

od0603

[华为OD机考2025B卷 - 组装最大可靠性设备 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 数字游戏 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 信道分配 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 上班之路/是否能到达公司 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 完美走位 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 异常的打卡记录 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 高矮个子排队 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 分班问题/幼儿园分班 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 模拟消息队列 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

[华为OD机考2025B卷 - 最小循环子数组 \(Java & Python& JS & C++ & C\) -CSDN博客](#)

华为OD机考2025B卷 - 组装最大可靠性设备 (Java & Python& JS & C++ & C) -CSDN博客

最新华为OD机试

真题目录: [点击查看目录](#)

华为OD面试真题精选: [点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

一个设备由N种类型元器件组成(每种类型元器件只需要一个, 类型type编号从0~N-1), 每个元器件均有可靠性属性reliability, 可靠性越高的器件其价格price越贵。

而设备的可靠性由组成设备的所有器件中可靠性最低的器件决定。

给定预算S, 购买N种元器件(每种类型元器件都需要购买一个), 在不超过预算的情况下, 请给出能够组成的设备的最大可靠性。

输入描述

S N, 其中 S 为总的预算, N 元器件的种类

total, 表示元器件的总数, 每种型号的元器件可以有多种

此后输入有 total 行具体器件的数据

type reliability price。其中 type 为整数类型, 代表元器件的类型编号从 0 ~ N-1; reliability 整数类型, 代表元器件的可靠性; price 整数类型, 代表元器件的价格

备注

- $0 \leq S, price \leq 10000000$
- $0 \leq N \leq 100$
- $0 \leq type \leq N-1$
- $0 \leq total \leq 100000$
- $0 < reliability \leq 100000$

输出描述

符合预算的设备的最大可靠性, 如果预算无法买齐N种器件, 则返回 -1

示例1

输入

▼	Plain Text		
1	500	3	
2	6		
3	0	80	100
4	0	90	200
5	1	50	50
6	1	70	210
7	2	50	100
8	2	60	150

输出

▼	Plain Text		
1	60		

说明

预算500，设备需要3种元件组成，方案
类型0的第一个(可靠性80)，
类型1的第二个(可靠性70)，
类型2的第二个(可靠性60)，
可以使设备的可靠性最大 60

示例2

输入

▼	Plain Text		
1	100	1	
2	1		
3	0	90	200

输出

▼	Plain Text		
1	-1		

说明

组成设备需要1个元件，但是元件价格大于预算，因此无法组成设备，返回-1

解题思路：

1. 分类处理元器件：

- 对输入的元器件按照类型分类，将每种类型的元器件分组，这样可以针对每个类型单独选择最合适的元器件。

2. 动态规划求解：

- 可以使用动态规划来解决这个问题。设 $dp[i][j]$ 表示在考虑到第 i 种元器件时，预算不超过 j 时的最大可靠性。
- 初始状态 $dp[0][...]$ 表示只选择第0类元器件时的情况。
- 对于每种类型，遍历该类型所有可选的元器件，更新状态，使得在不超过预算的情况下，最大化可靠性。

3. 选择合适的元器件：

- 对每个类型的元器件，从中挑选可靠性高且价格在预算范围内的元器件。
- 在更新状态时，设备的可靠性取决于选出的所有元器件中最低的可靠性，因此在选择元器件时，要尽量保证最终的最小可靠性最大化。

4. 边界情况：

- 如果预算不足以购买每种类型的元器件，则返回 -1 。

```

1  import java.util.*;
2
3  public class Main {
4      public static int maxReliability(int S, int N, int[][] components) {
5          // 初始化每种类型的元器件
6          List<int[]>[] types = new ArrayList[N];
7          for (int i = 0; i < N; i++) {
8              types[i] = new ArrayList<>();
9          }
10
11         // 将每个元器件根据其类型进行分类
12         for (int[] component : components) {
13             int t = component[0]; // 设备类型
14             int r = component[1]; // 设备可靠性
15             int p = component[2]; // 设备价格
16             types[t].add(new int[]{r, p});
17         }
18
19         // 初始化 dp 数组, dp[i][j] 表示选择前i种类型的元器件, 预算为j时的最大可靠性
20         int[][] dp = new int[N + 1][S + 1];
21         for (int[] row : dp) {
22             Arrays.fill(row, -1);
23         }
24         dp[0][0] = Integer.MAX_VALUE; // 初始化为无穷大, 表示没有选择任何元器件
25         // 时可靠性无穷大
26
27         // 对每种类型的元器件进行处理
28         for (int i = 1; i <= N; i++) {
29             // 遍历每种类型的所有元器件
30             for (int[] component : types[i - 1]) {
31                 int r = component[0]; // 设备可靠性
32                 int p = component[1]; // 设备价格
33                 // 从后向前更新dp数组, 确保每个元器件只使用一次
34                 for (int budget = S; budget >= p; budget--) {
35                     if (dp[i - 1][budget - p] != -1) {
36                         dp[i][budget] = Math.max(dp[i][budget], Math.min(d
37                             p[i - 1][budget - p], r));
38                     }
39                 }
40             }
41         }
42
43         // 找到预算范围内的最大可靠性
44         int result = Arrays.stream(dp[N]).max().getAsInt();

```

```

43         return result == -1 ? -1 : result;
44     }
45
46     public static void main(String[] args) {
47         Scanner sc = new Scanner(System.in);
48
49         // 从输入获取预算和元器件信息
50         int S = sc.nextInt(); // 预算
51         int N = sc.nextInt(); // 元器件类型数量
52         int total = sc.nextInt(); // 总元器件数目
53
54         int[][] components = new int[total][3]; // 存储每个元器件的类型、可靠性和
55         // 价格
56         for (int i = 0; i < total; i++) {
57             components[i][0] = sc.nextInt(); // 元器件类型
58             components[i][1] = sc.nextInt(); // 元器件可靠性
59             components[i][2] = sc.nextInt(); // 元器件价格
60         }
61
62         // 计算并输出最大可靠性
63         System.out.println(maxReliability(S, N, components));
64     }

```

Python

```

1  def max_reliability(S, N, components):
2      # 初始化每种类型的元器件
3      types = [[] for _ in range(N)]
4
5      # 将每个元器件根据其类型进行分类
6      for t, r, p in components:
7          types[t].append((r, p))
8
9      # 初始化 dp 数组, dp[i][j] 表示选择前i种类型的元器件, 预算为j时的最大可靠性
10     dp = [[-1] * (S + 1) for _ in range(N + 1)]
11     dp[0][0] = float('inf') # dp[0][0] 初始化为无穷大, 表示没有选择任何元器件时
    可靠性无穷大
12
13     # 对每种类型的元器件进行处理
14     for i in range(1, N + 1):
15         # 遍历每种类型的所有元器件
16         for r, p in types[i - 1]:
17             # 从后向前更新dp数组, 确保每个元器件只使用一次
18             for budget in range(S, p - 1, -1):
19                 if dp[i - 1][budget - p] != -1:
20                     dp[i][budget] = max(dp[i][budget], min(dp[i - 1][budget - p], r))
21
22     # 找到预算范围内的最大可靠性
23     result = max(dp[N])
24
25     # 如果结果仍然是-1, 表示无法满足条件, 返回-1
26     return result if result != -1 else -1
27
28 # 从输入获取预算和元器件信息
29 S, N = map(int, input().split())
30 total = int(input())
31
32 components = []
33
34 for _ in range(total):
35     t, r, p = map(int, input().split())
36     components.append((t, r, p))
37
38 # 计算并输出最大可靠性
39 print(max_reliability(S, N, components))

```

JavaScript

```

1  function maxReliability(S, N, components) {
2      // 初始化每种类型的元器件
3      let types = Array.from({ length: N }, () => []);
4
5      // 将每个元器件根据其类型进行分类
6      for (let [t, r, p] of components) {
7          types[t].push([r, p]);
8      }
9
10     // 初始化 dp 数组, dp[i][j] 表示选择前i种类型的元器件, 预算为j时的最大可靠性
11     let dp = Array.from({ length: N + 1 }, () => Array(S + 1).fill(-1));
12     dp[0][0] = Infinity; // dp[0][0] 初始化为无穷大
13
14     // 对每种类型的元器件进行处理
15     for (let i = 1; i <= N; i++) {
16         // 遍历每种类型的所有元器件
17         for (let [r, p] of types[i - 1]) {
18             // 从后向前更新dp数组
19             for (let budget = S; budget >= p; budget--) {
20                 if (dp[i - 1][budget - p] !== -1) {
21                     dp[i][budget] = Math.max(dp[i][budget], Math.min(dp[i
- 1][budget - p], r));
22                 }
23             }
24         }
25     }
26
27     // 找到预算范围内的最大可靠性
28     let result = Math.max(...dp[N]);
29     return result === -1 ? -1 : result;
30 }
31
32 // 从输入获取预算和元器件信息
33 const readline = require('readline');
34 const rl = readline.createInterface({
35     input: process.stdin,
36     output: process.stdout
37 });
38
39 rl.on('line', (input) => {
40     const [S, N] = input.split(' ').map(Number);
41     rl.on('line', (totalInput) => {
42         const total = Number(totalInput);
43         let components = [];

```



```
44         let count = 0;
45
46         rl.on('line', (line) => {
47             components.push(line.split(' ').map(Number));
48             count++;
49             if (count === total) {
50                 console.log(maxReliability(S, N, components));
51                 rl.close();
52             }
53         });
54     });
55 }
```

C++

```

1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <climits>
5  using namespace std;
6
7  int maxReliability(int S, int N, vector<vector<int>>& components) {
8      // 初始化每种类型的元器件
9      vector<vector<pair<int, int>>> types(N);
10     for (const auto& comp : components) {
11         int t = comp[0], r = comp[1], p = comp[2];
12         types[t].emplace_back(r, p);
13     }
14
15     // 初始化 dp 数组, dp[i][j] 表示选择前i种类型的元器件, 预算为j时的最大可靠性
16     vector<vector<int>> dp(N + 1, vector<int>(S + 1, -1));
17     dp[0][0] = INT_MAX; // dp[0][0] 初始化为无穷大
18
19     // 对每种类型的元器件进行处理
20     for (int i = 1; i <= N; ++i) {
21         // 遍历每种类型的所有元器件
22         for (const auto& [r, p] : types[i - 1]) {
23             // 从后向前更新dp数组
24             for (int budget = S; budget >= p; --budget) {
25                 if (dp[i - 1][budget - p] != -1) {
26                     dp[i][budget] = max(dp[i][budget], min(dp[i - 1][budget
27 t - p], r));
28                 }
29             }
30         }
31
32         // 找到预算范围内的最大可靠性
33         return *max_element(dp[N].begin(), dp[N].end());
34     }
35
36     int main() {
37         int S, N, total;
38         cin >> S >> N >> total;
39
40         // 从输入获取元器件信息
41         vector<vector<int>> components(total, vector<int>(3));
42         for (int i = 0; i < total; ++i) {
43             cin >> components[i][0] >> components[i][1] >> components[i][2];

```

```
44     }  
45  
46     // 计算并输出最大可靠性  
47     int result = maxReliability(S, N, components);  
48     cout << (result == -1 ? -1 : result) << endl;  
49     return 0;  
50 }
```

C语言

```
1  #include <stdio.h>
2  #include <limits.h>
3
4  int max(int a, int b) {
5      return a > b ? a : b;
6  }
7
8  int min(int a, int b) {
9      return a < b ? a : b;
10 }
11
12 int maxReliability(int S, int N, int components[][3], int total) {
13     // 初始化每种类型的元器件
14     int types[N][total][2]; // 最大支持的元器件数是 total
15     int type_count[N];
16     for (int i = 0; i < N; i++) {
17         type_count[i] = 0;
18     }
19
20     // 将每个元器件根据其类型进行分类
21     for (int i = 0; i < total; i++) {
22         int t = components[i][0]; // 元器件类型
23         int r = components[i][1]; // 元器件可靠性
24         int p = components[i][2]; // 元器件价格
25         types[t][type_count[t]][0] = r;
26         types[t][type_count[t]][1] = p;
27         type_count[t]++;
28     }
29
30     // 初始化 dp 数组, dp[i][j] 表示选择前i种类型的元器件, 预算为j时的最大可靠性
31     int dp[N + 1][S + 1];
32     for (int i = 0; i <= N; i++) {
33         for (int j = 0; j <= S; j++) {
34             dp[i][j] = -1;
35         }
36     }
37     dp[0][0] = INT_MAX; // dp[0][0] 初始化为无穷大
38
39     // 对每种类型的元器件进行处理
40     for (int i = 1; i <= N; i++) {
41         // 遍历每种类型的所有元器件
42         for (int k = 0; k < type_count[i - 1]; k++) {
43             int r = types[i - 1][k][0];
44             int p = types[i - 1][k][1];
```

```

45         // 从后向前更新dp数组
46         for (int budget = S; budget >= p; budget--) {
47             if (dp[i - 1][budget - p] != -1) {
48                 dp[i][budget] = max(dp[i][budget], min(dp[i - 1][budget
49             t - p], r));
50             }
51         }
52     }
53
54     // 找到预算范围内的最大可靠性
55     int result = -1;
56     for (int i = 0; i <= S; i++) {
57         if (dp[N][i] > result) {
58             result = dp[N][i];
59         }
60     }
61     return result;
62 }
63
64 int main() {
65     int S, N, total;
66     scanf("%d %d %d", &S, &N, &total);
67
68     // 从输入获取元器件信息
69     int components[total][3];
70     for (int i = 0; i < total; i++) {
71         scanf("%d %d %d", &components[i][0], &components[i][1], &components[i][2]);
72     }
73
74     // 计算并输出最大可靠性
75     int result = maxReliability(S, N, components, total);
76     printf("%d\n", result == -1 ? -1 : result);
77     return 0;
78 }

