<Integer> res) {
18        // 全部分割完成
19        if (index == n) {
20            res.add(count);
21            return;
22        }
23        for (int i = index + 1; i <= n; i++) {
24            int sum = preSum[i] - preSum[index];
25            if (judge(sum)) {
26                DFS(preSum, i, n, count + 1, res);
27            }
28        }
29    }
30
31    public static void main(String[] args) {
32        Scanner scanner = new Scanner(System.in);
33        if (!scanner.hasNextLine()) {
34            System.out.println(0);
35            return;
36        }
37        String input = scanner.nextLine();
38        int n = input.length();
39        if (n == 0) {
40            System.out.println(0);
41            return;
42        }
43        // 预计算前缀和 快速求区间和
```

```
45     int[] preSum = new int[n + 1];
46     for (int i = 1; i <= n; i++) {
47         preSum[i] = preSum[i - 1] + input.charAt(i - 1);
48     }
49
50     List<Integer> res = new ArrayList<>();
51     DFS(preSum, 0, n, 0, res);
52
53     if (res.isEmpty()) {
54         System.out.println(0);
55     } else if (res.size() > 1) {
56         System.out.println(-1);
57     } else {
58         System.out.println(res.get(0));
59     }
60 }
61 }
```

Python

```
1 def judge(num):
2     # 判断是否为水仙花数 (Narcissistic number)
3     if num < 100 or num > 999:
4         return False
5     sum_ = 0
6     tmp = num
7     while tmp != 0:
8         mod = tmp % 10
9         sum_ += mod ** 3
10        tmp //= 10
11    return sum_ == num
12
13 # 递归回溯
14 def DFS(preSum, index, n, count, res):
15     # 全部分割完成
16     if index == n:
17         res.append(count)
18         return
19     for i in range(index + 1, n + 1):
20         sum_ = preSum[i] - preSum[index]
21         if judge(sum_):
22             DFS(preSum, i, n, count + 1, res)
23
24 def main():
25     try:
26         input_str = input().strip()
27     except:
28         print(0)
29         return
30
31     n = len(input_str)
32     if n == 0:
33         print(0)
34         return
35
36     # 预计算前缀和 快速求区间和
37     preSum = [0] * (n + 1)
38     for i in range(1, n + 1):
39         preSum[i] = preSum[i - 1] + ord(input_str[i - 1])
40
41     # 存储分割方案的子串数目
42     res = []
43     DFS(preSum, 0, n, 0, res)
44
45     if not res:
```

```
46         print(0)
47     elif len(res) > 1:
48         print(-1)
49     else:
50         print(res[0])
51
52 if __name__ == "__main__":
53     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 function judge(num) {
9     // 判断是否为水仙花数
10    if (num < 100 || num > 999) return false;
11    let sum = 0, tmp = num;
12    while (tmp !== 0) {
13        let mod = tmp % 10;
14        sum += mod ** 3;
15        tmp = Math.floor(tmp / 10);
16    }
17    return sum === num;
18 }
19
20 // 递归回溯
21 function DFS(preSum, index, n, count, res) {
22     // 全部分割完成
23     if (index === n) {
24         res.push(count);
25         return;
26     }
27     for (let i = index + 1; i <= n; i++) {
28         const sum = preSum[i] - preSum[index];
29         if (judge(sum)) {
30             DFS(preSum, i, n, count + 1, res);
31         }
32     }
33 }
34
35 rl.on('line', function (input) {
36     const n = input.length;
37     if (n === 0) {
38         console.log(0);
39         process.exit();
40     }
41
42     // 预计算前缀和 快速求区间和
43     const preSum = new Array(n + 1).fill(0);
44     for (let i = 1; i <= n; i++) {
45         preSum[i] = preSum[i - 1] + input.charCodeAt(i - 1);
```

```
46     }
47
48     const res = [];
49     DFS(preSum, 0, n, 0, res);
50
51     if (res.length === 0) {
52         console.log(0);
53     } else if (res.length > 1) {
54         console.log(-1);
55     } else {
56         console.log(res[0]);
57     }
58
59     process.exit();
60 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 // 判断是否为水仙花数
10 func judge(num int) bool {
11     if num < 100 || num > 999 {
12         return false
13     }
14     sum, tmp := 0, num
15     for tmp != 0 {
16         mod := tmp % 10
17         sum += mod * mod * mod
18         tmp /= 10
19     }
20     return sum == num
21 }
22
23 // 递归回溯
24 func DFS(preSum []int, index, n, count int, res *[]int) {
25     if index == n {
26         *res = append(*res, count)
27         return
28     }
29     for i := index + 1; i <= n; i++ {
30         sum := preSum[i] - preSum[index]
31         if judge(sum) {
32             DFS(preSum, i, n, count+1, res)
33         }
34     }
35 }
36
37 func main() {
38     scanner := bufio.NewScanner(os.Stdin)
39     if !scanner.Scan() {
40         fmt.Println(0)
41         return
42     }
43     input := scanner.Text()
44     n := len(input)
45     if n == 0 {
```

```

46     fmt.Println(0)
47     return
48 }
49
50 // 预计算前缀和 快速求区间和
51 preSum := make([]int, n+1)
52 for i := 1; i <= n; i++ {
53     preSum[i] = preSum[i-1] + int(input[i-1])
54 }
55
56 var res []int
57 DFS(preSum, 0, n, 0, &res)
58
59 if len(res) == 0 {
60     fmt.Println(0)
61 } else if len(res) > 1 {
62     fmt.Println(-1)
63 } else {
64     fmt.Println(res[0])
65 }
66 }

```

| 来自: 华为OD机考 2025C卷 – 切割字符串 (C++ & Python & JAVA & JS & GO)–CSDN博客

切割字符串

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

给定非空字符串s，将该字符串分割成一些子串，使每个子串的ASCII码值的和均为水仙花数。

- 1、若分割不成功，则返回0；
- 2、若分割成功且分割结果不唯一，则返回-1；
- 3、若分割成功且分割结果唯一，则返回分割后子串的数目。

备注

"水仙花数"是指一个三位数，每位上数字的立方和等于该数字本身，如 371 是'水仙花数'，因为 $3^3 + 7^3 + 1^3 = 371$

输入描述

输入字符串的最大长度为200

输出描述

根据题目描述中情况，返回相应结果。

用例1

输入

Plain Text |

```
1 abc
```

输出

Plain Text |

```
1 0
```

说明

| 分割不成功

用例2

输入

Plain Text |

```
1 f3@d5a8
```

输出

Plain Text |

```
1 -1
```

说明

| 分割成功但分割结果不唯一，可以分割为两组，一组'f3'和'@d5a8'，另外一组'f3@d5'和'a8'

用例2

输入

Plain Text

1 AXdddF

输出

Plain Text

1 2

说明

| 分割成功且分割结果唯一，可以分割'AX'(153)和'dddF'(370)成两个子串

题解

思路： 前缀和 + 递归回溯

1. 预处理计算前缀和方便后续快速计算区间和的值。
2. 使用递归回溯尝试进行切割，使用 `res` 数组存储每种切割方案划分的子串数量。递归的逻辑如下：
 - a. 从当前位置作为起点，向右枚举子串终点，如果[起点，终点]的和能组成水仙花数。子串数量+1，继续向下递归。
 - b. 递归的结束条件为：将字符串全部切割完毕，此时可以获得这种切割方案的子串数量，存入`res`。
3. 输出结果，根据 `res` 数组的长度来进行对应输出即可。

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 // 判断是否为水仙花数
12 bool judge(int num) {
13     if (num < 100 || num > 999) {
14         return false;
15     }
16     int sum = 0;
17     int tmp = num;
18     while (tmp != 0) {
19         int mod = tmp % 10;
20         sum += mod * mod * mod;
21         tmp /= 10;
22     }
23     return sum == num;
24 }
25
26 // 递归回溯
27 void DFS(vector<int>& preSum, int index, int n, int count, vector<int>& res) {
28     // 全部分割完成
29     if (index == n) {
30         res.push_back(count);
31         return;
32     }
33     for (int i = index + 1; i <= n; i++) {
34         int sum = preSum[i] - preSum[index];
35         if (judge(sum)) {
36             DFS(preSum, i, n, count + 1, res);
37         }
38     }
39 }
40
41 int main() {
42     string input;
43     getline(cin, input);
44     int n = input.size();
```

```
45     if (n == 0) {
46         cout << 0;
47     }
48     // 预计算前缀和 快速求区间和
49     vector<int> preSum(n+1, 0);
50     for (int i = 1; i <= n; i++) {
51         preSum[i] = preSum[i-1] + input[i-1];
52     }
53     // 存储分割方案的子串数目
54     vector<int> res;
55     DFS(preSum, 0, n, 0, res);
56     if (res.empty()) {
57         cout << 0;
58     } else if (res.size() > 1) {
59         cout << -1;
60     } else {
61         cout << res[0];
62     }
63     return 0;
64 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为水仙花数
5     public static boolean judge(int num) {
6         if (num < 100 || num > 999) return false;
7         int sum = 0, tmp = num;
8         while (tmp != 0) {
9             int mod = tmp % 10;
10            sum += mod * mod * mod;
11            tmp /= 10;
12        }
13        return sum == num;
14    }
15
16    // 递归回溯
17    public static void DFS(int[] preSum, int index, int n, int count, List
<Integer> res) {
18        // 全部分割完成
19        if (index == n) {
20            res.add(count);
21            return;
22        }
23        for (int i = index + 1; i <= n; i++) {
24            int sum = preSum[i] - preSum[index];
25            if (judge(sum)) {
26                DFS(preSum, i, n, count + 1, res);
27            }
28        }
29    }
30
31    public static void main(String[] args) {
32        Scanner scanner = new Scanner(System.in);
33        if (!scanner.hasNextLine()) {
34            System.out.println(0);
35            return;
36        }
37        String input = scanner.nextLine();
38        int n = input.length();
39        if (n == 0) {
40            System.out.println(0);
41            return;
42        }
43        // 预计算前缀和 快速求区间和
```

```
45     int[] preSum = new int[n + 1];
46     for (int i = 1; i <= n; i++) {
47         preSum[i] = preSum[i - 1] + input.charAt(i - 1);
48     }
49
50     List<Integer> res = new ArrayList<>();
51     DFS(preSum, 0, n, 0, res);
52
53     if (res.isEmpty()) {
54         System.out.println(0);
55     } else if (res.size() > 1) {
56         System.out.println(-1);
57     } else {
58         System.out.println(res.get(0));
59     }
60 }
61 }
```