```

完整用例

用例1

▼	Plain Text		
1	1000	4	
2	8		
3	0 90	300	
4	0 80	200	
5	1 85	250	
6	1 75	200	
7	2 65	150	
8	2 70	170	
9	3 55	130	
10	3 60	160	

用例2

▼	Plain Text		
1	100	1	
2	1		
3	0 90	200	

用例3

▼	Plain Text		
1	300	2	
2	4		
3	0 30	100	
4	0 40	120	
5	1 20	100	
6	1 25	110	

用例4

▼				Plain Text
1	1000	5		
2	10			
3	0 20	90		
4	1 20	80		
5	2 20	70		
6	3 20	60		
7	4 20	50		
8	0 21	95		
9	1 21	85		
10	2 21	75		
11	3 21	65		
12	4 21	55		

用例5

▼				Plain Text
1	50	2		
2	3			
3	0 100	60		
4	1 90	50		
5	1 80	40		

用例6

▼				Plain Text
1	200	2		
2	4			
3	0 30	120		
4	0 35	130		
5	1 25	70		
6	1 28	75		

用例7


```
1  900 3
2  6
3  0 95 310
4  1 85 300
5  2 75 290
6  0 65 280
7  1 55 270
8  2 45 260
```

文章目录

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

机考真题 华为OD



CSDN @算法大师

来自: [华为OD机考2025B卷 – 组装最大可靠性设备 \(Java & Python& JS & C++ & C \)](#) –CSDN博客

华为OD机考2025B卷 - 数字游戏（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

2025[华为od 机试](#)2025B卷-华为机考OD2025年B卷

题目描述

小明玩一个游戏。

系统发1+n张牌，每张牌上有一个整数。

第一张给小明，后n张按照发牌顺序排成连续的一行。

需要小明判断，后n张牌中，是否存在连续的若干张牌，其和可以整除小明手中牌上的数字。

输入描述

输入数据有多组，每组输入数据有两行，输入到文件结尾结束。

第一行有两个整数n和m，空格隔开。m代表发给小明牌上的数字。

第二行有n个数，代表后续发的n张牌上的数字，以空格隔开。

备注

- $1 \leq n \leq 1000$
- $1 \leq \text{牌上的整数} \leq 400000$
- 输入的组数，不多于1000
- 用例确保输入都正确，不需要考虑非法情况。

输出描述

对每组输入，如果存在满足条件的连续若干张牌，则输出1;否则，输出0

示例1

输入

▼ Plain Text |

```
1 6 7
2 2 12 6 3 5 5
3 10 11
4 1 1 1 1 1 1 1 1 1 1
```

输出

▼ Plain Text |

```
1 1
2 0
```

说明

第一组小明牌的数字为7，再发了6张牌。第1、2两张牌数字和为14，可以整除7，输出1，第二组小明牌的数字为11，再发了10张牌，这10张牌数字和为10，无法整除11，输出0。

解题思路

题目描述可以理解为，小明玩一个游戏，游戏中系统会发1+n张牌，其中第一张牌给小明，后续的n张牌排成一行。小明需要判断在这n张牌中，是否存在连续的若干张牌，其数字之和可以被小明手中牌上的数字整除。

具体理解：

- 系统发1+n张牌，第一张牌上的数字是小明手中的牌上的数字，称为 `m`。
- 剩下的n张牌按照发牌顺序排成一行，并且每张牌上也有一个整数。
- 任务是要找到这些n张牌中的连续若干张牌，使得它们的数字之和能够被小明手中牌上的数字 `m` 整除。

示例解释：

- 例如，输入 `6 7` 表示小明手中的牌上的数字是7，后续发了6张牌，牌上的数字依次为 `2 12 6 3 5 5`。
 - 其中第1、2两张牌的数字和为14，可以被7整除，因此输出1。
- 第二组输入 `10 11` 表示小明手中的牌上的数字是11，后续发了10张牌，牌上的数字均为1。
 - 这10张牌的任何连续子数组的和都不能被11整除，因此输出0。

总体思路是使用累加和的余数来判断是否存在连续的若干张牌和可以整除m。通过遍历后续发的牌的数字，累加到sum中，并计算当前和的余数。如果之前已经存在相同的余数，说明存在连续的若干张牌和可以整除m，将found标记为true。最后根据found的值输出1或0，表示是否存在满足条件的连续若干张牌。

```
1  import java.util.Scanner;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6
7          while (scanner.hasNextLine()) {
8              // 读取输入的n和m, n代表牌的数量, m代表小明手中牌上的数字
9              String[] input = scanner.nextLine().split(" ");
10             int n = Integer.parseInt(input[0]);
11             int m = Integer.parseInt(input[1]);
12
13             // 读取后续发的n张牌的数字
14             int[] cardNumbers = new int[n];
15             String[] numStrings = scanner.nextLine().split(" ");
16             for (int i = 0; i < n; i++) {
17                 cardNumbers[i] = Integer.parseInt(numStrings[i]);
18             }
19
20
21             boolean[] remainderExists = new boolean[m];
22             remainderExists[0] = true; // 处理初始和为0的情况
23
24             int sum = 0;
25             boolean found = false;
26             for (int cardNumber : cardNumbers) {
27                 sum += cardNumber;
28                 int remainder = sum % m;
29
30                 if (remainderExists[remainder]) {
31                     found = true;
32                     System.out.println(1);
33                     break;
34                 } else {
35                     remainderExists[remainder] = true;
36                 }
37             }
38             if (!found) {
39                 System.out.println(0);
40             }
41         }
42     }
43 }
```

Python

Plain Text

```
1  import sys
2
3  for line in sys.stdin:
4      # 读取输入的n和m, n代表牌的数量, m代表小明手中牌上的数字
5      n, m = map(int, line.split())
6
7      # 读取后续发的n张牌的数字
8      cardNumbers = list(map(int, input().split()))
9
10     # 使用列表来记录余数的出现情况
11     remainderExists = [False] * m
12     remainderExists[0] = True
13
14     sum = 0
15     found = False
16     for cardNumber in cardNumbers:
17         sum += cardNumber # 将当前牌的数字累加到sum中
18         remainder = sum % m # 计算当前和的余数
19         if remainderExists[remainder]: # 如果之前已经存在相同的余数, 说明存在连
续的若干张牌和可以整除m
20             found = True
21             break
22         else:
23             remainderExists[remainder] = True # 将当前余数标记为已存在
24
25     print(1 if found else 0)
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let isFirstLine = true;
9  let n, m, cardNumbers;
10
11  rl.on('line', (line) => {
12    if (isFirstLine) {
13      // 读取输入的n和m, n代表牌的数量, m代表小明手中牌上的数字
14      [n, m] = line.split(' ').map(Number);
15      isFirstLine = false;
16    } else {
17      // 读取后续发的n张牌的数字
18      cardNumbers = line.split(' ').map(Number);
19
20      // 使用数组来记录余数的出现情况
21      let remainderExists = new Array(m).fill(false);
22      remainderExists[0] = true
23
24      let sum = 0;
25      let found = false;
26      for (let i = 0; i < n; i++) {
27        const cardNumber = cardNumbers[i];
28        sum += cardNumber; // 将当前牌的数字累加到sum中
29        const remainder = sum % m; // 计算当前和的余数
30        if (remainderExists[remainder]) { // 如果之前已经存在相同的余数, 说明存在
连续的若干张牌和可以整除m
31          found = true;
32          break;
33        } else {
34          remainderExists[remainder] = true; // 将当前余数标记为已存在
35        }
36      }
37
38      console.log(found ? 1 : 0);
39
40      isFirstLine = true;
41    }
42  });
```

C++


```
1  #include <iostream>
2  #include <sstream>
3  #include <vector>
4
5  int main() {
6      std::string line;
7      while (std::getline(std::cin, line)) {
8          std::istringstream iss(line);
9
10         // 读取输入的n和m, n代表牌的数量, m代表小明手中牌上的数字
11         int n, m;
12         iss >> n >> m;
13
14         // 读取后续发的n张牌的数字
15         std::vector<int> cardNumbers(n);
16         std::getline(std::cin, line);
17         iss.str(line);
18         iss.clear();
19         for (int i = 0; i < n; i++) {
20             iss >> cardNumbers[i];
21         }
22
23         // 使用bool数组来记录余数的出现情况
24         std::vector<bool> remainderExists(m, false);
25         remainderExists[0] = true;
26         int sum = 0;
27         bool found = false;
28         for (int cardNumber : cardNumbers) {
29             sum += cardNumber; // 将当前牌的数字累加到sum中
30             int remainder = sum % m; // 计算当前和的余数
31             if (remainderExists[remainder]) { // 如果之前已经存在相同的余数, 说
明存在连续的若干张牌和可以整除m
32                 found = true;
33                 break;
34             } else {
35                 remainderExists[remainder] = true; // 将当前余数标记为已存在
36             }
37         }
38
39         std::cout << (found ? 1 : 0) << std::endl;
40     }
41
42     return 0;
43 }
```

C语言

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  int main() {
6      int n, m; // 定义两个整数变量n和m, n代表牌的数量, m代表小明手中牌上的数字
7
8      // 使用while循环读取输入, 当输入不是文件结束符EOF时继续执行
9      while (scanf("%d %d", &n, &m) != EOF) {
10         int cardNumbers[n]; // 定义一个数组来存储n张牌的数字
11
12         // 读取n个牌的数字
13         for (int i = 0; i < n; i++) {
14             scanf("%d", &cardNumbers[i]); // 从输入中读取每个牌的数字并存入数组
15         }
16
17         // 定义一个布尔数组用于记录余数的出现情况, 初始值全部为false
18         bool remainderExists[m];
19         for (int i = 0; i < m; i++) {
20             remainderExists[i] = false; // 初始化布尔数组
21         }
22         remainderExists[0] = true;
23
24         int sum = 0; // 用于存储牌数字的累加和
25         bool found = false; // 标记是否找到满足条件的连续牌
26
27         // 遍历每张牌的数字, 计算累加和并求余数
28         for (int i = 0; i < n; i++) {
29             sum += cardNumbers[i]; // 将当前牌的数字累加到sum中
30             int remainder = sum % m; // 计算当前和的余数
31
32             // 如果当前余数为0, 或者之前已经存在相同的余数, 说明存在满足条件的连续牌
33             if (remainder == 0 || remainderExists[remainder]) {
34                 found = true; // 设置found为true表示找到满足条件的连续牌
35                 break; // 跳出循环
36             } else {
37                 remainderExists[remainder] = true; // 将当前余数标记为已存在
38             }
39         }
40
41         // 输出结果, 1表示找到满足条件的连续牌, 0表示没有找到
42         printf("%d\n", found ? 1 : 0);
43     }
44 }
```

```
45     return 0; // 返回0表示程序正常结束
46 }
```

完整用例

用例1

▼ Plain Text |

```
1  6 7
2  2 12 6 3 5 5
3  10 11
4  1 1 1 1 1 1 1 1 1 1
```

用例2

▼ Plain Text |

```
1  1 7
2  7
```

用例3

▼ Plain Text |

```
1  6 7
2  2 12 6 3 5 5
```

用例4

▼ Plain Text |

```
1  10 11
2  1 1 1 1 1 1 1 1 1 1
```

用例5

▼ Plain Text |

```
1  5 10
2  5 5 5 5 5
```

用例6

▼	Plain Text
1 3 6	
2 4 5 6	

用例7

▼	Plain Text
1 4 8	
2 2 4 6 8	

用例8

▼	Plain Text
1 5 9	
2 1 2 3 4 5	

用例9

▼	Plain Text
1 7 14	
2 2 4 6 8 10 12 14	

用例10

▼	Plain Text
1 6 12	
2 2 4 6 8 10 12	

文章目录

- 最新华为OD机试
- 题目描述
- 输入描述
 - 备注

- 输出描述
- 示例1
- 解题思路
 - 具体理解：
 - 示例解释：
- Java
- Python
- JavaScript
- C++
- C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10

机考真题

华为OD



CSDN @算法大师

来自：华为OD机考2025B卷 – 数字游戏（Java & Python& JS & C++ & C）–CSDN博客

华为OD机考2025B卷 - 信道分配（Java & Python& JS & C++ & C） -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)
华为OD面试真题精选：[点击立即查看](#)

题目描述

算法工程师小明面对着这样一个问题，需要将通信用的信道分配给尽量多的用户：

信道的条件及分配规则如下：

- 1. 所有信道都有属性：“阶”。阶为 r 的信道的容量为 2^r 比特；
- 2. 所有用户需要传输的数据量都一样： D 比特；
- 3. 一个用户可以分配多个信道，但每个信道只能分配给一个用户；
- 4. 只有当分配给一个用户的所有信道的容量和 $\geq D$ ，用户才能传输数据；

给出一组信道资源，最多可以为多少用户传输数据？

输入描述

第一行，一个数字 R 。 R 为最大阶数。
 $0 \leq R < 20$
第二行， $R+1$ 个数字，用空格隔开。代表每种信道的数量 N_i 。按照阶的值从小到大排列。
 $0 \leq i \leq R, 0 \leq N_i < 1000$ 。
第三行，一个数字 D 。 D 为单个用户需要传输的数据量。
 $0 < D < 1000000$

输出描述

一个数字,代表最多可以供多少用户传输数据。

用例

输入

▼ Plain Text