Python

```
1 def judge(num):
2     # 判断是否为水仙花数 (Narcissistic number)
3     if num < 100 or num > 999:
4         return False
5     sum_ = 0
6     tmp = num
7     while tmp != 0:
8         mod = tmp % 10
9         sum_ += mod ** 3
10        tmp //= 10
11    return sum_ == num
12
13 # 递归回溯
14 def DFS(preSum, index, n, count, res):
15     # 全部分割完成
16     if index == n:
17         res.append(count)
18         return
19     for i in range(index + 1, n + 1):
20         sum_ = preSum[i] - preSum[index]
21         if judge(sum_):
22             DFS(preSum, i, n, count + 1, res)
23
24 def main():
25     try:
26         input_str = input().strip()
27     except:
28         print(0)
29         return
30
31     n = len(input_str)
32     if n == 0:
33         print(0)
34         return
35
36     # 预计算前缀和 快速求区间和
37     preSum = [0] * (n + 1)
38     for i in range(1, n + 1):
39         preSum[i] = preSum[i - 1] + ord(input_str[i - 1])
40
41     # 存储分割方案的子串数目
42     res = []
43     DFS(preSum, 0, n, 0, res)
44
45     if not res:
```

```
46         print(0)
47     elif len(res) > 1:
48         print(-1)
49     else:
50         print(res[0])
51
52 if __name__ == "__main__":
53     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 function judge(num) {
9     // 判断是否为水仙花数
10    if (num < 100 || num > 999) return false;
11    let sum = 0, tmp = num;
12    while (tmp !== 0) {
13        let mod = tmp % 10;
14        sum += mod ** 3;
15        tmp = Math.floor(tmp / 10);
16    }
17    return sum === num;
18 }
19
20 // 递归回溯
21 function DFS(preSum, index, n, count, res) {
22     // 全部分割完成
23     if (index === n) {
24         res.push(count);
25         return;
26     }
27     for (let i = index + 1; i <= n; i++) {
28         const sum = preSum[i] - preSum[index];
29         if (judge(sum)) {
30             DFS(preSum, i, n, count + 1, res);
31         }
32     }
33 }
34
35 rl.on('line', function (input) {
36     const n = input.length;
37     if (n === 0) {
38         console.log(0);
39         process.exit();
40     }
41
42     // 预计算前缀和 快速求区间和
43     const preSum = new Array(n + 1).fill(0);
44     for (let i = 1; i <= n; i++) {
45         preSum[i] = preSum[i - 1] + input.charCodeAt(i - 1);
```

```
46     }
47
48     const res = [];
49     DFS(preSum, 0, n, 0, res);
50
51     if (res.length === 0) {
52         console.log(0);
53     } else if (res.length > 1) {
54         console.log(-1);
55     } else {
56         console.log(res[0]);
57     }
58
59     process.exit();
60 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 // 判断是否为水仙花数
10 func judge(num int) bool {
11     if num < 100 || num > 999 {
12         return false
13     }
14     sum, tmp := 0, num
15     for tmp != 0 {
16         mod := tmp % 10
17         sum += mod * mod * mod
18         tmp /= 10
19     }
20     return sum == num
21 }
22
23 // 递归回溯
24 func DFS(preSum []int, index, n, count int, res *[]int) {
25     if index == n {
26         *res = append(*res, count)
27         return
28     }
29     for i := index + 1; i <= n; i++ {
30         sum := preSum[i] - preSum[index]
31         if judge(sum) {
32             DFS(preSum, i, n, count+1, res)
33         }
34     }
35 }
36
37 func main() {
38     scanner := bufio.NewScanner(os.Stdin)
39     if !scanner.Scan() {
40         fmt.Println(0)
41         return
42     }
43     input := scanner.Text()
44     n := len(input)
45     if n == 0 {
```

```

46     fmt.Println(0)
47     return
48 }
49
50 // 预计算前缀和 快速求区间和
51 preSum := make([]int, n+1)
52 for i := 1; i <= n; i++ {
53     preSum[i] = preSum[i-1] + int(input[i-1])
54 }
55
56 var res []int
57 DFS(preSum, 0, n, 0, &res)
58
59 if len(res) == 0 {
60     fmt.Println(0)
61 } else if len(res) > 1 {
62     fmt.Println(-1)
63 } else {
64     fmt.Println(res[0])
65 }
66 }

```

| 来自: 华为OD机考 2025C卷 – 切割字符串 (C++ & Python & JAVA & JS & GO)–CSDN博客

切割字符串

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

给定非空字符串s，将该字符串分割成一些子串，使每个子串的ASCII码值的和均为水仙花数。

- 1、若分割不成功，则返回0；
- 2、若分割成功且分割结果不唯一，则返回-1；
- 3、若分割成功且分割结果唯一，则返回分割后子串的数目。

备注

"水仙花数"是指一个三位数，每位上数字的立方和等于该数字本身，如 371 是'水仙花数'，因为 $3^3 + 7^3 + 1^3 = 371$

输入描述

输入字符串的最大长度为200

输出描述

根据题目描述中情况，返回相应结果。

用例1

输入

Plain Text |

```
1 abc
```

输出

Plain Text |

```
1 0
```

说明

| 分割不成功

用例2

输入

Plain Text |

```
1 f3@d5a8
```

输出

Plain Text |

```
1 -1
```

说明

| 分割成功但分割结果不唯一，可以分割为两组，一组'f3'和'@d5a8'，另外一组'f3@d5'和'a8'

用例2

输入

Plain Text

1 AXdddF

输出

Plain Text

1 2

说明

| 分割成功且分割结果唯一，可以分割'AX'(153)和'dddF'(370)成两个子串

题解

思路： 前缀和 + 递归回溯

1. 预处理计算前缀和方便后续快速计算区间和的值。
2. 使用递归回溯尝试进行切割，使用 `res` 数组存储每种切割方案划分的子串数量。递归的逻辑如下：
 - a. 从当前位置作为起点，向右枚举子串终点，如果[起点，终点]的和能组成水仙花数。子串数量+1，继续向下递归。
 - b. 递归的结束条件为：将字符串全部切割完毕，此时可以获得这种切割方案的子串数量，存入`res`。
3. 输出结果，根据 `res` 数组的长度来进行对应输出即可。

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 // 判断是否为水仙花数
12 bool judge(int num) {
13     if (num < 100 || num > 999) {
14         return false;
15     }
16     int sum = 0;
17     int tmp = num;
18     while (tmp != 0) {
19         int mod = tmp % 10;
20         sum += mod * mod * mod;
21         tmp /= 10;
22     }
23     return sum == num;
24 }
25
26 // 递归回溯
27 void DFS(vector<int>& preSum, int index, int n, int count, vector<int>& res) {
28     // 全部分割完成
29     if (index == n) {
30         res.push_back(count);
31         return;
32     }
33     for (int i = index + 1; i <= n; i++) {
34         int sum = preSum[i] - preSum[index];
35         if (judge(sum)) {
36             DFS(preSum, i, n, count + 1, res);
37         }
38     }
39 }
40
41 int main() {
42     string input;
43     getline(cin, input);
44     int n = input.size();
```

```
45     if (n == 0) {
46         cout << 0;
47     }
48     // 预计算前缀和 快速求区间和
49     vector<int> preSum(n+1, 0);
50     for (int i = 1; i <= n; i++) {
51         preSum[i] = preSum[i-1] + input[i-1];
52     }
53     // 存储分割方案的子串数目
54     vector<int> res;
55     DFS(preSum, 0, n, 0, res);
56     if (res.empty()) {
57         cout << 0;
58     } else if (res.size() > 1) {
59         cout << -1;
60     } else {
61         cout << res[0];
62     }
63     return 0;
64 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为水仙花数
5     public static boolean judge(int num) {
6         if (num < 100 || num > 999) return false;
7         int sum = 0, tmp = num;
8         while (tmp != 0) {
9             int mod = tmp % 10;
10            sum += mod * mod * mod;
11            tmp /= 10;
12        }
13        return sum == num;
14    }
15
16    // 递归回溯
17    public static void DFS(int[] preSum, int index, int n, int count, List
<Integer> res) {
18        // 全部分割完成
19        if (index == n) {
20            res.add(count);
21            return;
22        }
23        for (int i = index + 1; i <= n; i++) {
24            int sum = preSum[i] - preSum[index];
25            if (judge(sum)) {
26                DFS(preSum, i, n, count + 1, res);
27            }
28        }
29    }
30
31    public static void main(String[] args) {
32        Scanner scanner = new Scanner(System.in);
33        if (!scanner.hasNextLine()) {
34            System.out.println(0);
35            return;
36        }
37        String input = scanner.nextLine();
38        int n = input.length();
39        if (n == 0) {
40            System.out.println(0);
41            return;
42        }
43        // 预计算前缀和 快速求区间和
```

```
45     int[] preSum = new int[n + 1];
46     for (int i = 1; i <= n; i++) {
47         preSum[i] = preSum[i - 1] + input.charAt(i - 1);
48     }
49
50     List<Integer> res = new ArrayList<>();
51     DFS(preSum, 0, n, 0, res);
52
53     if (res.isEmpty()) {
54         System.out.println(0);
55     } else if (res.size() > 1) {
56         System.out.println(-1);
57     } else {
58         System.out.println(res.get(0));
59     }
60 }
61 }
```

Python

```
1 def judge(num):
2     # 判断是否为水仙花数 (Narcissistic number)
3     if num < 100 or num > 999:
4         return False
5     sum_ = 0
6     tmp = num
7     while tmp != 0:
8         mod = tmp % 10
9         sum_ += mod ** 3
10        tmp //= 10
11    return sum_ == num
12
13 # 递归回溯
14 def DFS(preSum, index, n, count, res):
15     # 全部分割完成
16     if index == n:
17         res.append(count)
18         return
19     for i in range(index + 1, n + 1):
20         sum_ = preSum[i] - preSum[index]
21         if judge(sum_):
22             DFS(preSum, i, n, count + 1, res)
23
24 def main():
25     try:
26         input_str = input().strip()
27     except:
28         print(0)
29         return
30
31     n = len(input_str)
32     if n == 0:
33         print(0)
34         return
35
36     # 预计算前缀和 快速求区间和
37     preSum = [0] * (n + 1)
38     for i in range(1, n + 1):
39         preSum[i] = preSum[i - 1] + ord(input_str[i - 1])
40
41     # 存储分割方案的子串数目
42     res = []
43     DFS(preSum, 0, n, 0, res)
44
45     if not res:
```

```
46         print(0)
47     elif len(res) > 1:
48         print(-1)
49     else:
50         print(res[0])
51
52 if __name__ == "__main__":
53     main()
```

JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 function judge(num) {
9     // 判断是否为水仙花数
10    if (num < 100 || num > 999) return false;
11    let sum = 0, tmp = num;
12    while (tmp !== 0) {
13        let mod = tmp % 10;
14        sum += mod ** 3;
15        tmp = Math.floor(tmp / 10);
16    }
17    return sum === num;
18 }
19
20 // 递归回溯
21 function DFS(preSum, index, n, count, res) {
22     // 全部分割完成
23     if (index === n) {
24         res.push(count);
25         return;
26     }
27     for (let i = index + 1; i <= n; i++) {
28         const sum = preSum[i] - preSum[index];
29         if (judge(sum)) {
30             DFS(preSum, i, n, count + 1, res);
31         }
32     }
33 }
34
35 rl.on('line', function (input) {
36     const n = input.length;
37     if (n === 0) {
38         console.log(0);
39         process.exit();
40     }
41
42     // 预计算前缀和 快速求区间和
43     const preSum = new Array(n + 1).fill(0);
44     for (let i = 1; i <= n; i++) {
45         preSum[i] = preSum[i - 1] + input.charCodeAt(i - 1);
```

```
46     }
47
48     const res = [];
49     DFS(preSum, 0, n, 0, res);
50
51     if (res.length === 0) {
52         console.log(0);
53     } else if (res.length > 1) {
54         console.log(-1);
55     } else {
56         console.log(res[0]);
57     }
58
59     process.exit();
60 });
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 // 判断是否为水仙花数
10 func judge(num int) bool {
11     if num < 100 || num > 999 {
12         return false
13     }
14     sum, tmp := 0, num
15     for tmp != 0 {
16         mod := tmp % 10
17         sum += mod * mod * mod
18         tmp /= 10
19     }
20     return sum == num
21 }
22
23 // 递归回溯
24 func DFS(preSum []int, index, n, count int, res *[]int) {
25     if index == n {
26         *res = append(*res, count)
27         return
28     }
29     for i := index + 1; i <= n; i++ {
30         sum := preSum[i] - preSum[index]
31         if judge(sum) {
32             DFS(preSum, i, n, count+1, res)
33         }
34     }
35 }
36
37 func main() {
38     scanner := bufio.NewScanner(os.Stdin)
39     if !scanner.Scan() {
40         fmt.Println(0)
41         return
42     }
43     input := scanner.Text()
44     n := len(input)
45     if n == 0 {
```

```
46     fmt.Println(0)
47     return
48 }
49
50 // 预计算前缀和 快速求区间和
51 preSum := make([]int, n+1)
52 for i := 1; i <= n; i++ {
53     preSum[i] = preSum[i-1] + int(input[i-1])
54 }
55
56 var res []int
57 DFS(preSum, 0, n, 0, &res)
58
59 if len(res) == 0 {
60     fmt.Println(0)
61 } else if len(res) > 1 {
62     fmt.Println(-1)
63 } else {
64     fmt.Println(res[0])
65 }
66 }
```

| 来自: 华为OD机考 2025C卷 – 切割字符串 (C++ & Python & JAVA & JS & GO)-CSDN博客

| 来自: 华为OD机考 2025C卷 – 切割字符串 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考 2025C卷 - 哈夫曼树 (C++ & Python & JAVA & JS & GO)_哈弗曼树相关的考试题-CSDN博客

哈夫曼树

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

给定长度为 n 的无序的数字数组，每个数字代表二叉树的叶子节点的权值，数字数组的值均大于等于 1。请完成一个函数，根据输入的数字数组，生成[哈夫曼树]，并将哈夫曼树按照中序遍历输出。

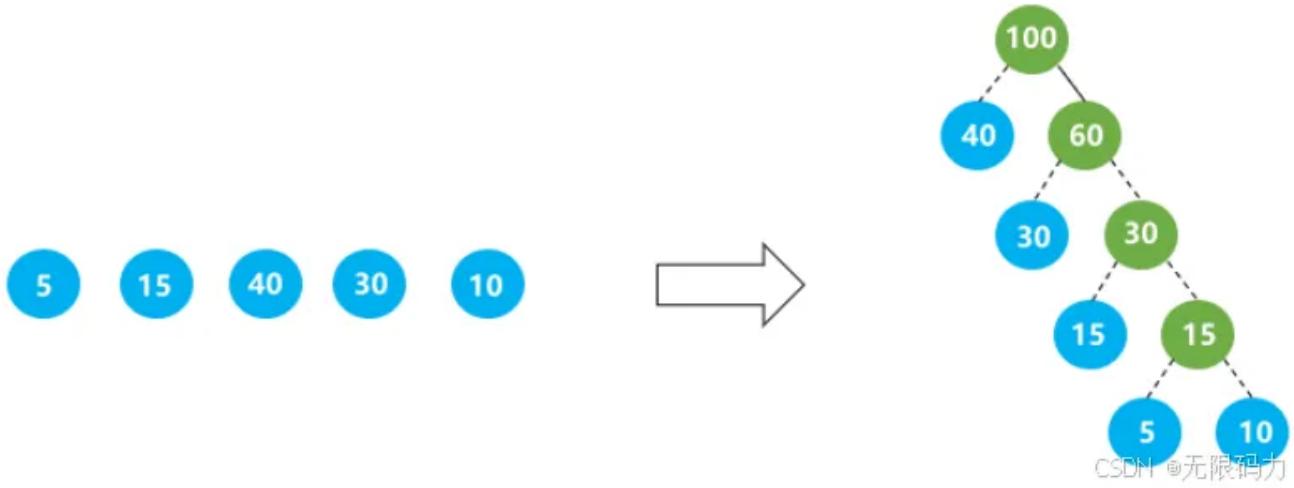
为了保证输出的[二叉树中序遍历]结果统一，增加以下限制:又树节点中，左节点权值小于等于右节点权值，根节点权值为左右节点权值之和。当左右节点权值相同时，左子树高度高度小于等于右子树。

注意: 所有用例保证有效，并能生成哈夫曼树提醒:哈夫曼树又称最优二叉树，是一种带权路径长度最短的一叉树。

所谓树的带权路径长度，就是树中所有的叶结点的权值乘上其到根结点的路径长度(若根结点为 0 00 层，叶结点到根结点的路径长度为叶结点的层数)

输入描述

例如：由叶子节点 5 15 40 30 10 生成的最优二叉树如下图所示，该树的最短带权路径长度为 $40 * 1 + 30 * 2 + 5 * 4 + 10 * 4 = 205$ 。



输出描述

输出一个哈夫曼的中序遍历数组，数值间以空格分隔

示例1

输入

```
▼ Plain Text
1 5
2 5 15 40 30 10
```

输出

```
▼ Plain Text
1 40 100 30 60 15 30 5 15 10
```

题解

思路： 哈夫曼树 是二叉树种的一个基础的数据结构。它的目标是 使得权值越小的节点离根越远，从而整体编码长度最小。

1. 使得权值 越小离根越远，从而可以发现，构建这棵树的总体逻辑是 从下到上 ，优先选择小的节点进行构建，获取小的节点的这个操作可以通过 优先队列 进行加速。
2. 构建哈夫曼树的基本处理过程 如下：
 - a. 将所有的叶子节点放入优先队列中(值小的位于栈顶)。

- b. 每次从优先队列中拿出两个节点(nodeA ,nodeB),新建一个新的节点 parent 作为这两个节点的父节点, 新节点的值为 nodeA.value + nodeB.value ,将 nodeA nodeB 值小的作为 parent 的左节点,另一个作为右节点。
 - c. 重复2的操作, 知道优先队列中只存在一个节点结束。此时优先队列的节点就是哈夫曼树的根节点。
3. 获取到哈夫曼树的根节点之后, 使用 DFS 进行中序遍历输出对应结果即可。

C++

```
1 #include <cstdio>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<queue>
8 #include<algorithm>
9 #include <functional>
10 using namespace std;
11
12
13 struct Node {
14     int value; // 节点的值
15     int num; // 节点子树拥有的节点数量
16     Node* left; // 左节点
17     Node* right; // 右节点
18     Node(int value, int num):value(value), left(nullptr), right(nullptr), n
um(num){}
19 };
20
21 // 自定义排序
22 struct CompareItem {
23     bool operator()(const Node* a, const Node* b) {
24         if (a->value == b->value) {
25             return a->num > b->num;
26         }
27         return a->value > b->value;
28     }
29 };
30
31 // 中序遍历
32 void dfs(Node* root) {
33     if (nullptr == root) {
34         return;
35     }
36     dfs(root->left);
37     cout << root->value << " ";
38     dfs(root->right);
39 }
40
41 int main() {
42     int n ;
43     cin >>n;
44     // 优先队列，自定义实现排序
```

```
45     priority_queue<Node*, vector<Node*>, CompareItem> pq;
46     for (int i = 0; i < n; i++) {
47         int tmp;
48         cin>>tmp;
49         pq.emplace(new Node(tmp, 1));
50     }
51
52     Node* root = nullptr;
53     while (pq.size() != 1) {
54         Node* left = pq.top();
55         pq.pop();
56         Node* right = pq.top();
57         pq.pop();
58         Node* parent = new Node(left->value + right->value, left->num +
right->num + 1);
59         parent->left = left;
60         parent->right = right;
61         pq.emplace(parent);
62     }
63     root = pq.top();
64
65     dfs(root);
66     return 0;
67
68 }
```

Java

```
1 import java.util.*;
2
3 class Node {
4     int value; // 节点的值
5     int num;   // 子树拥有的节点数量
6     Node left, right;
7
8     Node(int value, int num) {
9         this.value = value;
10        this.num = num;
11        this.left = null;
12        this.right = null;
13    }
14 }
15
16 // 自定义排序 (按照节点值升序, 若值相同则按照子树大小升序)
17 class CompareItem implements Comparator<Node> {
18     public int compare(Node a, Node b) {
19         if (a.value == b.value) {
20             return Integer.compare(a.num, b.num);
21         }
22         return Integer.compare(a.value, b.value);
23     }
24 }
25
26 public class Main {
27     // 中序遍历 (递归)
28     static void dfs(Node root) {
29         if (root == null) return;
30         dfs(root.left);
31         System.out.print(root.value + " ");
32         dfs(root.right);
33     }
34
35     public static void main(String[] args) {
36         Scanner scanner = new Scanner(System.in);
37         int n = scanner.nextInt();
38         PriorityQueue<Node> pq = new PriorityQueue<>(new CompareItem());
39
40         for (int i = 0; i < n; i++) {
41             int tmp = scanner.nextInt();
42             pq.offer(new Node(tmp, 1));
43         }
44         scanner.close();
45     }
}
```

```
46     Node root = null;
47     while (pq.size() > 1) {
48         Node left = pq.poll();
49         Node right = pq.poll();
50         Node parent = new Node(left.value + right.value, left.num + ri-
51             ght.num + 1);
52         parent.left = left;
53         parent.right = right;
54         pq.offer(parent);
55     }
56     root = pq.poll();
57     dfs(root);
58 }
59 }
```

Python

```
1 import sys
2 import heapq
3
4 class Node:
5     def __init__(self, value, num):
6         self.value = value # 节点的值
7         self.num = num # 子树拥有的节点数量
8         self.left = None # 左子节点
9         self.right = None # 右子节点
10
11    def __lt__(self, other):
12        if self.value == other.value:
13            return self.num < other.num # 若值相同，则按子树大小排序
14        return self.value < other.value # 按值排序
15
16    def dfs(root):
17        if root is None:
18            return
19        dfs(root.left)
20        print(root.value, end=" ")
21        dfs(root.right)
22
23    def main():
24        n = int(sys.stdin.readline().strip())
25        pq = []
26        ans = list(map(int, sys.stdin.read().split()))
27        for _ in range(n):
28            tmp = ans[_]
29            heapq.heappush(pq, Node(tmp, 1))
30
31        root = None
32        while len(pq) > 1:
33            left = heapq.heappop(pq)
34            right = heapq.heappop(pq)
35            parent = Node(left.value + right.value, left.num + right.num + 1)
36            parent.left = left
37            parent.right = right
38            heapq.heappush(pq, parent)
39
40        root = heapq.heappop(pq)
41        dfs(root)
42
43    if __name__ == "__main__":
44        main()
```