```
1 5
2 10 5 0 1 3 2
3 30
```

▼ Plain Text

1 4

解题思路

对于给定的用例，我们可以通过以下步骤模拟计算过程来理解题目的要求和解决方案：

- 1. 理解输入：
 - 最大阶数 $R = 5$ ，意味着信道的阶从0到5，共有6种不同阶的信道。
 - 信道数量 $N = [10, 5, 0, 1, 3, 2]$ ，表示阶0的信道有10个，阶1的信道有5个，阶2的信道有0个，依此类推。
 - 单个用户需要传输的数据量 $D = 30$ 比特。
- 2. 计算每种阶信道的容量：
 - 阶0的信道容量为 $2^0 = 1$ 比特。有10个这样的。
 - 阶1的信道容量为 $2^1 = 2$ 比特。有5个这样的。
 - 阶2的信道容量为 $2^2 = 4$ 比特。有0个这样的。
 - 阶3的信道容量为 $2^3 = 8$ 比特。有1个这样的。
 - 阶4的信道容量为 $2^4 = 16$ 比特。有3个这样的。
 - 阶5的信道容量为 $2^5 = 32$ 比特。有2个这样的。

用例

每个用户需要传输的数据量为30比特。存在多种信道分配方案，例如：

- 方案1：
 - 使用一个32比特信道满足一个用户（32比特）。
 - 再次使用一个32比特信道满足另一个用户（32比特）。
 - 使用两个16比特信道满足一个用户（32比特）。
 - 组合一个16比特信道、一个8比特信道和三个2比特信道满足一个用户（30比特）。
 - 剩余信道（两个2比特信道和十个1比特信道，共14比特）不足以满足另一个用户。
- 方案2：
 - 组合一个16比特信道、一个8比特信道和三个2比特信道满足一个用户（30比特）。
 - 组合一个16比特信道、两个2比特信道和十个1比特信道满足一个用户（30比特）。
 - 使用一个32比特信道满足一个用户（32比特）。
 - 再次使用一个32比特信道满足另一个用户（32比特）。
 - 剩余一个16比特信道（16比特）不足以满足另一个用户。
- 方案3：
 - 组合一个16比特信道、一个8比特信道和三个2比特信道满足一个用户（30比特）。
 - 组合一个16比特信道、两个2比特信道和十个1比特信道满足一个用户（30比特）。
 - 使用一个32比特信道满足一个用户（32比特）。

- 再次使用一个32比特信道满足另一个用户（32比特）。
- 所有信道恰好用完。

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  // 将二进制向量转换为十进制数，使用了一个更直接的方法
7  int binaryToDecimal(vector<int> binary) {
8      int decimal = 0; // 十进制结果
9      int base = 1; // 基数，初始为1，之后每次循环翻倍
10
11     // 遍历二进制向量，从最低位开始转换
12     for (int i = 0; i < binary.size(); i++) {
13         decimal += binary[i] * base; // 累加到十进制结果中
14         base *= 2; // 基数翻倍，对应二进制的位权增加
15     }
16
17     return decimal;
18 }
19
20 // 将十进制数转换为二进制向量
21 vector<int> decimalToBinary(int decimal) {
22     vector<int> binary; // 二进制向量结果
23
24     // 循环直到十进制数为0
25     while (decimal > 0) {
26         binary.emplace_back(decimal % 2); // 将十进制数除以2的余数作为二进制位
27         decimal /= 2; // 十进制数除以2，向下取整
28     }
29
30     return binary;
31 }
32
33 // 根据需求分配通道
34 bool allocateChannels(vector<int> &availableChannels, vector<int> &requirementBits) {
35     // 从最高位向最低位遍历
36     for (int i = availableChannels.size() - 1; i >= 0; i--) {
37         if (availableChannels[i] >= requirementBits[i]) {
38             availableChannels[i] -= requirementBits[i]; // 如果当前位的可用通道数足够，直接分配
39         } else {
40             // 如果当前位的可用通道数不够，需要检查是否能从更高位借用
41             if (binaryToDecimal(vector<int>(availableChannels.begin(), availableChannels.begin() + i + 1)) <
```

```

42         binaryToDecimal(vector<int>(requirementBits.begin(), requireme
ntBits.begin() + i + 1))) {
43             // 如果更高位的通道总数也不够，分配失败
44             for (int j = i + 1; j < availableChannels.size(); j++) {
45                 if (availableChannels[j] > 0) {
46                     availableChannels[j]--; // 从更高位借用一个通道
47                     return true;
48                 }
49             }
50             return false;
51         } else {
52             // 如果更高位的通道总数足够，进行调整
53             availableChannels[i] -= requirementBits[i];
54             if (i > 0) {
55                 availableChannels[i - 1] += availableChannels[i] * 2;
56             }
57             availableChannels[i] = 0;
58         }
59     }
60 }
61
62 return true;
63 }
64
65 int main() {
66     int maxLevel; // 最大级别
67     cin >> maxLevel;
68
69     vector<int> channelCounts(maxLevel + 1); // 存储每个级别的通道数
70     for (int i = 0; i <= maxLevel; i++) {
71         cin >> channelCounts[i]; // 输入每个级别的通道数
72     }
73
74     int dataRequirement; // 数据需求量
75     cin >> dataRequirement;
76
77     vector<int> requirementBits = decimalToBinary(dataRequirement); // 将
需求量转换为二进制向量
78     vector<int> availableChannels(requirementBits.size(), 0); // 初始化可用
通道向量
79     int usersServed = 0; // 已服务的用户数
80
81     // 初始化可用通道向量
82     for (int i = 0; i <= maxLevel; i++) {
83         if (i >= requirementBits.size()) {
84

```

```

85         usersServed += channelCounts[i]; // 如果级别超过需求位数，直接累加
      到已服务的用户数
86     } else {
87         availableChannels[i] = channelCounts[i]; // 设置每个级别的可用通
      道数
88     }
89 }
90
91 // 尝试分配通道直到不再可能
92 while (allocateChannels(availableChannels, requirementBits)) {
93     usersServed++; // 每成功一次，已服务的用户数增加
94 }
95
96 cout << usersServed << endl; // 输出已服务的用户数
97
98

```

Java

```
1  import java.util.Scanner;
2  import java.util.Arrays;
3
4  public class Main {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7
8          // 用户输入最大信道阶数
9          int maxLevel = scanner.nextInt();
10         scanner.nextLine(); // 消耗换行符, 避免影响下一次读取
11         // 用户输入每种信道的数量, 按阶数从小到大
12         String[] inputCounts = scanner.nextLine().split(" ");
13         // 使用流将字符串数组转换为整数数组
14         int[] channelCounts = Arrays.stream(inputCounts).mapToInt(Integer::parseInt).toArray();
15         // 用户输入单个用户需要的数据量
16         int dataNeed = scanner.nextInt();
17
18         // 将用户需求的数据量转换为二进制位数组, 并反转以符合从低阶到高阶的顺序
19         int[] requirementBits = toBitsArray(dataNeed);
20         // 初始化能服务的用户数
21         int usersServed = 0;
22
23         // 对超出需求位长度的高阶信道, 直接累加其数量至能服务的用户数
24         for (int i = requirementBits.length; i <= maxLevel; i++) {
25             usersServed += channelCounts[i];
26         }
27
28         // 初始化当前可用信道数组, 长度与需求位相同
29         int[] currentChannels = Arrays.copyOfRange(channelCounts, 0, requirementBits.length);
30         // 尝试分配信道直到不再能满足任何用户的需求
31         while (allocateChannels(currentChannels, requirementBits)) {
32             // 成功分配则用户数加一
33             usersServed++;
34         }
35
36         // 输出能服务的用户总数
37         System.out.println(usersServed);
38     }
39
40     // 将十进制数转换为二进制位数组的函数
41     private static int[] toBitsArray(int number) {
42         // 将整数转换为二进制字符串
```

```

43     String binaryString = Integer.toBinaryString(number);
44     // 创建与二进制字符串长度相同的数组
45     int[] bits = new int[binaryString.length()];
46     // 填充数组, 字符串末尾的位对应数组的开始
47     for (int i = 0; i < binaryString.length(); i++) {
48         bits[binaryString.length() - 1 - i] = binaryString.charAt(i)
- '0';
49     }
50     return bits;
51 }
52
53 // 定义分配信道的函数
54 private static boolean allocateChannels(int[] channels, int[] requirement) {
55     // 从最高阶开始向下遍历, 尝试分配信道以满足需求
56     for (int i = channels.length - 1; i >= 0; i--) {
57         if (channels[i] >= requirement[i]) {
58             channels[i] -= requirement[i];
59         } else {
60             // 如果当前阶信道数量不足以满足需求
61             if (bitsToNum(Arrays.copyOfRange(channels, 0, i + 1)) < bitsToNum(Arrays.copyOfRange(requirement, 0, i + 1))) {
62                 // 尝试从更高阶借用一个信道
63                 for (int j = i + 1; j < channels.length; j++) {
64                     if (channels[j] > 0) {
65                         channels[j]--;
66                         return true;
67                     }
68                 }
69                 return false;
70             } else {
71                 // 如果当前及更低阶信道总容量足够, 进行分配
72                 channels[i] -= requirement[i];
73                 if (i > 0) {
74                     // 将不足部分的需求通过倍增转移到下一低阶
75                     channels[i - 1] += channels[i] * 2;
76                 }
77                 channels[i] = 0;
78             }
79         }
80     }
81     return true;
82 }
83
84 // 将二进制位数组转换为十进制数的函数
85 private static int bitsToNum(int[] bits) {
86     int number = 0;

```

```
87         // 遍历二进制位数组，计算十进制数值
88         for (int i = 0; i < bits.length; i++) {
89             number += bits[i] * Math.pow(2, i);
90         }
91         return number;
92     }
```

javaScript

```
1  const readline = require('readline');
2  const rl = readline.createInterface({
3    input: process.stdin,
4    output: process.stdout
5  });
6
7  let inputLines = [];
8
9  rl.on("line", (line) => {
10    inputLines.push(line);
11    if(inputLines.length === 3) {
12      rl.close();
13      processInput(inputLines);
14    }
15  });
16
17  function processInput(inputLines) {
18    // 解析输入
19    const maxLevel = parseInt(inputLines[0]);
20    const channelCounts = inputLines[1].split(' ').map(Number);
21    const dataNeed = parseInt(inputLines[2]);
22
23    // 定义将二进制数组转换为十进制数的函数
24    function bitsToNum(bits) {
25      return bits.reduce((acc, val, idx) => acc + val * Math.pow(2, idx),
26      0);
27    }
28
29    // 定义信道分配函数
30    function allocateChannels(channels, requirement) {
31      for (let i = channels.length - 1; i >= 0; i--) {
32        if (channels[i] >= requirement[i]) {
33          channels[i] -= requirement[i];
34        } else {
35          if (bitsToNum(channels.slice(0, i + 1)) < bitsToNum(requirement.sl
36          ice(0, i + 1))) {
37            // 从更高阶尝试借用信道
38            for (let j = i + 1; j < channels.length; j++) {
39              if (channels[j] > 0) {
40                channels[j]--;
41                return true;
42              }
43            }
44          }
45          return false; // 如果无法借用, 则返回false
46        }
47      }
48    }
49  }
```



```

43         } else {
44             // 分配信道
45             channels[i] -= requirement[i];
46             if (i > 0) {
47                 channels[i - 1] += channels[i] * 2;
48             }
49             channels[i] = 0;
50         }
51     }
52 }
53 return true; // 返回true表示成功分配
54 }
55
56 // 将数据需求转换为二进制位数组，并反转以符合从低阶到高阶的顺序
57 let requirementBits = [...dataNeed.toString(2)].reverse().map(Number);
58 let usersServed = 0;
59
60 // 对超出需求位长度的高阶信道，直接累加其数量至能服务的用户数
61 for (let i = requirementBits.length; i <= maxLevel; i++) {
62     usersServed += channelCounts[i] || 0;
63 }
64
65 // 初始化当前可用信道数组，长度与需求位相同
66 let currentChannels = channelCounts.slice(0, requirementBits.length);
67 while (currentChannels.length < requirementBits.length) {
68     currentChannels.push(0); // 补充长度不足的部分为0
69 }
70
71 // 尝试分配信道直到不再能满足任何用户的需求
72 while (allocateChannels(currentChannels, requirementBits)) {
73     usersServed++;
74 }
75
76 // 输出能服务的用户总数
77 console.log(usersServed);
78 }