JavaScript

```
1 const readline = require("readline");
2
3 class Node {
4     constructor(value, num) {
5         this.value = value; // 节点的值
6         this.num = num; // 子树的节点数量
7         this.left = null; // 左子节点
8         this.right = null; // 右子节点
9     }
10 }
11
12 // 自定义排序规则 (优先队列)
13 class PriorityQueue {
14     constructor() {
15         this.data = [];
16     }
17
18     push(node) {
19         this.data.push(node);
20         this.data.sort((a, b) => {
21             if (a.value === b.value) {
22                 return a.num - b.num; // 如果值相等, 按子树大小排序
23             }
24             return a.value - b.value; // 按值从小到大排序
25         });
26     }
27
28     pop() {
29         return this.data.shift();
30     }
31
32     size() {
33         return this.data.length;
34     }
35
36     top() {
37         return this.data[0];
38     }
39 }
40
41 function dfs(root) {
42     if (!root) return;
43     dfs(root.left);
44     process.stdout.write(root.value + " ");
45     dfs(root.right);
```

```
46  }
47
48  const rl = readline.createInterface({
49    input: process.stdin,
50    output: process.stdout
51  });
52
53  let inputs = [];
54  rl.on("line", (line) => {
55    inputs.push(line.trim());
56    if (inputs.length === 2) {
57      rl.close();
58    }
59  });
60
61  rl.on("close", () => {
62    let n = parseInt(inputs[0]); // 读取 n
63    let values = inputs[1].split(" ").map(Number); // 读取数组并转换为数字
64
65    if (values.length !== n) {
66      console.error("输入数据不匹配");
67      return;
68    }
69
70    let pq = new PriorityQueue();
71    for (let i = 0; i < n; i++) {
72      pq.push(new Node(values[i], 1));
73    }
74
75    let root = null;
76    while (pq.size() > 1) {
77      let left = pq.pop();
78      let right = pq.pop();
79      let parent = new Node(left.value + right.value, left.num + right.n
80      um + 1);
81      parent.left = left;
82      parent.right = right;
83      pq.push(parent);
84    }
85    root = pq.top();
86
87    dfs(root);
88    console.log(); // 换行
});
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8     "strconv"
9     "strings"
10    )
11
12 // Node 结构体
13 type Node struct {
14     value int
15     num   int
16     left  *Node
17     right *Node
18 }
19
20 // PriorityQueue 实现最小堆
21 type PriorityQueue []*Node
22
23 func (pq PriorityQueue) Len() int { return len(pq) }
24
25 func (pq PriorityQueue) Less(i, j int) bool {
26     if pq[i].value == pq[j].value {
27         return pq[i].num < pq[j].num // 若值相等，按子树大小排序
28     }
29     return pq[i].value < pq[j].value // 按值从小到大排序
30 }
31
32 func (pq PriorityQueue) Swap(i, j int) {
33     pq[i], pq[j] = pq[j], pq[i]
34 }
35
36 func (pq *PriorityQueue) Push(x interface{}) {
37     *pq = append(*pq, x.(*Node))
38 }
39
40 func (pq *PriorityQueue) Pop() interface{} {
41     old := *pq
42     n := len(old)
43     item := old[n-1]
44     *pq = old[0 : n-1]
45     return item
```

```

46    }
47
48    // 中序遍历
49    func dfs(root *Node) {
50        if root == nil {
51            return
52        }
53        dfs(root.left)
54        fmt.Println(root.value, " ")
55        dfs(root.right)
56    }
57
58    func main() {
59        reader := bufio.NewReader(os.Stdin)
60
61        // 读取 n
62        line, _ := reader.ReadString('\n')
63        n, _ := strconv.Atoi(strings.TrimSpace(line))
64
65        // 读取数组
66        line, _ = reader.ReadString('\n')
67        strs := strings.Fields(line)
68
69        if len(strs) != n {
70            fmt.Println("输入数据不匹配")
71            return
72        }
73
74        pq := &PriorityQueue{}
75        heap.Init(pq)
76
77        // 读取输入数据并加入优先队列
78        for _, s := range strs {
79            value, _ := strconv.Atoi(s)
80            heap.Push(pq, &Node{value: value, num: 1})
81        }
82
83        // 构建哈夫曼树
84        var root *Node
85        for pq.Len() > 1 {
86            left := heap.Pop(pq).(*Node)
87            right := heap.Pop(pq).(*Node)
88            parent := &Node{value: left.value + right.value, num: left.num + right.num + 1, left: left, right: right}
89            heap.Push(pq, parent)
90        }
91        root = heap.Pop(pq).(*Node)
92

```

```
93     // 中序遍历输出
94     dfs(root)
95     fmt.Println()
96 }
```

来自: 华为OD机考 2025C卷 – 哈夫曼树 (C++ & Python & JAVA & JS & GO)_哈弗曼树相关的考试题–CSDN博客

华为OD机试 2025C卷 - 代码编辑器 (C++ & Python & JAVA & JS & GO)_华为代码编辑器-csdn-CSDN博客

代码编辑器

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

某公司为了更高效的编写代码，邀请你开发一款代码编辑器程序。

程序的输入为 已有的代码文本和指令序列，程序需输出编辑后的最终文本。指针初始位置位于文本的开头。

支持的指令(X为大于等于0的整数, word 为无空格的字符串):

- FORWARD X 指针向前(右)移动X,如果指针移动位置超过了文本末尾，则将指针移动到文本末尾
- BACKWARD X 指针向后(左)移动X,如果指针移动位置超过了文本开头，则将指针移动到文本开头
- SEARCH-FORWARD word 从指针当前位置向前查找 word 并将指针移动到word的起始位置，如果未找到则保持不变
- SEARCH-BACKWARD word 在文本中向后查找 word 并将指针移动到word的起始位置，如果未找到则保持不变
- INSERT word 在指针当前位置前插入word，并将指针移动到word的结尾
- REPLACE word 在指针当前位置替换并插入字符(删除原有字符，并增加新的字符)
- DELETE X 在指针位置删除X个字符

| 备注： 文本最长长度不超过 256K

输入描述

输入的第一行为命令列表的长度K

输入的第二行为文件中的原始文本

接下来的K行，每行为一个指令

输出描述

编辑后的最终结果

用例1

输入

```
1 1  
2 ello  
3 INSERT h
```

Plain Text |

输出

```
1 hello
```

Plain Text |

说明

| 在文本开头插入

用例2

输入

```
1 2  
2 hillo  
3 FORWARD 1  
4 INSERT e
```

Plain Text |

输出

```
1 hello
```

Plain Text |

说明

| 在文本的第一个位置插入

用例3

输入

```
1 2
2 hell
3 FORWARD 1000
4 INSERT o
```

Plain Text

输出

```
1 hello
```

Plain Text

说明

| 在文本的结尾插入

用例4

输入

```
1 1
2 hello
3 REPLACE HELLO
```

Plain Text

输出

```
1 HELLO
```

Plain Text

题解

思路： 模拟题，参照题目描述实现对应指令操作就行。

注意以下几个点即可：

1. `INSERT word`：指针需要更新到插入word的最后一个字符后面。
2. `SEARCH-FORWARD`：从当前指针，向左查找第一个匹配字符串。
3. `SEARCH-BACKWARD`：从当前指针，向右查找第一个匹配字符串。

C++

```
1 #include <iostream>
2 #include <string>
3 #include <vector>
4 #include <sstream>
5 #include <algorithm>
6
7 using namespace std;
8
9 int main() {
10     int k;
11     cin >> k;
12     cin.ignore(); // 清除换行符
13
14     string text;
15     getline(cin, text);
16
17     vector<pair<string, string>> commands;
18     for (int i = 0; i < k; ++i) {
19         string line;
20         getline(cin, line);
21         istringstream iss(line);
22         string cmd, arg;
23         iss >> cmd;
24         if (iss >> arg) {
25             commands.emplace_back(cmd, arg);
26         } else {
27             commands.emplace_back(cmd, "");
28         }
29     }
30
31     int curIdx = 0;
32
33     for (const auto& [cmd, arg] : commands) {
34         if (cmd == "FORWARD") {
35             int x = stoi(arg);
36             curIdx = min(curIdx + x, (int)text.length());
37         } else if (cmd == "BACKWARD") {
38             int x = stoi(arg);
39             curIdx = max(curIdx - x, 0);
40         } else if (cmd == "SEARCH-FORWARD") {
41             size_t pos = text.find(arg, curIdx);
42             if (pos != string::npos) {
43                 curIdx = pos;
44             }
45         } else if (cmd == "SEARCH-BACKWARD") {
```

```
46     size_t found = text.rfind(arg, curIdx);
47     if (found != string::npos) {
48         curIdx = found;
49     }
50 } else if (cmd == "INSERT") {
51     text.insert(curIdx, arg);
52     curIdx += arg.length();
53 } else if (cmd == "REPLACE") {
54     // 替换当前位置开始长度为 arg.length() 的内容
55     int len = arg.length();
56     if (curIdx + len > (int)text.size()) {
57         text.replace(curIdx, text.size() - curIdx, arg);
58     } else {
59         text.replace(curIdx, len, arg);
60     }
61 } else if (cmd == "DELETE") {
62     int len = stoi(arg);
63     if (curIdx < (int)text.size()) {
64         int toDelete = min(len, (int)text.size() - curIdx);
65         text.erase(curIdx, toDelete);
66     }
67 }
68 }
69 cout << text << endl;
70 return 0;
71 }
```

JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int k = Integer.parseInt(sc.nextLine()); // 命令数量
7         String text = sc.nextLine(); // 初始文本
8
9         String[] commands = new String[k];
10        for (int i = 0; i < k; i++) {
11            commands[i] = sc.nextLine(); // 读取每条命令
12        }
13
14        StringBuilder sb = new StringBuilder(text);
15        int curIdx = 0; // 当前索引位置
16
17        for (String line : commands) {
18            String[] parts = line.split(" ", 2); // 按空格拆分, 最多拆两部分
19            String cmd = parts[0]; // 命令名
20            String arg = parts.length > 1 ? parts[1] : ""; // 命令参数
21
22            switch (cmd) {
23                case "FORWARD":
24                    // 光标向后移动x步, 但不超过文本长度
25                    curIdx = Math.min(curIdx + Integer.parseInt(arg), sb.length());
26                    break;
27                case "BACKWARD":
28                    // 光标向前移动x步, 但不小于0
29                    curIdx = Math.max(curIdx - Integer.parseInt(arg), 0);
30                    break;
31                case "SEARCH-FORWARD":
32                    // 从当前光标开始搜索指定字符串, 找到则光标移动到匹配位置
33                    int pos = sb.indexOf(arg, curIdx);
34                    if (pos != -1) {
35                        curIdx = pos;
36                    }
37                    break;
38                case "SEARCH-BACKWARD":
39                    // 从当前光标向前搜索指定字符串, 找到则光标移动到匹配位置
40                    int posBack = sb.lastIndexOf(arg, curIdx);
41                    if (posBack != -1) {
42                        curIdx = posBack;
43                    }
44                    break;
45            }
46        }
47    }
48}
```

```
45     case "INSERT":  
46         // 在当前光标处插入字符串，光标向后移动插入长度  
47         sb.insert(curIdx, arg);  
48         curIdx += arg.length();  
49         break;  
50     case "REPLACE":  
51         // 替换当前位置开始长度为参数长度的字符串  
52         int len = arg.length();  
53         int end = Math.min(curIdx + len, sb.length());  
54         sb.replace(curIdx, end, arg);  
55         break;  
56     case "DELETE":  
57         // 删除当前位置开始指定长度的字符串，防止越界  
58         int delLen = Integer.parseInt(arg);  
59         if (curIdx < sb.length()) {  
60             int toDelete = Math.min(delLen, sb.length() - curI  
61             dx);  
62             sb.delete(curIdx, curIdx + toDelete);  
63         }  
64         break;  
65     }  
66 }  
67 System.out.println(sb.toString()); // 输出最终文本  
68 }  
69 }
```

Python

```
1 import sys
2
3 k = int(sys.stdin.readline().strip()) # 命令数量
4 text = sys.stdin.readline().rstrip('\n') # 读取初始文本
5
6 curIdx = 0 # 当前索引位置
7
8 for _ in range(k):
9     line = sys.stdin.readline().rstrip('\n')
10    parts = line.split(' ', 1) # 最多拆成两个部分, 命令和参数
11    cmd = parts[0]
12    arg = parts[1] if len(parts) > 1 else ""
13
14    if cmd == "FORWARD":
15        # 向后移动光标, 但不超过文本长度
16        curIdx = min(curIdx + int(arg), len(text))
17    elif cmd == "BACKWARD":
18        # 向前移动光标, 但不小于0
19        curIdx = max(curIdx - int(arg), 0)
20    elif cmd == "SEARCH-FORWARD":
21        # 向后搜索字符串, 找到则移动光标
22        pos = text.find(arg, curIdx)
23        if pos != -1:
24            curIdx = pos
25    elif cmd == "SEARCH-BACKWARD":
26        # 向前搜索字符串, 找到则移动光标, 注意rfind的结束索引是开区间, +1
27        pos = text.rfind(arg, 0, curIdx + 1)
28        if pos != -1:
29            curIdx = pos
30    elif cmd == "INSERT":
31        # 在光标处插入字符串, 更新光标位置
32        text = text[:curIdx] + arg + text[curIdx:]
33        curIdx += len(arg)
34    elif cmd == "REPLACE":
35        # 替换光标开始, 长度为参数长度的字符串
36        length = len(arg)
37        if curIdx + length > len(text):
38            text = text[:curIdx] + arg
39        else:
40            text = text[:curIdx] + arg + text[curIdx + length:]
41    elif cmd == "DELETE":
42        # 删除光标开始, 指定长度的字符串
43        length = int(arg)
44        if curIdx < len(text):
45            text = text[:curIdx] + text[curIdx + length:]
```

46

```
print(text) # 输出结果
```

JavaScript


```
46         break;
47     case "REPLACE":
48         // 替换字符串
49         let len = arg.length;
50         if (curIdx + len > text.length) {
51             text = text.slice(0, curIdx) + arg;
52         } else {
53             text = text.slice(0, curIdx) + arg + text.slice(curId
54             x + len);
55         }
56         break;
57     case "DELETE":
58         // 删除字符串
59         let delLen = parseInt(arg);
60         if (curIdx < text.length) {
61             text = text.slice(0, curIdx) + text.slice(curIdx + del
62             Len);
63         }
64     }
65     console.log(text); // 输出结果
66 });
```

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.NewScanner(os.Stdin)
13
14     // 读取命令数量 k
15     reader.Scan()
16     k, _ := strconv.Atoi(reader.Text())
17
18     // 读取初始文本内容
19     reader.Scan()
20     text := reader.Text()
21
22     // 当前光标索引
23     curIdx := 0
24
25     // 循环读取每条命令并执行
26     for i := 0; i < k; i++ {
27         reader.Scan()
28         line := reader.Text()
29         parts := strings.SplitN(line, " ", 2)
30         cmd := parts[0]
31         arg := ""
32         if len(parts) > 1 {
33             arg = parts[1]
34         }
35
36         switch cmd {
37         case "FORWARD":
38             // 光标向后移动 x 步
39             x, _ := strconv.Atoi(arg)
40             curIdx += x
41             if curIdx > len(text) {
42                 curIdx = len(text)
43             }
44
45         case "BACKWARD":
```

```
46 // 光标向前移动 x 步
47 x, _ := strconv.Atoi(arg)
48 curIdx -= x
49 if curIdx < 0 {
50     curIdx = 0
51 }
52

53 case "SEARCH-FORWARD":
54     // 从当前光标位置向后查找 arg 字符串
55     pos := strings.Index(text[curIdx:], arg)
56     if pos != -1 {
57         curIdx += pos
58     }
59

60 case "SEARCH-BACKWARD":
61     // 从当前光标位置向前查找 arg 字符串
62     found := -1
63     for j := curIdx; j >= 0; j-- {
64         if j+len(arg) <= len(text) && text[j:j+len(arg)] == arg {
65             found = j
66             break
67         }
68     }
69     if found != -1 {
70         curIdx = found
71     }
72

73 case "INSERT":
74     // 在当前光标处插入 arg 字符串
75     text = text[:curIdx] + arg + text[curIdx:]
76     curIdx += len(arg)
77

78 case "REPLACE":
79     // 替换当前位置起始长度为 len(arg) 的内容
80     end := curIdx + len(arg)
81     if end > len(text) {
82         text = text[:curIdx] + arg
83     } else {
84         text = text[:curIdx] + arg + text[end:]
85     }
86

87 case "DELETE":
88     // 从当前光标删除指定长度
89     delLen, _ := strconv.Atoi(arg)
90     if curIdx < len(text) {
91         toDelete := delLen
92         if curIdx+toDelete > len(text) {
93             toDelete = len(text) - curIdx
```

```
94         }
95         text = text[:curIdx] + text[curIdx+toDelete:]
96     }
97 }
98 }
99 }
100 // 输出最终结果
101 fmt.Println(text)
102 }
```

来自: 华为OD机试 2025C卷 – 代码编辑器 (C++ & Python & JAVA & JS & GO)_华为代码编辑器–csdn–CSDN博客

华为OD机试 2025C卷 - 构成正方形的数量 (C++ & Python & JAVA & JS & GO)_华为od机试2025a 卷 - 构成正方形的数量-CSDN博客

构成正方形的数量

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

输入N个互不相同的二维整数坐标，求这N个坐标可以构成的正方形数量。[内积为零的两个向量垂直]

输入描述

第一行输入为N，N代表坐标数量，N为正整数。N <= 100

之后的 K 行输入为坐标x y以空格分隔，x, y为整数，-10<=x, y<=10

输出描述

输出可以构成的正方形数量。

示例1

输入

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

Plain Text

输出

```
1 0
```

Plain Text

说明

(3个点不足以构成正方形)

示例2

输入

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

Plain Text

输出

```
1 1
```

Plain Text

题解

思路：

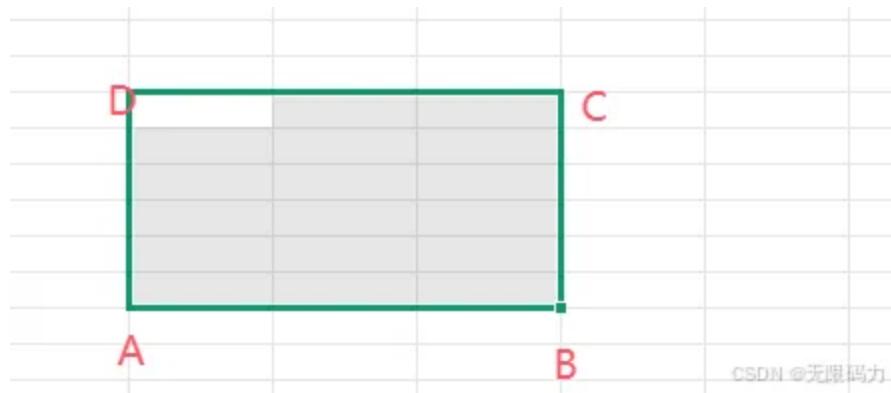
利用的是正方形中的边的向量关系进行求解。假设A(x_1, y_1)和B(x_2, y_2)是正方形的两条边。组成的向量AB为 $\{x_2 - x_1, y_2 - y_1\}$ 。

注意向量旋转的两个公式：

```
1 逆时针旋转90°, (x, y) -> (-y, x)  
2 顺时针旋转90°, (x, y) -> (y, -x)
```

Plain Text

情况一：(图画的不是很准确应该是正方形的)



CSDN @无限码力

BA向量的值为 $\{x_1-x_2, y_1-y_2\}$, 逆时针旋转可以得出AD、BC向量的值 => $x_3 = x_1 - (y_1 - y_2)$, $y_3 = y_1 + (x_1 - x_2)$, $x_4=x_2 - (y_1 - y_2)$, $y_4= y_2 + (x_1 - x_2)$

情况二: (图画的不是很准确应该是正方形的)



可以推导出 $x_3 = x_2 + (y_1 - y_2)$, $y_3=y_2 - (x_1 - x_2)$, $x_4=x_1+(y_1-y_2)$, $y_4=y_3 - (x_1 - x_2)$

根据上述推到过程, 可以枚举两个点, 然后去判断两外两个点是否存在来[判断是否存在](#)正方形。四个点两两组合, 一共会组合出4次, 所以最终结果 = 统计结果 / 4。

C++

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 // 定义一个点结构体
6 struct Point {
7     int x, y;
8 };
9
10 // 判断两个点是否相等
11 bool arePointsEqual(Point a, Point b) {
12     return a.x == b.x && a.y == b.y;
13 }
14
15 // 检查数组中是否存在某点
16 bool pointExists(vector<Point>& points, Point p) {
17     for (auto& point : points) {
18         if (arePointsEqual(point, p)) {
19             return true;
20         }
21     }
22     return false;
23 }
24
25 int main() {
26     int n;
27     cin >> n;
28
29     vector<Point> points(n);
30
31     // 读取坐标并存入数组
32     for (int i = 0; i < n; i++) {
33         cin >> points[i].x >> points[i].y;
34     }
35
36     int squareCount = 0;
37
38     // 遍历所有点对，检查是否能构成正方形
39     for (int i = 0; i < n; i++) {
40         int x1 = points[i].x;
41         int y1 = points[i].y;
42
43         for (int j = i + 1; j < n; j++) {
44             int x2 = points[j].x;
45             int y2 = points[j].y;
```

```
46  
47     // 计算两个可能的对角点  
48     Point p3 = {x1 - (y1 - y2), y1 + (x1 - x2)};  
49     Point p4 = {x2 - (y1 - y2), y2 + (x1 - x2)};  
50  
51     if (pointExists(points, p3) && pointExists(points, p4)) {  
52         squareCount++;  
53     }  
54  
55     // 计算另外两个可能的对角点  
56     Point p5 = {x1 + (y1 - y2), y1 - (x1 - x2)};  
57     Point p6 = {x2 + (y1 - y2), y2 - (x1 - x2)};  
58  
59     if (pointExists(points, p5) && pointExists(points, p6)) {  
60         squareCount++;  
61     }  
62     }  
63 }  
64  
65 // 每个正方形被计算了4次，因此结果需要除以4  
66 cout << squareCount / 4 << endl;  
67  
68     return 0;  
69 }
```

Java

```
1 import java.util.*;
2
3 class Point {
4     int x, y;
5
6     Point(int x, int y) {
7         this.x = x;
8         this.y = y;
9     }
10
11    // 判断两个点是否相等
12    @Override
13    public boolean equals(Object obj) {
14        if (obj instanceof Point) {
15            Point p = (Point) obj;
16            return this.x == p.x && this.y == p.y;
17        }
18        return false;
19    }
20
21    @Override
22    public int hashCode() {
23        return Objects.hash(x, y);
24    }
25 }
26
27 public class Main {
28     public static void main(String[] args) {
29         Scanner scanner = new Scanner(System.in);
30         int n = scanner.nextInt();
31
32         Set<Point> points = new HashSet<>();
33         List<Point> pointList = new ArrayList<>();
34
35         // 读取坐标并存入集合和列表
36         for (int i = 0; i < n; i++) {
37             int x = scanner.nextInt();
38             int y = scanner.nextInt();
39             Point p = new Point(x, y);
40             points.add(p);
41             pointList.add(p);
42         }
43
44         int squareCount = 0;
```

```
46 // 遍历所有点对，检查是否能构成正方形
47 for (int i = 0; i < n; i++) {
48     Point p1 = pointList.get(i);
49
50     for (int j = i + 1; j < n; j++) {
51         Point p2 = pointList.get(j);
52
53         // 计算两个可能的对角点
54         Point p3 = new Point(p1.x - (p1.y - p2.y), p1.y + (p1.x -
55             p2.x));
56         Point p4 = new Point(p2.x - (p1.y - p2.y), p2.y + (p1.x -
57             p2.x));
58
58         if (points.contains(p3) && points.contains(p4)) {
59             squareCount++;
60         }
61
62         // 计算另外两个可能的对角点
63         Point p5 = new Point(p1.x + (p1.y - p2.y), p1.y - (p1.x -
64             p2.x));
65         Point p6 = new Point(p2.x + (p1.y - p2.y), p2.y - (p1.x -
66             p2.x));
67
68         if (points.contains(p5) && points.contains(p6)) {
69             squareCount++;
70         }
71     }
72
73     // 每个正方形被计算了4次，因此结果需要除以4
74     System.out.println(squareCount / 4);
75     scanner.close();
76 }
```

Python