```

Python

```

1  # 用户输入最大信道阶数
2  max_level = int(input())
3  # 用户输入每种信道的数量，按阶数从小到大
4  channel_counts = list(map(int, input().split()))
5  # 用户输入单个用户需要的数据量
6  data_need = int(input())
7
8  # 定义将二进制位数组转换为十进制数的函数
9  def bits_to_num(bits):
10     # 对二进制位数组进行遍历，每位值乘以其对应的2的幂次求和得到十进制数
11     return sum(val * (2 ** idx) for idx, val in enumerate(bits))
12
13 # 定义分配信道的函数
14 def allocate_channels(channel阿s, requirement):
15     # 从最高阶开始向下遍历，尝试分配信道以满足需求
16     for i in reversed(range(len(channels))):
17         # 如果当前阶信道数量足够，直接减去需求量
18         if channels[i] >= requirement[i]:
19             channels[i] -= requirement[i]
20         else:
21             # 如果当前阶信道数量不足以满足需求，需要判断是否可以通过低阶信道组合满足需求
22             # 首先判断当前及更低阶信道总容量是否小于需求的总容量
23             if bits_to_num(channels[:i + 1]) < bits_to_num(requirement[:i
24 + 1]):
25                 # 如果低阶总容量不足，尝试从更高阶借用一个信道
26                 for j in range(i + 1, len(channels)):
27                     if channels[j] > 0:
28                         channels[j] -= 1
29                         return True
30                 # 如果高阶也无法借用，说明无法满足当前需求，返回False
31                 return False
32             else:
33                 # 如果当前及更低阶信道总容量足够，先尝试分配当前阶，不足部分由更低阶信道通过倍增补足
34                 channels[i] -= requirement[i]
35                 if i > 0:
36                     # 将不足部分的需求通过倍增转移到下一低阶，即需求翻倍
37                     channels[i - 1] += channels[i] * 2
38                     # 将当前阶信道数量清零，因为已尽可能分配
39                     channels[i] = 0
40             # 如果所有需求都能满足，返回True
41             return True

```

```

42 # 将用户需求的数据量转换为二进制位数组，并反转以符合从低阶到高阶的顺序
43 requirement_bits = list(map(int, str(bin(data_need))[2:]))[::-1]
44 users_served = 0 # 初始化能服务的用户数
45 # 对超出需求位长度的高阶信道，直接累加其数量至能服务的用户数
46 for i in range(len(requirement_bits), max_level + 1):
47     users_served += channel_counts[i]
48 # 初始化当前可用信道数组，长度与需求位相同
49 current_channels = channel_counts[:len(requirement_bits)]
50 while len(current_channels) < len(requirement_bits):
51     # 如果当前可用信道数组长度不足，补零以匹配需求位长度
52     current_channels.append(0)
53 # 尝试分配信道直到不再能满足任何用户的需求
54 while allocate_channels(current_channels, requirement_bits):
55     users_served += 1 # 成功分配则用户数加一
56
57 # 输出能服务的用户总数
58 print(users_served)

```

C语言

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * 将十进制数转换为二进制数组
6   * @param decimal 十进制输入
7   * @param size 用于存储生成的二进制数组的大小
8   * @return 返回二进制数组的指针
9   */
10 int* decimalToBinary(int decimal, int* size) {
11     // 动态分配足够大的空间以存储二进制数
12     int* binary = (int*)malloc(32 * sizeof(int));
13     *size = 0; // 初始化大小为0
14
15     // 循环直到十进制数被完全转换
16     while (decimal > 0) {
17         binary[(*size)++] = decimal % 2; // 存储余数, 即当前二进制位
18         decimal /= 2; // 将十进制数除以2进行下一步转换
19     }
20
21     return binary; // 返回二进制数组
22 }
23
24 /**
25 * 将二进制数组转换为十进制数
26 * @param binary 二进制数组指针
27 * @param size 二进制数组的大小
28 * @return 返回转换得到的十进制数
29 */
30 int binaryToDecimal(int* binary, int size) {
31     int decimal = 0; // 初始化十进制结果为0
32     int base = 1; // 初始化基数为1
33
34     // 遍历二进制数组, 从最低位开始转换
35     for (int i = 0; i < size; i++) {
36         decimal += binary[i] * base; // 累加计算十进制结果
37         base *= 2; // 基数翻倍, 准备下一个二进制位的转换
38     }
39
40     return decimal; // 返回十进制结果
41 }
42
43 /**
44 * 根据需求分配通道
```

```

45  * @param availableChannels 可用通道数组
46  * @param requirementBits 需求位数组
47  * @param size 数组的大小
48  * @return 分配成功返回1, 否则返回0
49  */
50  int allocateChannels(int* availableChannels, int* requirementBits, int size) {
51      // 从最高位向最低位遍历
52      for (int i = size - 1; i >= 0; i--) {
53          if (availableChannels[i] >= requirementBits[i]) {
54              // 如果当前位的可用通道数足够, 则直接分配
55              availableChannels[i] -= requirementBits[i];
56          } else {
57              // 计算当前位及以上位的可用通道总数与需求总数
58              int availableDecimal = binaryToDecimal(availableChannels, i
+ 1);
59              int requirementDecimal = binaryToDecimal(requirementBits, i
+ 1);
60              if (availableDecimal < requirementDecimal) {
61                  // 如果不够, 则尝试从更高位借用通道
62                  for (int j = i + 1; j < size; j++) {
63                      if (availableChannels[j] > 0) {
64                          availableChannels[j]--;
65                          return 1; // 成功借用通道
66                      }
67                  }
68                  return 0; // 无法借用通道, 分配失败
69              } else {
70                  // 如果足够, 则进行相应调整
71                  availableChannels[i] -= requirementBits[i];
72                  if (i > 0) {
73                      availableChannels[i - 1] += availableChannels[i] *
2; // 向下一位借用通道
74                  }
75                  availableChannels[i] = 0;
76              }
77          }
78      }
79
80      return 1; // 成功分配通道
81  }
82
83  int main() {
84      int maxLevel; // 存储最大级别
85      scanf("%d", &maxLevel); // 输入最大级别
86
87      // 根据最大级别动态分配存储通道数的数组

```

```

88     int* channelCounts = (int*)malloc((maxLevel + 1) * sizeof(int));
89     for (int i = 0; i <= maxLevel; i++) {
90         scanf("%d", &channelCounts[i]); // 输入每个级别的通道数
91     }
92
93     int dataRequirement; // 存储数据需求量
94     scanf("%d", &dataRequirement); // 输入数据需求量
95
96     int size;
97     // 将数据需求量转换为二进制表示
98     int* requirementBits = decimalToBinary(dataRequirement, &size);
99
100    // 动态分配数组以存储每个级别的可用通道数，初始化为0
101    int* availableChannels = (int*)malloc(size * sizeof(int));
102    for (int i = 0; i < size; i++) {
103        availableChannels[i] = 0;
104    }
105
106    // 根据输入的通道数，更新可用通道数组
107    int usersServed = 0; // 初始化已服务的用户数为0
108    for (int i = 0; i <= maxLevel; i++) {
109        if (i >= size) {
110            // 如果当前级别超过了需求的二进制表示长度，直接累加该级别的通道数到已服务
            // 用户数
111            usersServed += channelCounts[i];
112        } else {
113            // 否则，更新对应二进制位的可用通道数
114            availableChannels[i] = channelCounts[i];
115        }
116    }
117
118    // 循环尝试分配通道直到无法再分配
119    while (allocateChannels(availableChannels, requirementBits, size)) {
120        usersServed++; // 成功分配则用户数增加
121    }
122
123    printf("%d\n", usersServed); // 输出最终可以服务的用户数
124
125    // 释放之前动态分配的内存资源
126    free(channelCounts);
127    free(requirementBits);
128    free(availableChannels);
129
130    return 0; // 程序结束

```

完整用例

用例1

▼	Plain Text
1 3	
2 5 3 2 1	
3 10	

用例2

▼	Plain Text
1 2	
2 0 0 5	
3 2	

用例3

▼	Plain Text
1 3	
2 1 1 1 1	
3 14	

用例4

▼	Plain Text
1 4	
2 2 3 1 0 4	
3 16	

用例5

▼	Plain Text
1 4	
2 4 5 6 7 8	
3 30	

用例6

机考真题 华为OD



CSDN @算法大师

来自: [华为OD机考2025B卷 – 信道分配 \(Java & Python& JS & C++ & C \)](#) –CSDN博客

华为OD机考2025B卷 - 上班之路/是否能到达公司 (Java & Python& JS & C++ & C) -CSDN博客

最新华为OD机试

真题目录: [点击查看目录](#)

华为OD面试真题精选: [点击立即查看](#)

最新华为OD机试





真题目录: [点击查看目录](#)

华为OD面试真题精选: [点击立即查看](#)

题目描述

Jungle 生活在美丽的蓝鲸城, 大马路都是方方正正, 但是每天马路的封闭情况都不一样。

地图由以下元素组成:

-  — 空地, 可以达到;
-  — 路障, 不可达到;
-  — *Jungle*的家;
-  — 公司;

其中我们会限制*Jungle*拐弯的次数, 同时*Jungle*可以清除给定个数的路障, 现在你的任务是计算*Jungle*是否可以从家里出发到达公司。

输入描述

输入的第一行为两个整数 t, c ($0 \leq t, c \leq 100$), t 代表可以拐弯的次数, c 代表可以清除的路障个数。

输入的第二行为两个整数 n, m ($1 \leq n, m \leq 100$), 代表地图的大小。

接下来是 n 行包含 m 个字符的地图。 n 和 m 可能不一样大。

我们保证地图里有S和T。

输出描述

输出是否可以从家里出发到达公司, 是则输出YES, 不能则输出NO。

示例1

输入

▼	Plain Text
1	2 0
2	5 5
3	..S..
4	****.
5	T....
6	****.
7

输出

▼	Plain Text
1	YES

说明

示例2

输入

▼	Plain Text
1	1 2
2	5 5
3	.*S*.
4	*****
5	..*..
6	*****
7	T....

输出

▼	Plain Text
1	NO

说明

该用例中，至少需要拐弯1次，清除3个路障，所以无法到达

解题思路

题目解析

题目描述了一种网格图路径问题，目的是判断从起点 **S** 到目标点 **T** 是否存在一条路径，该路径需要满足以下限制条件：

1. **拐弯次数限制**：路径不能超过给定的最大拐弯次数 **t**。
2. **路障清除限制**：路径最多可以清除给定数量的路障 **c**。

同时，网格中包含以下元素：

- **.**：空地，可以直接通行；
 - *****：路障，可以清除，但受到次数限制；
 - **S**：起点；
 - **T**：目标点。
-

问题细化

1. 路径限制：

- 需要考虑拐弯次数：
 - 拐弯次数是指路径方向的改变，比如从向右转为向下。
- 路障清除次数：
 - 路障清除有次数限制，即最多可以清除 **c** 个 *****。

2. 地图的特点：

- 图是一个二维矩阵。
- 从起点到终点，路径可以向上下左右四个方向移动。

3. 算法实现关键：

- **状态表示**：
 - 路径的状态包括：
 - 当前所在位置；
 - 当前的拐弯次数；
 - 已清除的路障个数；
 - 当前的移动方向。
- **搜索方法**：
 - 采用 **广度优先搜索 (BFS)** 或 **深度优先搜索 (DFS)**，尝试所有可能的路径。
 - 通过记录状态避免重复访问。

4. 边界条件：

- 超出地图范围时终止路径；
 - 如果清除的路障数超过 **c**，或者拐弯次数超过 **t**，路径无效。
-
-

解题思路总结

1. 搜索算法选择：

- 采用 BFS（广度优先搜索），因为 BFS 能够优先找到最短路径，适合处理带约束条件的路径问题。
- DFS 也可以，但需要注意剪枝优化。

2. 状态表示：

- 每个状态需要记录：
 - 当前坐标；
 - 当前方向；
 - 当前已用的拐弯次数；
 - 当前已清除的路障个数。

3. 剪枝优化：

- 如果某个状态的拐弯次数或清除路障个数超过限制，则剪枝。

4. 实现框架：

- 初始化 BFS 队列，将起点状态入队；
- 使用方向数组模拟移动；
- 每次移动更新状态，检查是否达到目标点；
- 输出结果。

```
1  import java.util.Scanner;
2
3  public class Main {
4      // 定义全局变量
5      static int maxTurns, maxBreaks, rows, cols;
6      static char[][] matrix;
7      static int[][] offsets = {{-1, 0, 1}, {1, 0, 2}, {0, -1, 3}, {0, 1,
4}}};
8      static int targetRow, targetCol;
9      static boolean[][] visited;
10
11     public static void main(String[] args) {
12         Scanner scanner = new Scanner(System.in);
13
14         maxTurns = scanner.nextInt();
15         maxBreaks = scanner.nextInt();
16         rows = scanner.nextInt();
17         cols = scanner.nextInt();
18
19         // 初始化地图矩阵
20         matrix = new char[rows][cols];
21         visited = new boolean[rows][cols];
22
23         // 读取地图数据, 找到起点 S 和终点 T
24         int startRow = -1, startCol = -1;
25         for (int i = 0; i < rows; i++) {
26             String line = scanner.next();
27             for (int j = 0; j < cols; j++) {
28                 matrix[i][j] = line.charAt(j);
29                 if (matrix[i][j] == 'S') {
30                     startRow = i;
31                     startCol = j;
32                 } else if (matrix[i][j] == 'T') {
33                     targetRow = i;
34                     targetCol = j;
35                 }
36             }
37         }
38
39         // 调用深度优先搜索方法, 输出结果
40         System.out.println(dfs(startRow, startCol, 0, 0, -1) ? "YES" : "N
0");
41     }
42 }
```

```

43 // 深度优先搜索方法
44 public static boolean dfs(int row, int col, int turnCount, int breakCo
unt, int lastDirect) {
45 // 如果当前位置是终点, 返回 true
46 if (row == targetRow && col == targetCol) {
47     return true;
48 }
49
50 // 将当前位置标记为已访问
51 visited[row][col] = true;
52
53 // 遍历四个方向
54 for (int[] offset : offsets) {
55     int newRow = row + offset[0];
56     int newCol = col + offset[1];
57     int direct = offset[2];
58
59 // 检查新的行和列是否在地图范围内且未访问
60 if (newRow >= 0 && newRow < rows && newCol >= 0 && newCol < co
ls && !visited[newRow][newCol]) {
61     boolean isTurn = (lastDirect != -1 && lastDirect != direc
t); // 是否拐弯
62     boolean isObstacle = (matrix[newRow][newCol] == '*'); //
是否遇到路障
63
64     // 如果拐弯次数或清除路障次数超过限制, 跳过
65     if ((isTurn && turnCount + 1 > maxTurns) || (isObstacle &
& breakCount + 1 > maxBreaks)) continue;
66
67 // 递归调用深度优先搜索方法
68 if (dfs(newRow, newCol, turnCount + (isTurn ? 1 : 0), brea
kCount + (isObstacle ? 1 : 0), direct)) {
69     return true;
70 }
71 }
72 }
73
74 // 回溯: 将当前位置标记为未访问
75 visited[row][col] = false;
76 return false;
77 }
78 }

```

Python

```

1  # 读取输入数据
2  max_turns, max_breaks = map(int, input().split())
3  rows, cols = map(int, input().split())
4
5  # 初始化地图矩阵
6  matrix = [list(input().strip()) for _ in range(rows)]
7  visited = [[False] * cols for _ in range(rows)]
8
9  # 偏移量和方向定义
10 offsets = [(-1, 0, 1), (1, 0, 2), (0, -1, 3), (0, 1, 4)]
11
12 # 找到起点 S 和终点 T 的位置
13 start_row, start_col = -1, -1
14 target_row, target_col = -1, -1
15 for i in range(rows):
16     for j in range(cols):
17         if matrix[i][j] == 'S':
18             start_row, start_col = i, j
19         elif matrix[i][j] == 'T':
20             target_row, target_col = i, j
21
22 # 深度优先搜索方法
23 def dfs(row, col, turn_count, break_count, last_direct):
24     # 如果当前位置是终点, 返回 True
25     if row == target_row and col == target_col:
26         return True
27
28     # 将当前位置标记为已访问
29     visited[row][col] = True
30
31     # 遍历四个方向
32     for offset in offsets:
33         new_row = row + offset[0]
34         new_col = col + offset[1]
35         direct = offset[2]
36
37         # 检查新的行和列是否在地图范围内且未访问
38         if 0 <= new_row < rows and 0 <= new_col < cols and not visited[new
_row][new_col]:
39             is_turn = (last_direct != -1 and last_direct != direct) # 是
否拐弯
40             is_obstacle = (matrix[new_row][new_col] == '*') # 是否遇到路障
41
42             # 如果拐弯次数或清除路障次数超过限制, 跳过

```



```

43         if (is_turn and turn_count + 1 > max_turns) or (is_obstacle and break_count + 1 > max_breaks):
44             continue
45
46         # 递归调用深度优先搜索方法
47         if dfs(new_row, new_col, turn_count + (1 if is_turn else 0), break_count + (1 if is_obstacle else 0), direct):
48             return True
49
50     # 回溯: 将当前位置标记为未访问
51     visited[row][col] = False
52     return False
53
54 # 调用深度优先搜索方法并输出结果
55 if dfs(start_row, start_col, 0, 0, -1):
56     print("YES")
57 else:
58     print("NO")

```

JavaScript

```
1 // 引入 Node.js 内置的 readline 模块, 用于读取标准输入 (用户输入)
2 const readline = require('readline');
3
4 // 创建 readline 接口, 用于从标准输入读取数据
5 const rl = readline.createInterface({
6   input: process.stdin, // 输入流为标准输入 (通常是键盘输入)
7   output: process.stdout // 输出流为标准输出 (通常是屏幕输出)
8 });
9
10 let inputLines = []; // 用于存储所有输入的行
11 let maxTurns, maxBreaks, rows, cols, matrix; // 定义变量: 最大拐弯数、最大破坏
    障碍物数、行数、列数、地图矩阵
12
13 // 当接收到输入行时触发的事件
14 rl.on('line', (line) => {
15   // 将输入的每一行存入数组
16   inputLines.push(line);
17
18   // 如果已读取到前两行, 解析最大拐弯数和破坏障碍物数, 以及地图的行数和列数
19   if (inputLines.length === 2) {
20     [maxTurns, maxBreaks] = inputLines[0].split(" ").map(Number); // 第1
    行: 最大拐弯数和最大破坏数
21     [rows, cols] = inputLines[1].split(" ").map(Number); // 第2行: 地图的行
    数和列数
22   }
23
24   // 当读取完所有地图行数时, 构建地图矩阵
25   if (rows && inputLines.length === rows + 2) {
26     matrix = inputLines.slice(2).map((line) => line.split("")); // 从第3行
    开始读取地图数据, 转换为矩阵
27     rl.close(); // 输入读取完成, 关闭输入流
28   }
29 });
30
31 // 当输入流关闭时触发的事件
32 rl.on('close', () => {
33   // 初始化一个二维数组, 表示地图的访问状态, 初始值为 false (未访问)
34   const visited = Array.from({ length: rows }, () => Array(cols).fill(false));
35
36   // 定义移动四个方向: [行偏移量, 列偏移量, 方向编号]
37   const offsets = [[-1, 0, 1], [1, 0, 2], [0, -1, 3], [0, 1, 4]];
38
39   // 查找起点 ('S') 和终点 ('T') 的坐标
```

```

40     let startRow, startCol, targetRow, targetCol;
41     for (let i = 0; i < rows; i++) {
42         for (let j = 0; j < cols; j++) {
43             if (matrix[i][j] === 'S') {
44                 startRow = i; // 记录起点的行
45                 startCol = j; // 记录起点的列
46             } else if (matrix[i][j] === 'T') {
47                 targetRow = i; // 记录终点的行
48                 targetCol = j; // 记录终点的列
49             }
50         }
51     }
52
53     /**
54      * 深度优先搜索 (DFS) 函数
55      * @param {number} row 当前所在的行
56      * @param {number} col 当前所在的列
57      * @param {number} turnCount 当前的拐弯次数
58      * @param {number} breakCount 当前破坏障碍物的次数
59      * @param {number} lastDirect 上一次的移动方向
60      * @returns {boolean} 是否能成功到达终点
61      */
62     function dfs(row, col, turnCount, breakCount, lastDirect) {
63         // 基础情况: 如果当前到达终点, 则返回 true
64         if (row === targetRow && col === targetCol) {
65             return true;
66         }
67
68         // 将当前位置标记为已访问
69         visited[row][col] = true;
70
71         // 遍历所有可能的方向
72         for (const [dRow, dCol, direct] of offsets) {
73             const newRow = row + dRow; // 新的行
74             const newCol = col + dCol; // 新的列
75
76             // 检查新的位置是否在地图范围内, 且未被访问过
77             if (
78                 newRow >= 0 && newRow < rows &&
79                 newCol >= 0 && newCol < cols &&
80                 !visited[newRow][newCol]
81             ) {
82                 const isTurn = lastDirect !== -1 && lastDirect !== direct; // 判
断是否拐弯
83                 const isObstacle = matrix[newRow][newCol] === '*'; // 判断是否遇到
障碍物
84

```

```

85         // 如果拐弯次数或破坏障碍物次数超过限制，则跳过
86         if ((isTurn && turnCount + 1 > maxTurns) || (isObstacle && breakC
ount + 1 > maxBreaks)) {
87             continue; // 超过限制则跳过此方向
88         }
89
90         // 递归调用 DFS 进行深度搜索
91         if (dfs(
92             newRow, newCol,
93             turnCount + (isTurn ? 1 : 0), // 如果拐弯则增加拐弯次数
94             breakCount + (isObstacle ? 1 : 0), // 如果遇到障碍物则增加破坏次数
95             direct // 更新当前方向
96         )) {
97             return true; // 如果找到路径则返回 true
98         }
99     }
100 }
101
102 // 回溯：将当前位置标记为未访问，以便其他路径尝试
103 visited[row][col] = false;
104 return false; // 未找到路径则返回 false
105 }
106
107 // 调用 DFS 函数从起点开始搜索，输出结果
108 const result = dfs(startRow, startCol, 0, 0, -1) ? "YES" : "NO";
109 console.log(result); // 输出 "YES" 或 "NO"

```

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4
5  using namespace std;
6
7  // 定义全局变量
8  int maxTurns, maxBreaks, rows, cols;
9  vector<vector<char>> matrix;
10 vector<vector<bool>> visited;
11 int offsets[4][3] = {{-1, 0, 1}, {1, 0, 2}, {0, -1, 3}, {0, 1, 4}};
12 int targetRow, targetCol;
13
14 // 深度优先搜索方法
15 bool dfs(int row, int col, int turnCount, int breakCount, int lastDirect)
16 {
17     // 如果当前位置是终点, 返回 true
18     if (row == targetRow && col == targetCol) {
19         return true;
20     }
21     // 将当前位置标记为已访问
22     visited[row][col] = true;
23
24     // 遍历四个方向
25     for (auto& offset : offsets) {
26         int newRow = row + offset[0];
27         int newCol = col + offset[1];
28         int direct = offset[2];
29
30         // 检查新的行和列是否在地图范围内且未访问
31         if (newRow >= 0 && newRow < rows && newCol >= 0 && newCol < cols &
32             & !visited[newRow][newCol]) {
33             bool isTurn = (lastDirect != -1 && lastDirect != direct); //
34             是否拐弯
35             bool isObstacle = (matrix[newRow][newCol] == '*'); // 是否遇到
36             路障
37
38             // 如果拐弯次数或清除路障次数超过限制, 跳过
39             if ((isTurn && turnCount + 1 > maxTurns) || (isObstacle && breakCount + 1 > maxBreaks)) continue;
40
41             // 递归调用深度优先搜索方法
```

```

40         if (dfs(newRow, newCol, turnCount + (isTurn ? 1 : 0), breakCou
41 nt + (isObstacle ? 1 : 0), direct)) {
42             return true;
43         }
44     }
45 }
46
47 // 回溯：将当前位置标记为未访问
48 visited[row][col] = false;
49 return false;
50 }
51
52 int main() {
53     // 读取输入数据
54     cin >> maxTurns >> maxBreaks;
55     cin >> rows >> cols;
56
57     // 初始化地图矩阵
58     matrix.resize(rows, vector<char>(cols));
59     visited.resize(rows, vector<bool>(cols, false));
60
61     // 找到起点 S 和终点 T 的位置
62     int startRow = -1, startCol = -1;
63     for (int i = 0; i < rows; ++i) {
64         string line;
65         cin >> line;
66         for (int j = 0; j < cols; ++j) {
67             matrix[i][j] = line[j];
68             if (matrix[i][j] == 'S') {
69                 startRow = i;
70                 startCol = j;
71             } else if (matrix[i][j] == 'T') {
72                 targetRow = i;
73                 targetCol = j;
74             }
75         }
76     }
77
78     // 调用深度优先搜索方法并输出结果
79     cout << (dfs(startRow, startCol, 0, 0, -1) ? "YES" : "NO") << endl;
80
81     return 0;