```
1 import sys
2
3 # 读取输入
4 n = int(sys.stdin.readline().strip())
5 points = set()
6 point_list = []
7
8 # 读取坐标并存入集合和列表
9 for _ in range(n):
10     x, y = map(int, sys.stdin.readline().strip().split())
11     point = (x, y)
12     points.add(point)
13     point_list.append(point)
14
15 square_count = 0
16
17 # 遍历所有点对，检查是否能构成正方形
18 for i in range(n):
19     x1, y1 = point_list[i]
20
21     for j in range(i + 1, n):
22         x2, y2 = point_list[j]
23
24         # 计算两个可能的对角点
25         p3 = (x1 - (y1 - y2), y1 + (x1 - x2))
26         p4 = (x2 - (y1 - y2), y2 + (x1 - x2))
27
28         if p3 in points and p4 in points:
29             square_count += 1
30
31         # 计算另外两个可能的对角点
32         p5 = (x1 + (y1 - y2), y1 - (x1 - x2))
33         p6 = (x2 + (y1 - y2), y2 - (x1 - x2))
34
35         if p5 in points and p6 in points:
36             square_count += 1
37
38 # 每个正方形被计算了4次，因此结果需要除以4
39 print(square_count // 4)
```

JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 const points = new Set();
9 const pointList = [];
10 let n;
11 let lineCount = 0;
12
13 rl.on("line", (line) => {
14     if (lineCount === 0) {
15         n = parseInt(line.trim());
16     } else {
17         const [x, y] = line.trim().split(" ").map(Number);
18         const key = `${x},${y}`;
19         points.add(key);
20         pointList.push([x, y]);
21     }
22
23     lineCount++;
24
25     if (lineCount > n) {
26         rl.close();
27     }
28 });
29
30 rl.on("close", () => {
31     let squareCount = 0;
32
33     // 遍历所有点对，检查是否能构成正方形
34     for (let i = 0; i < n; i++) {
35         const [x1, y1] = pointList[i];
36
37         for (let j = i + 1; j < n; j++) {
38             const [x2, y2] = pointList[j];
39
40             // 计算两个可能的对角点
41             const p3 = `${x1 - (y1 - y2)}, ${y1 + (x1 - x2)}`;
42             const p4 = `${x2 - (y1 - y2)}, ${y2 + (x1 - x2)}`;
43
44             if (points.has(p3) && points.has(p4)) {
45                 squareCount++;
46             }
47         }
48     }
49
50     console.log(`Found ${squareCount} squares`);
51 });
52
53 rl.on("SIGINT", () => {
54     rl.close();
55 });
56
57 rl.on("SIGTERM", () => {
58     rl.close();
59 });
60
61 rl.on("SIGKILL", () => {
62     rl.close();
63 });
64
65 rl.on("SIGQUIT", () => {
66     rl.close();
67 });
68
69 rl.on("SIGPOLL", () => {
70     rl.close();
71 });
72
73 rl.on("SIGPIPE", () => {
74     rl.close();
75 });
76
77 rl.on("SIGALRM", () => {
78     rl.close();
79 });
80
81 rl.on("SIGCHLD", () => {
82     rl.close();
83 });
84
85 rl.on("SIGSTOP", () => {
86     rl.close();
87 });
88
89 rl.on("SIGCONT", () => {
90     rl.close();
91 });
92
93 rl.on("SIGPOLLNORM", () => {
94     rl.close();
95 });
96
97 rl.on("SIGPOLLERR", () => {
98     rl.close();
99 });
100 rl.on("SIGPOLLHUP", () => {
101     rl.close();
102 });
103
104 rl.on("SIGPOLLNORM", () => {
105     rl.close();
106 });
107
108 rl.on("SIGPOLLERR", () => {
109     rl.close();
110 });
111
112 rl.on("SIGPOLLHUP", () => {
113     rl.close();
114 });
115
116 rl.on("SIGPOLLNORM", () => {
117     rl.close();
118 });
119
120 rl.on("SIGPOLLERR", () => {
121     rl.close();
122 });
123
124 rl.on("SIGPOLLHUP", () => {
125     rl.close();
126 });
127
128 rl.on("SIGPOLLNORM", () => {
129     rl.close();
130 });
131
132 rl.on("SIGPOLLERR", () => {
133     rl.close();
134 });
135
136 rl.on("SIGPOLLHUP", () => {
137     rl.close();
138 });
139
140 rl.on("SIGPOLLNORM", () => {
141     rl.close();
142 });
143
144 rl.on("SIGPOLLERR", () => {
145     rl.close();
146 });
147
148 rl.on("SIGPOLLHUP", () => {
149     rl.close();
150 });
151
152 rl.on("SIGPOLLNORM", () => {
153     rl.close();
154 });
155
156 rl.on("SIGPOLLERR", () => {
157     rl.close();
158 });
159
160 rl.on("SIGPOLLHUP", () => {
161     rl.close();
162 });
163
164 rl.on("SIGPOLLNORM", () => {
165     rl.close();
166 });
167
168 rl.on("SIGPOLLERR", () => {
169     rl.close();
170 });
171
172 rl.on("SIGPOLLHUP", () => {
173     rl.close();
174 });
175
176 rl.on("SIGPOLLNORM", () => {
177     rl.close();
178 });
179
180 rl.on("SIGPOLLERR", () => {
181     rl.close();
182 });
183
184 rl.on("SIGPOLLHUP", () => {
185     rl.close();
186 });
187
188 rl.on("SIGPOLLNORM", () => {
189     rl.close();
190 });
191
192 rl.on("SIGPOLLERR", () => {
193     rl.close();
194 });
195
196 rl.on("SIGPOLLHUP", () => {
197     rl.close();
198 });
199
200 rl.on("SIGPOLLNORM", () => {
201     rl.close();
202 });
203
204 rl.on("SIGPOLLERR", () => {
205     rl.close();
206 });
207
208 rl.on("SIGPOLLHUP", () => {
209     rl.close();
210 });
211
212 rl.on("SIGPOLLNORM", () => {
213     rl.close();
214 });
215
216 rl.on("SIGPOLLERR", () => {
217     rl.close();
218 });
219
220 rl.on("SIGPOLLHUP", () => {
221     rl.close();
222 });
223
224 rl.on("SIGPOLLNORM", () => {
225     rl.close();
226 });
227
228 rl.on("SIGPOLLERR", () => {
229     rl.close();
230 });
231
232 rl.on("SIGPOLLHUP", () => {
233     rl.close();
234 });
235
236 rl.on("SIGPOLLNORM", () => {
237     rl.close();
238 });
239
240 rl.on("SIGPOLLERR", () => {
241     rl.close();
242 });
243
244 rl.on("SIGPOLLHUP", () => {
245     rl.close();
246 });
247
248 rl.on("SIGPOLLNORM", () => {
249     rl.close();
250 });
251
252 rl.on("SIGPOLLERR", () => {
253     rl.close();
254 });
255
256 rl.on("SIGPOLLHUP", () => {
257     rl.close();
258 });
259
260 rl.on("SIGPOLLNORM", () => {
261     rl.close();
262 });
263
264 rl.on("SIGPOLLERR", () => {
265     rl.close();
266 });
267
268 rl.on("SIGPOLLHUP", () => {
269     rl.close();
270 });
271
272 rl.on("SIGPOLLNORM", () => {
273     rl.close();
274 });
275
276 rl.on("SIGPOLLERR", () => {
277     rl.close();
278 });
279
280 rl.on("SIGPOLLHUP", () => {
281     rl.close();
282 });
283
284 rl.on("SIGPOLLNORM", () => {
285     rl.close();
286 });
287
288 rl.on("SIGPOLLERR", () => {
289     rl.close();
290 });
291
292 rl.on("SIGPOLLHUP", () => {
293     rl.close();
294 });
295
296 rl.on("SIGPOLLNORM", () => {
297     rl.close();
298 });
299
300 rl.on("SIGPOLLERR", () => {
301     rl.close();
302 });
303
304 rl.on("SIGPOLLHUP", () => {
305     rl.close();
306 });
307
308 rl.on("SIGPOLLNORM", () => {
309     rl.close();
310 });
311
312 rl.on("SIGPOLLERR", () => {
313     rl.close();
314 });
315
316 rl.on("SIGPOLLHUP", () => {
317     rl.close();
318 });
319
320 rl.on("SIGPOLLNORM", () => {
321     rl.close();
322 });
323
324 rl.on("SIGPOLLERR", () => {
325     rl.close();
326 });
327
328 rl.on("SIGPOLLHUP", () => {
329     rl.close();
330 });
331
332 rl.on("SIGPOLLNORM", () => {
333     rl.close();
334 });
335
336 rl.on("SIGPOLLERR", () => {
337     rl.close();
338 });
339
340 rl.on("SIGPOLLHUP", () => {
341     rl.close();
342 });
343
344 rl.on("SIGPOLLNORM", () => {
345     rl.close();
346 });
347
348 rl.on("SIGPOLLERR", () => {
349     rl.close();
350 });
351
352 rl.on("SIGPOLLHUP", () => {
353     rl.close();
354 });
355
356 rl.on("SIGPOLLNORM", () => {
357     rl.close();
358 });
359
360 rl.on("SIGPOLLERR", () => {
361     rl.close();
362 });
363
364 rl.on("SIGPOLLHUP", () => {
365     rl.close();
366 });
367
368 rl.on("SIGPOLLNORM", () => {
369     rl.close();
370 });
371
372 rl.on("SIGPOLLERR", () => {
373     rl.close();
374 });
375
376 rl.on("SIGPOLLHUP", () => {
377     rl.close();
378 });
379
380 rl.on("SIGPOLLNORM", () => {
381     rl.close();
382 });
383
384 rl.on("SIGPOLLERR", () => {
385     rl.close();
386 });
387
388 rl.on("SIGPOLLHUP", () => {
389     rl.close();
390 });
391
392 rl.on("SIGPOLLNORM", () => {
393     rl.close();
394 });
395
396 rl.on("SIGPOLLERR", () => {
397     rl.close();
398 });
399
400 rl.on("SIGPOLLHUP", () => {
401     rl.close();
402 });
403
404 rl.on("SIGPOLLNORM", () => {
405     rl.close();
406 });
407
408 rl.on("SIGPOLLERR", () => {
409     rl.close();
410 });
411
412 rl.on("SIGPOLLHUP", () => {
413     rl.close();
414 });
415
416 rl.on("SIGPOLLNORM", () => {
417     rl.close();
418 });
419
420 rl.on("SIGPOLLERR", () => {
421     rl.close();
422 });
423
424 rl.on("SIGPOLLHUP", () => {
425     rl.close();
426 });
427
428 rl.on("SIGPOLLNORM", () => {
429     rl.close();
430 });
431
432 rl.on("SIGPOLLERR", () => {
433     rl.close();
434 });
435
436 rl.on("SIGPOLLHUP", () => {
437     rl.close();
438 });
439
440 rl.on("SIGPOLLNORM", () => {
441     rl.close();
442 });
443
444 rl.on("SIGPOLLERR", () => {
445     rl.close();
446 });
447
448 rl.on("SIGPOLLHUP", () => {
449     rl.close();
450 });
451
452 rl.on("SIGPOLLNORM", () => {
453     rl.close();
454 });
455
456 rl.on("SIGPOLLERR", () => {
457     rl.close();
458 });
459