```

C语言

```
1  #include <stdio.h>
2  #include <stdbool.h>
3
4  #define MAX_ROWS 100
5  #define MAX_COLS 100
6
7  // 定义全局变量
8  int maxTurns, maxBreaks, rows, cols;
9  char matrix[MAX_ROWS][MAX_COLS];
10 bool visited[MAX_ROWS][MAX_COLS];
11 int offsets[4][3] = {{-1, 0, 1}, {1, 0, 2}, {0, -1, 3}, {0, 1, 4}};
12 int targetRow, targetCol;
13
14 // 深度优先搜索方法
15 bool dfs(int row, int col, int turnCount, int breakCount, int lastDirect)
16 {
17     // 如果当前位置是终点, 返回 true
18     if (row == targetRow && col == targetCol) {
19         return true;
20     }
21     // 将当前位置标记为已访问
22     visited[row][col] = true;
23
24     // 遍历四个方向
25     for (int i = 0; i < 4; ++i) {
26         int newRow = row + offsets[i][0];
27         int newCol = col + offsets[i][1];
28         int direct = offsets[i][2];
29
30         // 检查新的行和列是否在地图范围内且未访问
31         if (newRow >= 0 && newRow < rows && newCol >= 0 && newCol < cols &
32             & !visited[newRow][newCol]) {
33             bool isTurn = (lastDirect != -1 && lastDirect != direct); //
34             // 是否拐弯
35             bool isObstacle = (matrix[newRow][newCol] == '*'); // 是否遇到
36             // 路障
37
38             // 如果拐弯次数或清除路障次数超过限制, 跳过
39             if ((isTurn && turnCount + 1 > maxTurns) || (isObstacle && breakCount + 1 > maxBreaks)) continue;
40
41             // 递归调用深度优先搜索方法
```

```

40         if (dfs(newRow, newCol, turnCount + (isTurn ? 1 : 0), breakCou
41 nt + (isObstacle ? 1 : 0), direct)) {
42             return true;
43         }
44     }
45 }
46
47 // 回溯: 将当前位置标记为未访问
48 visited[row][col] = false;
49 return false;
50 }
51
52 int main() {
53     // 读取输入数据
54     scanf("%d %d", &maxTurns, &maxBreaks);
55     scanf("%d %d", &rows, &cols);
56
57     // 初始化地图矩阵
58     int startRow = -1, startCol = -1;
59     for (int i = 0; i < rows; ++i) {
60         scanf("%s", matrix[i]);
61         for (int j = 0; j < cols; ++j) {
62             if (matrix[i][j] == 'S') {
63                 startRow = i;
64                 startCol = j;
65             } else if (matrix[i][j] == 'T') {
66                 targetRow = i;
67                 targetCol = j;
68             }
69         }
70     }
71
72     // 调用深度优先搜索方法并输出结果
73     if (dfs(startRow, startCol, 0, 0, -1)) {
74         printf("YES\n");
75     } else {
76         printf("NO\n");
77     }
78     return 0;

```


机考真题 华为OD



CSDN @算法大师

来自: [华为OD机考2025B卷 – 上班之路/是否能到达公司 \(Java & Python& JS & C++ & C \)](#) –CSDN 博客

华为OD机考2025B卷 - 完美走位（Java & Python& JS & C++ & C） -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)
华为OD面试真题精选：[点击立即查看](#)

题目描述

在第一人称射击游戏中，玩家通过键盘的A、S、D、W四个按键控制游戏人物分别向左、向后、向右、向前进行移动，从而完成走位。

假设玩家每按动一次键盘，游戏任务会向某个方向移动一步，如果玩家在操作一定次数的键盘并且各个方向的步数相同时，此时游戏任务必定会回到原点，则称此次走位为完美走位。

现给定玩家的走位（例如：ASDA），请通过更换其中一段连续走位的方式使得原走位能够变成一个完美走位。其中待更换的连续走位可以是相同长度的任何走位。

请返回待更换的连续走位的最小可能长度。

如果原走位本身是一个完美走位，则返回0。

输入描述

输入为由键盘字母表示的走位s，例如：ASDA

说明：

- 1、走位长度 $1 \leq s.length \leq 100000$ （也就是长度不一定是偶数）
- 2、s.length 是 4 的倍数
- 3、s中只含有'A'，'S'，'D'，'W' 四种字符

输出描述

输出为待更换的连续走位的最小可能长度。

示例1

输入

▼

Plain Text

1 WASDAASD

输出

▼ Plain Text |

1 1

说明

将第二个A替换为W，即可得到完美走位

示例2

输入

▼ Plain Text |

1 AAAA

输出

▼ Plain Text |

1 3

说明

将其中三个连续的A替换为WSD，即可得到完美走位

解题思路## 分析

完美走位是上下左右的次数一样的。如果不一样，我们需要在给定的走位中，找到一个连续的字串，替换走位，是的走位完美。

题目要求，保持W,A,S,D字母个数平衡，即相等，如果不相等，可以从字符串中选取一段连续子串替换，来让字符串平衡。

比如：WWWAAAASSSS

字符串长度12，W,A,S,D平衡的话，则每个字母个数应该是3个，而现在W,A,S各有4个，也就是说各超了1个。

因此我们应该从字符串中，选取一段包含1个W，1个A，1个S的子串，来替换为D。

WWWAAAASSSS

而符合这种要求的子串可能很多，我们需要找出其中最短的，即 **WAAAAS**。

代码思路

1. 首先，我们需要统计输入字符串中每个字符的出现次数。这可以通过遍历字符串并使用一个哈希表（字典）来实现。

2. 然后，我们需要确定正整数序列的起始值。我们可以通过计算字符串长度除以序列长度来得到每个正整数的平均长度。然后，我们可以从10的（平均长度-1）次方开始，作为起始值。
3. 接下来，我们需要使用滑动窗口的方法来寻找还原后的连续正整数序列。我们可以初始化两个指针，分别表示滑动窗口的左边界和右边界。同时，我们需要一个临时哈希表（字典）来存储当前滑动窗口内的字符出现次数。
4. 在滑动窗口的过程中，我们需要不断更新左右边界的字符出现次数，并检查当前滑动窗口内的字符出现次数是否与输入字符串的字符出现次数匹配。如果匹配，说明我们找到了一个可能的连续正整数序列，此时我们可以更新结果变量。
5. 当右边界超过字符串长度时，滑动窗口的过程结束。此时，结果变量中存储的就是还原后的连续正整数序列中的最小数字。

```
1  import java.util.HashMap;
2  import java.util.Scanner;
3
4  public class Main {
5
6      public static int minReplacementLength(String inputStr) {
7          // 初始化方向键计数字典
8          HashMap<Character, Integer> directionCount = new HashMap<>();
9          directionCount.put('W', 0);
10         directionCount.put('A', 0);
11         directionCount.put('S', 0);
12         directionCount.put('D', 0);
13
14         // 统计输入字符串中每个方向键的出现次数
15         for (char c : inputStr.toCharArray()) {
16             directionCount.put(c, directionCount.get(c) + 1);
17         }
18
19         // 初始化左右指针和结果变量
20         int left = 0;
21         int right = 0;
22         int minLength = inputStr.length();
23
24         // 更新右指针对应的方向键计数
25         directionCount.put(inputStr.charAt(0), directionCount.get(inputStr.charAt(0)) - 1);
26
27         while (true) {
28             // 计算当前最大方向键计数
29             int maxCount = 0;
30             for (int count : directionCount.values()) {
31                 maxCount = Math.max(maxCount, count);
32             }
33
34             // 计算当前窗口长度和可替换的字符数
35             int windowLength = right - left + 1;
36             int replaceableChars = windowLength;
37             for (int count : directionCount.values()) {
38                 replaceableChars -= maxCount - count;
39             }
40
41             // 如果可替换字符数大于等于0且能被4整除, 则更新结果变量
42             if (replaceableChars >= 0 && replaceableChars % 4 == 0) {
43                 minLength = Math.min(minLength, windowLength);
```

```

44
45 // 更新左指针并检查是否越界
46 if (left < inputStr.length()) {
47     directionCount.put(inputStr.charAt(left), directionCou
nt.get(inputStr.charAt(left)) + 1);
48     left++;
49 } else {
50     break;
51 }
52 } else {
53     // 更新右指针并检查是否越界
54     right++;
55     if (right >= inputStr.length()) {
56         break;
57     }
58     directionCount.put(inputStr.charAt(right), directionCount.
get(inputStr.charAt(right)) - 1);
59     }
60 }
61
62     return minLength;
63 }
64
65 public static void main(String[] args) {
66     Scanner scanner = new Scanner(System.in);
67     String inputStr = scanner.nextLine();
68     System.out.println(minReplacementLength(inputStr));
69 }
70 }

```

Python

```
1 def min_replacement_length(input_str):
2     # 初始化方向键计数字典
3     direction_count = {'W': 0, 'A': 0, 'S': 0, 'D': 0}
4
5     # 统计输入字符串中每个方向键的出现次数
6     for char in input_str:
7         direction_count[char] += 1
8
9     # 初始化左右指针和结果变量
10    left = 0
11    right = 0
12    min_length = len(input_str)
13
14    # 更新右指针对应的方向键计数
15    direction_count[input_str[0]] -= 1
16
17    while True:
18        # 计算当前最大方向键计数
19        max_count = max(direction_count.values())
20
21        # 计算当前窗口长度和可替换的字符数
22        window_length = right - left + 1
23        replaceable_chars = window_length - sum(max_count - count for count in direction_count.values())
24
25        # 如果可替换字符数大于等于0且能被4整除, 则更新结果变量
26        if replaceable_chars >= 0 and replaceable_chars % 4 == 0:
27            min_length = min(min_length, window_length)
28
29            # 更新左指针并检查是否越界
30            if left < len(input_str):
31                direction_count[input_str[left]] += 1
32                left += 1
33            else:
34                break
35        else:
36            # 更新右指针并检查是否越界
37            right += 1
38            if right >= len(input_str):
39                break
40            direction_count[input_str[right]] -= 1
41
42    return min_length
43
```

```
44
45  if __name__ == "__main__":
46      input_str = input()
47      print(min_replacement_length(input_str))
```

JavaScript


```
1 function minReplacementLength(inputStr) {
2     // 初始化方向键计数字典
3     const directionCount = { W: 0, A: 0, S: 0, D: 0 };
4
5     // 统计输入字符串中每个方向键的出现次数
6     for (const c of inputStr) {
7         directionCount[c]++;
8     }
9
10    // 初始化左右指针和结果变量
11    let left = 0;
12    let right = 0;
13    let minLength = inputStr.length;
14
15    // 更新右指针对应的方向键计数
16    directionCount[inputStr[0]]--;
17
18    while (true) {
19        // 计算当前最大方向键计数
20        const maxCount = Math.max(...Object.values(directionCount));
21
22        // 计算当前窗口长度和可替换的字符数
23        const windowLength = right - left + 1;
24        let replaceableChars = windowLength;
25        for (const count of Object.values(directionCount)) {
26            replaceableChars -= maxCount - count;
27        }
28
29        // 如果可替换字符数大于等于0且能被4整除，则更新结果变量
30        if (replaceableChars >= 0 && replaceableChars % 4 === 0) {
31            minLength = Math.min(minLength, windowLength);
32
33            // 更新左指针并检查是否越界
34            if (left < inputStr.length) {
35                directionCount[inputStr[left]]++;
36                left++;
37            } else {
38                break;
39            }
40        } else {
41            // 更新右指针并检查是否越界
42            right++;
43            if (right >= inputStr.length) {
44                break;
```

```

45         }
46         directionCount[inputStr[right]]--;
47     }
48 }
49
50     return minLength;
51 }
52
53 const readline = require('readline').createInterface({
54     input: process.stdin,
55     output: process.stdout
56 });
57
58 readline.on('line', (inputStr) => {
59     console.log(minReplacementLength(inputStr));
60     readline.close();
61 }).