460 rl.on("SIGPOLLHUP", () => {
461     rl.close();
462 });
463
464 rl.on("SIGPOLLNORM", () => {
465     rl.close();
466 });
467
468 rl.on("SIGPOLLERR", () => {
469     rl.close();
470 });
471
472 rl.on("SIGPOLLHUP", () => {
473     rl.close();
474 });
475
476 rl.on("SIGPOLLNORM", () => {
477     rl.close();
478 });
479
480 rl.on("SIGPOLLERR", () => {
481     rl.close();
482 });
483
484 rl.on("SIGPOLLHUP", () => {
485     rl.close();
486 });
487
488 rl.on("SIGPOLLNORM", () => {
489     rl.close();
490 });
491
492 rl.on("SIGPOLLERR", () => {
493     rl.close();
494 });
495
496 rl.on("SIGPOLLHUP", () => {
497     rl.close();
498 });
499
500 rl.on("SIGPOLLNORM", () => {
501     rl.close();
502 });
503
504 rl.on("SIGPOLLERR", () => {
505     rl.close();
506 });
507
508 rl.on("SIGPOLLHUP", () => {
509     rl.close();
510 });
511
512 rl.on("SIGPOLLNORM", () => {
513     rl.close();
514 });
515
516 rl.on("SIGPOLLERR", () => {
517     rl.close();
518 });
519
520 rl.on("SIGPOLLHUP", () => {
521     rl.close();
522 });
523
524 rl.on("SIGPOLLNORM", () => {
525     rl.close();
526 });
527
528 rl.on("SIGPOLLERR", () => {
529     rl.close();
530 });
531
532 rl.on("SIGPOLLHUP", () => {
533     rl.close();
534 });
535
536 rl.on("SIGPOLLNORM", () => {
537     rl.close();
538 });
539
540 rl.on("SIGPOLLERR", () => {
541     rl.close();
542 });
543
544 rl.on("SIGPOLLHUP", () => {
545     rl.close();
546 });
547
548 rl.on("SIGPOLLNORM", () => {
549     rl.close();
550 });
551
552 rl.on("SIGPOLLERR", () => {
553     rl.close();
554 });
555
556 rl.on("SIGPOLLHUP", () => {
557     rl.close();
558 });
559
560 rl.on("SIGPOLLNORM", () => {
561     rl.close();
562 });
563
564 rl.on("SIGPOLLERR", () => {
565     rl.close();
566 });
567
568 rl.on("SIGPOLLHUP", () => {
569     rl.close();
570 });
571
572 rl.on("SIGPOLLNORM", () => {
573     rl.close();
574 });
575
576 rl.on("SIGPOLLERR", () => {
577     rl.close();
578 });
579
580 rl.on("SIGPOLLHUP", () => {
581     rl.close();
582 });
583
584 rl.on("SIGPOLLNORM", () => {
585     rl.close();
586 });
587
588 rl.on("SIGPOLLERR", () => {
589     rl.close();
590 });
591
592 rl.on("SIGPOLLHUP", () => {
593     rl.close();
594 });
595
596 rl.on("SIGPOLLNORM", () => {
597     rl.close();
598 });
599
600 rl.on("SIGPOLLERR", () => {
601     rl.close();
602 });
603
604 rl.on("SIGPOLLHUP", () => {
605     rl.close();
606 });
607
608 rl.on("SIGPOLLNORM", () => {
609     rl.close();
610 });
611
612 rl.on("SIGPOLLERR", () => {
613     rl.close();
614 });
615
616 rl.on("SIGPOLLHUP", () => {
617     rl.close();
618 });
619
620 rl.on("SIGPOLLNORM", () => {
621     rl.close();
622 });
623
624 rl.on("SIGPOLLERR", () => {
625     rl.close();
626 });
627
628 rl.on("SIGPOLLHUP", () => {
629     rl.close();
630 });
631
632 rl.on("SIGPOLLNORM", () => {
633     rl.close();
634 });
635
636 rl.on("SIGPOLLERR", () => {
637     rl.close();
638 });
639
640 rl.on("SIGPOLLHUP", () => {
641     rl.close();
642 });
643
644 rl.on("SIGPOLLNORM", () => {
645     rl.close();
646 });
647
648 rl.on("SIGPOLLERR", () => {
649     rl.close();
650 });
651
652 rl.on("SIGPOLLHUP", () => {
653     rl.close();
654 });
655
656 rl.on("SIGPOLLNORM", () => {
657     rl.close();
658 });
659
660 rl.on("SIGPOLLERR", () => {
661     rl.close();
662 });
663
664 rl.on("SIGPOLLHUP", () => {
665     rl.close();
666 });
667
668 rl.on("SIGPOLLNORM", () => {
669     rl.close();
670 });
671
672 rl.on("SIGPOLLERR", () => {
673     rl.close();
674 });
675
676 rl.on("SIGPOLLHUP", () => {
677     rl.close();
678 });
679
680 rl.on("SIGPOLLNORM", () => {
681     rl.close();
682 });
683
684 rl.on("SIGPOLLERR", () => {
685     rl.close();
686 });
687
688 rl.on("SIGPOLLHUP", () => {
689     rl.close();
690 });
691
692 rl.on("SIGPOLLNORM", () => {
693     rl.close();
694 });
695
696 rl.on("SIGPOLLERR", () => {
697     rl.close();
698 });
699
700 rl.on("SIGPOLLHUP", () => {
701     rl.close();
702 });
703
704 rl.on("SIGPOLLNORM", () => {
705     rl.close();
706 });
707
708 rl.on("SIGPOLLERR", () => {
709     rl.close();
710 });
711
712 rl.on("SIGPOLLHUP", () => {
713     rl.close();
714 });
715
716 rl.on("SIGPOLLNORM", () => {
717     rl.close();
718 });
719
720 rl.on("SIGPOLLERR", () => {
721     rl.close();
722 });
723
724 rl.on("SIGPOLLHUP", () => {
725     rl.close();
726 });
727
728 rl.on("SIGPOLLNORM", () => {
729     rl.close();
730 });
731
732 rl.on("SIGPOLLERR", () => {
733     rl.close();
734 });
735
736 rl.on("SIGPOLLHUP", () => {
737     rl.close();
738 });
739
740 rl.on("SIGPOLLNORM", () => {
741     rl.close();
742 });
743
744 rl.on("SIGPOLLERR", () => {
745     rl.close();
746 });
747
748 rl.on("SIGPOLLHUP", () => {
749     rl.close();
750 });
751
752 rl.on("SIGPOLLNORM", () => {
753     rl.close();
754 });
755
756 rl.on("SIGPOLLERR", () => {
757     rl.close();
758 });
759
760 rl.on("SIGPOLLHUP", () => {
761     rl.close();
762 });
763
764 rl.on("SIGPOLLNORM", () => {
765     rl.close();
766 });
767
768 rl.on("SIGPOLLERR", () => {
769     rl.close();
770 });
771
772 rl.on("SIGPOLLHUP", () => {
773     rl.close();
774 });
775
776 rl.on("SIGPOLLNORM", () => {
777     rl.close();
778 });
779
780 rl.on("SIGPOLLERR", () => {
781     rl.close();
782 });
783
784 rl.on("SIGPOLLHUP", () => {
785     rl.close();
786 });
787
788 rl.on("SIGPOLLNORM", () => {
789     rl.close();
790 });
791
792 rl.on("SIGPOLLERR", () => {
793     rl.close();
794 });
795
796 rl.on("SIGPOLLHUP", () => {
797     rl.close();
798 });
799
800 rl.on("SIGPOLLNORM", () => {
801     rl.close();
802 });
803
804 rl.on("SIGPOLLERR", () => {
805     rl.close();
806 });
807
808 rl.on("SIGPOLLHUP", () => {
809     rl.close();
810 });
811
812 rl.on("SIGPOLLNORM", () => {
813     rl.close();
814 });
815
816 rl.on("SIGPOLLERR", () => {
817     rl.close();
818 });
819
820 rl.on("SIGPOLLHUP", () => {
821     rl.close();
822 });
823
824 rl.on("SIGPOLLNORM", () => {
825     rl.close();
826 });
827
828 rl.on("SIGPOLLERR", () => {
829     rl.close();
830 });
831
832 rl.on("SIGPOLLHUP", () => {
833     rl.close();
834 });
835
836 rl.on("SIGPOLLNORM", () => {
837     rl.close();
838 });
839
840 rl.on("SIGPOLLERR", () => {
841     rl.close();
842 });
843
844 rl.on("SIGPOLLHUP", () => {
845     rl.close();
846 });
847
848 rl.on("SIGPOLLNORM", () => {
849     rl.close();
850 });
851
852 rl.on("SIGPOLLERR", () => {
853     rl.close();
854 });
855
856 rl.on("SIGPOLLHUP", () => {
857     rl.close();
858 });
859
860 rl.on("SIGPOLLNORM", () => {
861     rl.close();
862 });
863
864 rl.on("SIGPOLLERR", () => {
865     rl.close();
866 });
867
868 rl.on("SIGPOLLHUP", () => {
869     rl.close();
870 });
871
872 rl.on("SIGPOLLNORM", () => {
873     rl.close();
874 });
875
876 rl.on("SIGPOLLERR", () => {
877     rl.close();
878 });
879
880 rl.on("SIGPOLLHUP", () => {
881     rl.close();
882 });
883
884 rl.on("SIGPOLLNORM", () => {
885     rl.close();
886 });
887
888 rl.on("SIGPOLLERR", () => {
889     rl.close();
890 });
891
892 rl.on("SIGPOLLHUP", () => {
893     rl.close();
894 });
895
896 rl.on("SIGPOLLNORM", () => {
897     rl.close();
898 });
899
900 rl.on("SIGPOLLERR", () => {
901     rl.close();
902 });
903
904 rl.on("SIGPOLLHUP", () => {
905     rl.close();
906 });
907
908 rl.on("SIGPOLLNORM", () => {
909     rl.close();
910 });
911
912 rl.on("SIGPOLLERR", () => {
913     rl.close();
914 });
915
916 rl.on("SIGPOLLHUP", () => {
917     rl.close();
918 });
919
920 rl.on("SIGPOLLNORM", () => {
921     rl.close();
922 });
923
924 rl.on("SIGPOLLERR", () => {
925     rl.close();
926 });
927
928 rl.on("SIGPOLLHUP", () => {
929     rl.close();
930 });
931
932 rl.on("SIGPOLLNORM", () => {
933     rl.close();
934 });
935
936 rl.on("SIGPOLLERR", () => {
937     rl.close();
938 });
939
940 rl.on("SIGPOLLHUP", () => {
941     rl.close();
942 });
943
944 rl.on("SIGPOLLNORM", () => {
945     rl.close();
946 });
947
948 rl.on("SIGPOLLERR", () => {
949     rl.close();
950 });
951
952 rl.on("SIGPOLLHUP", () => {
953     rl.close();
954 });
955
956 rl.on("SIGPOLLNORM", () => {
957     rl.close();
958 });
959
960 rl.on("SIGPOLLERR", () => {
961     rl.close();
962 });
963
964 rl.on("SIGPOLLHUP", () => {
965     rl.close();
966 });
967
968 rl.on("SIGPOLLNORM", () => {
969     rl.close();
970 });
971
972 rl.on("SIGPOLLERR", () => {
973     rl.close();
974 });
975
976 rl.on("SIGPOLLHUP", () => {
977     rl.close();
978 });
979
980 rl.on("SIGPOLLNORM", () => {
981     rl.close();
982 });
983
984 rl.on("SIGPOLLERR", () => {
985     rl.close();
986 });
987
988 rl.on("SIGPOLLHUP", () => {
989     rl.close();
990 });
991
992 rl.on("SIGPOLLNORM", () => {
993     rl.close();
994 });
995
996 rl.on("SIGPOLLERR", () => {
997     rl.close();
998 });
999
1000 rl.on("SIGPOLLHUP", () => {
1001     rl.close();
1002 });
1003
1004 rl.on("SIGPOLLNORM", () => {
1005     rl.close();
1006 });
1007
1008 rl.on("SIGPOLLERR", () => {
1009     rl.close();
1010 });
1011
1012 rl.on("SIGPOLLHUP", () => {
1013     rl.close();
1014 });
1015
1016 rl.on("SIGPOLLNORM", () => {
1017     rl.close();
1018 });
1019
1020 rl.on("SIGPOLLERR", () => {
1021     rl.close();
1022 });
1023
1024 rl.on("SIGPOLLHUP", () => {
1025     rl.close();
1026 });
1027
1028 rl.on("SIGPOLLNORM", () => {
1029     rl.close();
1030 });
1031
1032 rl.on("SIGPOLLERR", () => {
1033     rl.close();
1034 });
1035
1036 rl.on("SIGPOLLHUP", () => {
1037     rl.close();
1038 });
1039
1040 rl.on("SIGPOLLNORM", () => {
1041     rl.close();
1042 });
1043
1044 rl.on("SIGPOLLERR", () => {
1045     rl.close();
1046 });
1047
1048 rl.on("SIGPOLLHUP", () => {
1049     rl.close();
1050 });
1051
1052 rl.on("SIGPOLLNORM", () => {
1053     rl.close();
1054 });
1055
1056 rl.on("SIGPOLLERR", () => {
1057     rl.close();
1058 });
1059
1060 rl.on("SIGPOLLHUP", () => {
1061     rl.close();
1062 });
1063
1064 rl.on("SIGPOLLNORM", () => {
1065     rl.close();
1066 });
1067
1068 rl.on("SIGPOLLERR", () => {
1069     rl.close();
1070 });
1071
1072 rl.on("SIGPOLLHUP", () => {
1073     rl.close();
1074 });
1075
1076 rl.on("SIGPOLLNORM", () => {
1077     rl.close();
1078 });
1079
1080 rl.on("SIGPOLLERR", () => {
1081     rl.close();
1082 });
1083
1084 rl.on("SIGPOLLHUP", () => {
1085     rl.close();
1086 });
1087
1088 rl.on("SIGPOLLNORM", () => {
1089     rl.close();
1090 });
1091
1092 rl.on("SIGPOLLERR", () => {
1093     rl.close();
1094 });
1095
1096 rl.on("SIGPOLLHUP", () => {
1097     rl.close();
1098 });
1099
1100 rl.on("SIGPOLLNORM", () => {
1101     rl.close();
1102 });
1103
1104 rl.on("SIGPOLLERR", () => {
1105     rl.close();
1106 });
1107
1108 rl.on("SIGPOLLHUP", () => {
1109     rl.close();
1110 });
1111
1112 rl.on("SIGPOLLNORM", () => {
1113     rl.close();
1114 });
1115
1116 rl.on("SIGPOLLERR", () => {
1117     rl.close();
1118 });
1119
1120 rl.on("SIGPOLLHUP", () => {
1121     rl.close();
1122 });
1123
1124 rl.on("SIGPOLLNORM", () => {
1125     rl.close();
1126 });
1127
1128 rl.on("SIGPOLLERR", () => {
1129     rl.close();
1130 });
1131
1132 rl.on("SIGPOLLHUP", () => {
1133     rl.close();
1134 });
1135
1136 rl.on("SIGPOLLNORM", () => {
1137     rl.close();
1138 });
1139
1140 rl.on("SIGPOLLERR", () => {
1141     rl.close();
1142 });
1143
1144 rl.on("SIGPOLLHUP", () => {
1145     rl.close();
1146 });
1147
1148 rl.on("SIGPOLLNORM", () => {
1149     rl.close();
1150 });
1151
1152 rl.on("SIGPOLLERR", () => {
1153     rl.close();
1154 });
1155
1156 rl.on("SIGPOLLHUP", () => {
1157     rl.close();
1158 });
1159
1160 rl.on("SIGPOLLNORM", () => {
1161     rl.close();
1162 });
1163
1164 rl.on("SIGPOLLERR", () => {
1165     rl.close();
1166 });
1167
1168 rl.on("SIGPOLLHUP", () => {
1169     rl.close();
1170 });
1171
1172 rl.on("SIGPOLLNORM", () => {
1173     rl.close();
1174 });
1175
1176 rl.on("SIGPOLLERR", () => {
1177     rl.close();
1178 });
1179
1180 rl.on("SIGPOLLHUP", () => {
1181     rl.close();
1182 });
1183
1184 rl.on("SIGPOLLNORM", () => {
1185     rl.close();
1186 });
1187
1188 rl.on("SIGPOLLERR", () => {
1189     rl.close();
1190 });
1191
1192 rl.on("SIGPOLLHUP", () => {
1193     rl.close();
1194 });
1195
1196 rl.on("SIGPOLLNORM", () => {
1197     rl.close();
1198 });
1199
1200 rl.on("SIGPOLLERR", () => {
1201     rl.close();
1202 });
1203
1204 rl.on("SIGPOLLHUP", () => {
1205     rl.close();
1206 });
1207
1208 rl.on("SIGPOLLNORM", () => {
1209     rl.close();
1210 });
1211
1212 rl.on("SIGPOLLERR", () => {
1213     rl.close();
1214 });
1215
1216 rl.on("SIGPOLLHUP", () => {
1217     rl.close();
1218 });
1219
1220 rl.on("SIGPOLLNORM", () => {
1221     rl.close();
1222 });
1223
1224 rl.on("SIGPOLLERR", () => {
1225     rl.close();
1226 });
1227
1228 rl.on("SIGPOLLHUP", () => {
1229     rl.close();
1230 });
1231
1232 rl.on("SIGPOLLNORM", () => {
1233     rl.close();
1234 });
1235
1236 rl.on("SIGPOLLERR", () => {
1237     rl.close();
1238 });
1239
1240 rl.on("SIGPOLLHUP", () => {
1241     rl.close();
1242 });
1243
1244 rl.on("SIGPOLLNORM", () => {
1245     rl.close();
1246 });
1247
1248 rl.on("SIGPOLLERR", () => {
1249     rl.close();
1250 });
1251
1252 rl.on("SIGPOLLHUP", () => {
1253     rl.close();
1254 });
1255
1256 rl.on("SIGPOLLNORM", () => {
1257     rl.close();
1258 });
1259
1260 rl.on("SIGPOLLERR", () => {
1261     rl.close();
1262 });
1263
1264 rl.on("SIGPOLLHUP", () => {
1265     rl.close();
1266 });
1267
1268 rl.on("SIGPOLLNORM", () => {
1269     rl.close();
1270 });
1271
1272 rl.on("SIGPOLLERR", () => {
1273     rl.close();
1274 });
1275
1276 rl.on("SIGPOLLHUP", () => {
1277     rl.close();
1278 });
1279
1280 rl.on("SIGPOLLNORM", () => {
1281     rl.close();
1282 });
1283
1284 rl.on("SIGPOLLERR", () => {
1285     rl.close();
1286 });
1287
1288 rl.on("SIGPOLLHUP", () => {
1289     rl.close();
1290 });
1291
1292 rl.on("SIGPOLLNORM", () => {
1293     rl.close();
1294 });
1295
1296 rl.on("SIGPOLLERR", () => {
1297     rl.close();
1298 });
1299
1300 rl.on("SIGPOLLHUP", () => {
1301     rl.close();
1302 });
1303
1304 rl.on("SIGPOLLNORM", () => {
1305     rl.close();
1306 });
1307
1308 rl.on("SIGPOLLERR", () => {
1309     rl.close();
1310 });
1311
1312 rl.on("SIGPOLLHUP", () => {
1313     rl.close();
1314 });
1315
1316 rl.on("SIGPOLLNORM", () => {
1317     rl.close();
1318 });
1319
1320 rl.on("SIGPOLLERR", () => {
1321     rl.close();
1322 });
1323
1324 rl.on("SIGPOLLHUP", () => {
1325     rl.close();
1326 });
1327
1328 rl.on("SIGPOLLNORM", () => {
1329     rl.close();
1330 });
1331
1332 rl.on("SIGPOLLERR", () => {
1333     rl.close();
1334 });
1335
1336 rl.on("SIGPOLLHUP", () => {
1337     rl.close();
1338 });
1339
1340 rl.on("SIGPOLLNORM", () => {
1341     rl.close();
1342 });
1343
1344 rl.on("SIGPOLLERR", () => {
1345     rl.close();
1346 });
1347
1348 rl.on("SIGPOLLHUP", () => {
1349     rl.close();
1350 });
1351
1352 rl.on("SIGPOLLNORM", () => {
1353     rl.close();
1354 });
1355
1356 rl.on("SIGPOLLERR", () => {
1357     rl.close();
1358 });
1359
1360 rl.on("SIGPOLLHUP", () => {
1361     rl.close();
1362 });
1363
1364 rl.on("SIGPOLLNORM", () => {
1365     rl.close();
1366 });
1367
1368 rl.on("SIGPOLLERR", () => {
1369     rl.close();
1370 });
1371
1372 rl.on("SIGPOLLHUP", () => {
1373     rl.close();
1374 });
1375
1376 rl.on("SIGPOLLNORM", () => {
1377     rl.close();
1378 });
1379
1380 rl.on("SIGPOLLERR", () => {
1381     rl.close();
1382 });
1383
1384 rl.on("SIGPOLLHUP", () =&gt
```

```
46     }
47
48     // 计算另外两个可能的对角点
49     const p5 = `${x1 + (y1 - y2)},{${y1 - (x1 - x2)}}`;
50     const p6 = `${x2 + (y1 - y2)},{${y2 - (x1 - x2)}}`;
51
52     if (points.has(p5) && points.has(p6)) {
53         squareCount++;
54     }
55 }
56
57 // 每个正方形被计算了4次，因此结果需要除以4
58 console.log(Math.floor(squareCount / 4));
59
60 };
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 解析输入
12 func parseInput() ([][2]int, map[[2]int]bool) {
13     scanner := bufio.NewScanner(os.Stdin)
14     scanner.Scan()
15     n, _ := strconv.Atoi(scanner.Text())
16
17     points := make(map[[2]int]bool)
18     pointList := make([][2]int, n)
19
20     for i := 0; i < n; i++ {
21         scanner.Scan()
22         tokens := strings.Fields(scanner.Text())
23         x, _ := strconv.Atoi(tokens[0])
24         y, _ := strconv.Atoi(tokens[1])
25
26         p := [2]int{x, y}
27         points[p] = true
28         pointList[i] = p
29     }
30     return pointList, points
31 }
32
33 func main() {
34     pointList, points := parseInput()
35     n := len(pointList)
36     squareCount := 0
37
38     // 遍历所有点对，检查是否能构成正方形
39     for i := 0; i < n; i++ {
40         x1, y1 := pointList[i][0], pointList[i][1]
41
42         for j := i + 1; j < n; j++ {
43             x2, y2 := pointList[j][0], pointList[j][1]
44
45             p3 := [2]int{x1 - (y1 - y2), y1 + (x1 - x2)}
```

```
46     p4 := [2]int{x2 - (y1 - y2), y2 + (x1 - x2)}
47
48     if points[p3] && points[p4] {
49         squareCount++
50     }
51
52     p5 := [2]int{x1 + (y1 - y2), y1 - (x1 - x2)}
53     p6 := [2]int{x2 + (y1 - y2), y2 - (x1 - x2)}
54
55     if points[p5] && points[p6] {
56         squareCount++
57     }
58 }
59 }
60 }
61     fmt.Println(squareCount / 4)
62 }
```

来自: 华为OD机试 2025C卷 – 构成正方形的数量 (C++ & Python & JAVA & JS & GO)_华为od机试
2025a卷 – 构成正方形的数量-CSDN博客