```

C++

```
1  #include <iostream>
2  #include <string>
3  #include <unordered_map>
4  #include <algorithm>
5
6  using namespace std;
7  int min_replacement_length(const string &input_str) {
8      // 初始化方向键计数字典
9      unordered_map<char, int> direction_count = {{'W', 0}, {'A', 0}, {'S',
10     0}, {'D', 0}};
11
12     // 统计输入字符串中每个方向键的出现次数
13     for (char c : input_str) {
14         direction_count[c]++;
15     }
16
17     // 初始化左右指针和结果变量
18     int left = 0;
19     int right = 0;
20     int min_length = input_str.length();
21
22     // 更新右指针对应的方向键计数
23     direction_count[input_str[0]]--;
24
25     while (true) {
26         // 计算当前最大方向键计数
27         int max_count = max({direction_count['W'], direction_count['A'], d
28         irection_count['S'], direction_count['D']});
29
30         // 计算当前窗口长度和可替换的字符数
31         int window_length = right - left + 1;
32         int replaceable_chars = window_length;
33         for (const auto &kv : direction_count) {
34             replaceable_chars -= max_count - kv.second;
35         }
36
37         // 如果可替换字符数大于等于0且能被4整除，则更新结果变量
38         if (replaceable_chars >= 0 && replaceable_chars % 4 == 0) {
39             min_length = min(min_length, window_length);
40
41             // 更新左指针并检查是否越界
42             if (left < input_str.length()) {
43                 direction_count[input_str[left]]++;
44                 left++;
45             }
46         }
47         right++;
48     }
49 }
```

```

43         } else {
44             break;
45         }
46     } else {
47         // 更新右指针并检查是否越界
48         right++;
49         if (right >= input_str.length()) {
50             break;
51         }
52         direction_count[input_str[right]]--;
53     }
54 }
55
56 return min_length;
57 }
58
59 int main() {
60     string input_str;
61     cin >> input_str;
62     cout << min_replacement_length(input_str) << endl;
63     return 0;
64 }

```

C语言

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <limits.h>
4
5  // 用于计算字符频率的辅助函数
6  void calculate_frequency(const char *s, int *freq) {
7      for (int i = 0; s[i] != '\0'; i++) {
8          freq[s[i]]++;
9      }
10 }
11
12 // 主函数
13 int main() {
14     char s[100001]; // 假设输入长度最大为100000
15     scanf("%s", s); // 读取用户输入的走位字符串
16
17     int n = strlen(s); // 计算走位字符串的长度
18     int required = n / 4; // 每个方向的步数在完美走位中应当相等
19
20     // 初始化各方向的步数频率统计
21     int freq[256] = {0}; // 统计'A', 'S', 'D', 'W'出现的次数
22     calculate_frequency(s, freq); // 计算初始的频率
23
24     // 如果已经是完美走位, 则直接输出0
25     if (freq['A'] == required && freq['S'] == required &&
26         freq['D'] == required && freq['W'] == required) {
27         printf("0\n");
28         return 0;
29     }
30
31     // 初始化滑动窗口的左右边界和最小长度
32     int left = 0, min_length = INT_MAX;
33
34     // 滑动窗口的核心逻辑
35     for (int right = 0; right < n; right++) {
36         // 移动右边界, 并减少对应字符的频率
37         freq[s[right]]--;
38
39         // 检查是否满足完美走位的条件
40         while (freq['A'] <= required && freq['S'] <= required &&
41             freq['D'] <= required && freq['W'] <= required) {
42             // 更新最小长度
43             min_length = (right - left + 1) < min_length ? (right - left
+ 1) : min_length;
```

```

44
45         // 移动左边界，并恢复对应字符的频率
46         freq[s[left]]++;
47         left++;
48     }
49 }
50
51 // 输出最小替换长度
52 printf("%d\n", min_length);
53 return 0;
54 }
```

完整用例

用例1

	▼	Plain Text
1	WASDAASD	

用例2

	▼	Plain Text
1	AAAA	

用例3

	▼	Plain Text
1	WASD	

用例4

	▼	Plain Text
1	WWW	

用例5

▼	Plain Text
1	WASDWASD

用例6

▼	Plain Text
1	WASDASDW

用例7

▼	Plain Text
1	WASDAAAD

用例8

▼	Plain Text
1	WASDSSSD

用例9

▼	Plain Text
1	WASDDDDD

用例10

▼	Plain Text
1	WASDWSWD

文章目录

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

- 示例1
- 示例2
- 解题思路## 分析
 - 代码思路
- Java
- Python
- JavaScript
- C++
- C语言
- 完整用例
 - 用例1
 - 用例2
 - 用例3
 - 用例4
 - 用例5
 - 用例6
 - 用例7
 - 用例8
 - 用例9
 - 用例10



来自: 华为OD机考2025B卷 – 完美走位 (Java & Python& JS & C++ & C) –CSDN博客

华为OD机考2025B卷 - 异常的打卡记录（Java & Python& JS & C++ & C） -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)
华为OD面试真题精选：[点击立即查看](#)

题目描述

考勤记录是分析和考核职工工作时间利用情况的原始依据，也是计算职工工资的原始依据，为了正确地计算职工工资和监督工资基金使用情况，公司决定对员工的手机打卡记录进行异常排查。

如果出现以下两种情况，则认为打卡异常：

1. 实际设备号与注册设备号不一样
2. 或者，同一个员工的两个打卡记录的时间小于60分钟并且打卡距离超过5km。

给定打卡记录的[字符串数组]clockRecords（每个打卡记录组成为：工号;时间（分钟）;打卡距离（km）；实际设备号;注册设备号），返回其中异常的打卡记录（按输入顺序输出）。

输入描述

第一行输入为N，表示打卡记录数；
之后的N行为打卡记录，每一行为一条打卡记录。

输出描述

输出[异常的](#)打卡记录。

备注

- clockRecords长度 ≤ 1000
- clockRecords[i] 格式：{id},{time},{distance},{actualDeviceNumber},{registeredDeviceNumber}
- id由6位数字组成
- time由整数组成，范围为0~1000
- distance由整数组成，范围为0~100
- actualDeviceNumber与registeredDeviceNumber由思维大写字母组成

用例

输入	2 100000,10,1,ABCD,ABCD 100000,50,10,ABCD,ABCD
----	--

输出	100000,10,1,ABCD,ABCD;100000,50,10,ABCD,ABCD
说明	第一条记录是异常得，因为第二题记录与它得间隔不超过60分钟，但是打卡距离超过了5km，同理第二条记录也是异常得。

输入	2 100000,10,1,ABCD,ABCD 100001,80,10,ABCE,ABCE
输出	null
说明	无异常打卡记录，所以返回null

输入	2 100000,10,1,ABCD,ABCD 100000,80,10,ABCE,ABCD
输出	100000,80,10,ABCE,ABCD
说明	第二条记录得注册设备号与打卡设备号不一致，所以是异常记录

题目解析

第一步是对单条打卡记录进行比较，判断其是否异常。异常情况有两种：一是实际设备号和注册设备号不同；二是同一员工的两条打卡记录时间间隔小于60分钟但是打卡距离超过5km。

第二步是对同一员工下设备号一致的打卡记录进行两两对比，判断是否异常。如果有两个打卡记录时间间隔小于60分钟且打卡距离超过5km，则视为异常。需要注意的是，一旦有两条打卡记录对比异常了，其他打卡记录也需要和这两条异常记录对比。

在保存异常打卡记录时，需要记录其输入时的索引，以便按照输入顺序输出异常记录。

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <set>
5  #include <algorithm>
6  using namespace std;
7
8  /* 按操作符分隔字符串 */
9  vector<string> split_str( string params_str, string op )
10 {
11     vector<string> p;
12     while ( params_str.find( op ) != string::npos )
13     {
14         int found = params_str.find( op );
15         p.push_back( params_str.substr( 0, found ) );
16         params_str = params_str.substr( found + 1 );
17     }
18     p.push_back( params_str );
19     return(p);
20 }
21
22
23 /* 用于比较两个打卡记录 */
24 bool comp( vector<string> & a, vector<string> & b )
25 {
26     return(stoi( a[1] ) > stoi( b[1] ) );
27 }
28
29
30 int main()
31 {
32     /* 读取输入 */
33     string num_str;
34     getline( cin, num_str );
35     int n = stoi( num_str );
36     vector<vector<string> > records; /* 存放所有打卡记录 */
37     for ( int i = 0; i < n; i++ )
38     {
39         string record_str;
40         getline( cin, record_str );
41         records.push_back( split_str( record_str, "," ) );
42     }
43
44
```

```

45     map<string, vector<vector<string> > > record_map;                /* 存放每
46 位员工的打卡记录 */
47     set<int>          result;                /* 存放异常记录的索引 */
48
49  /* 异常规则1: 同一天内, 进和出的门不一致 */
50     for ( int i = 0; i < records.size(); i++ )
51     {
52         records[i].push_back( to_string( i ) );                /* 加一个索引
53 i, 以便排序后输出原来的顺序 */
54         vector<string> single_record = records[i];
55         if ( single_record[3] != single_record[4] )
56         {
57             result.insert( i );                /* 如果进和出的门不一
58 致, 则为异常记录 */
59             } else {
60                 if ( record_map.count( single_record[0] ) )    /* 将相同员工的记录存
61 放在一起 */
62                 {
63                     record_map[single_record[0]].push_back( single_record );
64                 } else {
65                     vector<vector<string> > temp;
66                     temp.push_back( single_record );
67                     record_map[single_record[0]] = temp;
68                 }
69             }
70         }
71     }
72
73  /* 异常规则2: 在一小时内, 距离相差超过5米 */
74     for ( auto single_records_info : record_map )
75     {
76         vector<vector<string> > single_employee_records = single_records_inf
77 o.second;
78
79         /* 用打卡时间排序, 以加速双层循环 */
80         sort( single_employee_records.begin(), single_employee_records.end
81 (), comp );
82
83         for ( int i = 0; i < single_employee_records.size(); i++ )
84         {
85             int time1 = stoi( single_employee_records[i][1] );
86             int dist1 = stoi( single_employee_records[i][2] );
87
88             for ( int j = i + 1; j < single_employee_records.size(); j++ )
89             {
90                 int time2 = stoi( single_employee_records[j][1] );
91                 int dist2 = stoi( single_employee_records[j][2] );

```

```

87
88     /* 如果当前的两次打卡时间超过60分钟，则后续的肯定也超过60分钟了 */
89     if ( time2 - time1 >= 60 )
90     {
91         break;
92     } else {
93         if ( abs( dist2 - dist1 ) > 5 )
94         {
95             result.insert( stoi( single_employee_records[i][5] ) );
96             result.insert( stoi( single_employee_records[j][5] ) );
97         }
98     }
99 }
100 }
101 }
102
103 /* 输出结果 */
104 if ( result.empty() )
105 {
106     cout << "null";
107 } else {
108     for ( int i : result ) /* 遍历异常记录的索引 */
109     {
110         for ( int j = 0; j < records[i].size() - 1; j++ )
111         {
112             cout << records[i][j];
113             if ( j != records[i].size() - 2 )
114             {
115                 cout << ",";
116             }
117         }
118         cout << ":";
119     }
120 }
121

```

JavaScript

```
1 // 引入readline模块
2 const readline = require("readline");
3
4 // 创建readline接口实例
5 const rl = readline.createInterface({
6   input: process.stdin,
7   output: process.stdout,
8 });
9
10 // 定义全局变量
11 const records = []; // 所有打卡记录
12 let numEmployees; // 员工数量
13
14 // 监听每一行输入
15 rl.on("line", (line) => {
16   // 将输入的每一行记录存入records数组中
17   records.push(line);
18
19   // 如果记录数量达到要求, 则开始处理
20   if (records.length === numEmployees + 1) {
21     // 获取结果并输出
22     const result = getResult(records.slice(1));
23     console.log(result);
24
25     // 重置记录数组和员工数量
26     records.length = 0;
27     numEmployees = 0;
28   } else if (records.length === 1) {
29     // 如果记录数量为1, 则说明是员工数量, 将其存入numEmployees变量中
30     numEmployees = parseInt(line);
31   }
32 });
33
34 /**
35  * @param {*} clockRecords 打卡记录的字符串数组, [工号, 时间, 打卡距离, 实际设备
36  * 号, 注册设备号]
37  */
38 function getResult(clockRecords) {
39   const employees = {}; // 员工打卡记录的对象
40   const ans = new Set(); // 异常打卡记录的集合
41
42   // 遍历所有打卡记录, 将异常记录加入ans集合, 将正常记录存入employees对象中
43   for (let i = 0; i < clockRecords.length; i++) {
```

```

44     const clockRecord = [...clockRecords[i].split(","), i]; // 将打卡记录转
45     为数组形式，并追加输入索引
46     const [id, time, dis, actDevice, regDevice, index] = clockRecord;
47
48     // 如果实际设备号与注册设备号不一致，则认为是异常打卡记录
49     if (actDevice !== regDevice) {
50         ans.add(index);
51     } else {
52         // 如果实际设备号与注册设备号一致，则将打卡记录存入employees对象中
53         if (employees[id]) {
54             employees[id].push(clockRecord);
55         } else {
56             employees[id] = [clockRecord];
57         }
58     }
59 }
60
61 // 遍历employees对象，检查每个员工的打卡记录是否存在异常
62 for (let id in employees) {
63     const records = employees[id]; // 该员工的所有打卡记录
64     const n = records.length; // 打卡记录数量
65
66     // 将该员工的打卡记录按照打卡时间升序排序
67     records.sort((a, b) => a[1] - b[1]);
68
69     // 遍历该员工的打卡记录，检查是否存在异常
70     for (let i = 0; i < n; i++) {
71         const time1 = records[i][1];
72         const dis1 = records[i][2];
73
74         for (let j = i + 1; j < n; j++) {
75             const time2 = records[j][1];
76             const dis2 = records[j][2];
77
78             // 如果两次打卡时间超过60分钟，则后面的打卡记录不用再检查了
79             if (time2 - time1 >= 60) {
80                 break;
81             } else {
82                 // 如果两次打卡时间小于60分钟，且打卡距离超过5公里，则认为是异常打卡记录
83                 if (Math.abs(dis2 - dis1) > 5) {
84                     // 如果打卡记录已经加入ans集合，则无需再次加入，否则需要加入
85                     if (!ans.has(records[i][5])) {
86                         ans.add(records[i][5]);
87                     }
88
89                     if (!ans.has(records[j][5])) {
90                         ans.add(records[j][5]);

```

```

91         }
92     }
93 }
94 }
95 }
96 }
97
98 // 如果没有异常打卡记录，则返回null
99 if (!ans.size) {
100     return "null";
101 }
102
103 // 将异常打卡记录按照输入顺序排序，并返回结果字符串
104 return [...ans]
105     .sort((a, b) => a - b)
106     .map((i) => clockRecords[i])
    (...ans)

```

Java


```

1  import java.util.ArrayList;
2  import java.util.Arrays;
3  import java.util.HashMap;
4  import java.util.Scanner;
5  import java.util.TreeSet;
6  import java.util.stream.Collectors;
7  class Main {
8      public static void main(String[] args) {
9          Scanner scanner = new Scanner(System.in);
10         // 读取输入
11         int numRecords = scanner.nextInt(); // 打卡记录数目
12         scanner.nextLine();
13         String[][] records = new String[numRecords][];
14         for (int i = 0; i < numRecords; i++) {
15             records[i] = scanner.nextLine().split(",");
16         }
17
18         // 存放每位员工的打卡记录
19         HashMap<String, ArrayList<String[]>> recordMap = new HashMap<>();
20         TreeSet<Integer> result = new TreeSet<>();
21
22         // 异常规则1: 打卡时间不一致
23         for (int i = 0; i < records.length; i++) {
24             // 在打卡记录后面增加一个索引, 方便后面输出时按照输入顺序排序
25             String[] singleRecord = Arrays.copyOf(records[i], records[i].length + 1);
26             singleRecord[singleRecord.length - 1] = i + "";
27
28             if (!singleRecord[3].equals(singleRecord[4])) {
29                 result.add(i);
30             } else {
31                 String id = singleRecord[0];
32                 if (recordMap.containsKey(id)) {
33                     recordMap.get(id).add(singleRecord);
34                 } else {
35                     ArrayList<String[]> list = new ArrayList<>();
36                     list.add(singleRecord);
37                     recordMap.put(id, list);
38                 }
39             }
40         }
41
42         // 异常规则2: 在60分钟内, 两次打卡地点距离大于5米
43         for (String id : recordMap.keySet()) {

```

```

44         ArrayList<String[]> idRecords = recordMap.get(id);
45
46         // 按照打卡时间排序，以便后面双层循环加速
47         idRecords.sort((a, b) -> Integer.parseInt(a[1]) - Integer.parseInt(b[1]));
48
49         for (int i = 0; i < idRecords.size(); i++) {
50             int time1 = Integer.parseInt(idRecords.get(i)[1]);
51             int dist1 = Integer.parseInt(idRecords.get(i)[2]);
52
53             for (int j = i + 1; j < idRecords.size(); j++) {
54                 int time2 = Integer.parseInt(idRecords.get(j)[1]);
55                 int dist2 = Integer.parseInt(idRecords.get(j)[2]);
56
57                 // 如果当前的两次打卡时间超过60分钟，后面的肯定也超过60分钟了
58                 if (time2 - time1 >= 60) {
59                     break;
60                 } else {
61                     if (Math.abs(dist2 - dist1) > 5) {
62                         result.add(Integer.parseInt(idRecords.get(i)
63 [5]));
64
65                         result.add(Integer.parseInt(idRecords.get(j)
66 [5]));
67
68                     }
69                 }
70             }
71         }
72
73         // 输出结果
74         if (result.isEmpty()) {
75             System.out.println("null");
76         } else {
77             String resStr = result.stream()
78                 .map(index -> join(records[index]))
79                 .collect(Collectors.joining(":"));
80             System.out.println(resStr);
81         }
82
83         // 将字符串数组拼接成一个字符串
84         public static String join(String[] strs) {
85             String s = "";
86             for (String str : strs) {
87                 s+=str+",";
88             }
89             return s.substring(0, s.length()-1);

```

```
88  
89     }
```

Python

```
1  n = int(input())
2  clock_records = [input().split(",") for i in range(n)]
3
4
5  # 算法入口
6  def get_result(clock_records):
7      employees = {}
8      ans = set()
9
10     for i in range(len(clock_records)):
11         id, time, dis, act_device, reg_device = clock_records[i]
12
13         # 实际设备号与注册设备号不一样,则认为打卡异常
14         if act_device != reg_device:
15             ans.add(i)
16         # 如果实际设备号和注册设备号一样,则统计到该员工名下
17         else:
18             # 由于异常打卡记录需要按输入顺序输出,因此这里追加一个输入索引到打卡记录中
19             clock_record = [id, time, dis, act_device, reg_device, i]
20             if employees.get(id) is None:
21                 employees[id] = [clock_record]
22             else:
23                 employees[id].append(clock_record)
24
25     for id in employees.keys():
26         # 某id员工的所有打卡记录
27         records = employees[id]
28         n = len(records)
29
30         # 将该员工打卡记录按照打卡时间升序
31         records.sort(key=lambda x: x[1])
32
33         for i in range(n):
34             time1 = int(records[i][1])
35             dis1 = int(records[i][2])
36
37             for j in range(i + 1, n):
38                 time2 = int(records[j][1])
39                 dis2 = int(records[j][2])
40
41                 # 如果两次打卡时间超过60分治,则不计入异常,由于已按打卡时间升序,因此
42                 # 后面的都不用检查了
43                 if time2 - time1 >= 60:
44                     break
```

```

44         else:
45             # 如果两次打开时间小于60MIN, 且打卡距离超过5KM, 则这两次打卡记
录算作异常
46             if abs(dis2 - dis1) > 5:
47                 # 如果打卡记录已经加入异常列表ans, 则无需再次加入, 否则需要
加入
48                 if records[i][5] not in ans:
49                     ans.add(records[i][5])
50
51                 if records[j][5] not in ans:
52                     ans.add(records[j][5])
53
54             if len(ans) > 0:
55                 tmp = list(ans)
56                 tmp.sort()
57                 return ";".join(list(map(lambda i: ",".join(clock_records
[i]), tmp)))
58             else:
59                 return "null"
60
61
62 # 调用算法
63 print(get_result(clock_records))

```

机考真题

华为OD



CSDN @算法大师

来自: [华为OD机考2025B卷 – 异常的打卡记录 \(Java & Python& JS & C++ & C \)](#) –CSDN博客

华为OD机考2025B卷 - 高矮个子排队（Java & Python& JS & C++ & C） -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)
华为OD面试真题精选：[点击立即查看](#)
华为OD机考2025B卷

题目描述

现在有一队小朋友，他们高矮不同，我们以正整数数组表示这一队小朋友的身高，如数组{5,3,1,2,3}。我们现在希望小朋友排队，以“高”“矮”“高”“矮”顺序排列，每一个“高”位置的小朋友要比相邻的位置高或者相等；每一个“矮”位置的小朋友要比相邻的位置矮或者相等；要求小朋友们移动的距离和最小，第一个从“高”位开始排，输出最小移动距离即可。

例如，在示范小队{5,3,1,2,3}中，{5, 1, 3, 2, 3}是排序结果。
{5, 2, 3, 1, 3} 虽然也满足“高”“矮”“高”“矮”顺序排列，但小朋友们的移动距离大，所以不是最优结果。

移动距离的定义如下所示：

第二位小朋友移到第三位小朋友后面，移动距离为1，若移动到第四位小朋友后面，移动距离为2；

输入描述

排序前的小朋友，以英文空格的正整数：

4 3 5 7 8

注：小朋友<100个

输出描述

排序后的小朋友，以英文空格分割的正整数：4 3 7 5 8

备注：4（高）3（矮）7（高）5（矮）8（高）， 输出结果为最小移动距离，只有5和7交换了位置，移动距离都是1。

示例1

输入

▼

Plain Text

1 4 1 3 5 2

输出

▼	Plain Text
1	4 1 5 2 3

说明

示例2

输入

▼	Plain Text
1	1 1 1 1 1 1

输出

▼	Plain Text
1	1 1 1 1 1 1

说明

相邻位置可以相等

示例3

输入

▼	Plain Text
1	xxx

输出

▼	Plain Text
1	[]

说明

出现非法参数情况， 返回空数组。

解题思路

这道题看似简单，但看完题目可能会觉得不止这么简单。因为要保证移动距离最小，这意味着可能存在多种情况需要多次比较。不过，实际并没有那么复杂。

比如在用例1中，乍一看好像有点问题：直接让5和2交换位置，得到的结果是：4 1 3 2 5，这样也符合题意，而且移动距离只有1，似乎更符合要求。

然而，这样想的同学可能忽略了题目中的一句关键提示：“第一个从‘高’位开始排”。这句话的意思是，我们只需要从第一个小朋友开始排列，并在发现不符合要求的排队顺序时，就进行交换。这样大大降低了题目难度。

代码解释

为了实现将小朋友的身高按照“高矮交替”的顺序排列，代码中使用了以下判断条件：

这个条件可以分成两个部分来解释：

1. `heights[i] != heights[j]` :
 - 这个条件确保只在两个相邻的小朋友身高不相等的情况下才进行进一步的判断。如果两个小朋友的身高相等，那么无需交换位置，因为它们已经符合“高矮交替”的要求。
2. `(heights[i] > heights[j]) != (i % 2 == 0)` :
 - 这个条件用于检查当前的排列是否符合“高矮交替”的要求。
 - `i % 2 == 0` : 判断当前索引 `i` 是否为偶数。根据题目的要求，如果 `i` 是偶数位置，那么我们期望 `heights[i] > heights[j]`，即当前小朋友的身高应该高于下一个小朋友的身高。如果 `i` 是奇数位置，我们期望 `heights[i] < heights[j]`。
 - `(heights[i] > heights[j]) != (i % 2 == 0)` : 这一部分的意思是：
 - 如果 `i` 是偶数，那么期望 `heights[i] > heights[j]`。如果此时 `heights[i] > heights[j]` 为 `true`，与 `i % 2 == 0` 的结果相同，所以条件成立，说明不需要交换。
 - 如果 `i` 是奇数，那么期望 `heights[i] < heights[j]`。如果此时 `heights[i] > heights[j]` 为 `false`，与 `i % 2 == 0` 的结果相反，条件成立，说明不需要交换。
 - 如果实际情况与期望情况不符（即 `heights[i] > heights[j]` 和 `i` 的奇偶性不一致），那么 `(heights[i] > heights[j]) != (i % 2 == 0)` 结果为 `true`，表示当前排列不符合要求，需要交换 `heights[i]` 和 `heights[j]`。



Plain Text

```
1 heights[i] != heights[j] && (heights[i] > heights[j]) != (i % 2 == 0)
```



```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner sc = new Scanner(System.in);
6          String s = sc.nextLine();
7
8          // 使用正则表达式检查输入字符串是否只包含数字和空格
9          // 如果字符串中包含非法字符（非数字或空格），则输出"[]"并退出程序
10         if (!s.matches("[0-9\\s]+")) {
11             System.out.println("[]");
12             return;
13         }
14
15         // 将输入字符串按空格分割，并将每个部分转换为整数，存储在数组heights中
16         int[] heights = Arrays.stream(s.split(" ")).mapToInt(Integer::parseInt).toArray();
17
18         // 初始化两个指针i和j，分别指向相邻的两个小朋友
19         int i = 0, j = 1;
20
21         // 遍历数组，调整相邻小朋友的身高顺序以满足"高矮交替"的要求
22         while (j < heights.length) {
23             // 判断当前两个相邻小朋友的身高是否满足要求
24             // 条件解释：如果heights[i] > heights[j]且i是偶数，或者heights[i]
25             // < heights[j]且i是奇数
26             // 则需要交换heights[i]和heights[j]的值，以符合"高矮交替"的规则
27             if (heights[i] != heights[j] && (heights[i] > heights[j]) !=
28                 (i % 2 == 0)) {
29                 // 交换heights[i]和heights[j]的值
30                 int tmp = heights[i];
31                 heights[i] = heights[j];
32                 heights[j] = tmp;
33             }
34
35             // 移动指针，检查下一个相邻的小朋友
36             i++;
37             j++;
38         }
39
40         // 使用StringJoiner将排序后的身高数组转换为字符串，并以空格分隔
41         StringJoiner sj = new StringJoiner(" ");
42         for (int h : heights) { // 遍历heights数组中的每一个元素
43             sj.add(String.valueOf(h));
44         }
45         System.out.println(sj.toString());
46     }
47 }
```

```
42         sj.add(String.valueOf(h)); // 将元素转换为字符串并添加到StringJoi
43     ner中
44     }
45     // 输出最终排序结果
46     System.out.println(sj.toString());
    }
```

Python

▼ Plain Text

```
1  import re
2
3  s = input()
4  if not re.match(r"[0-9\s]+", s):
5      print("[]")
6      exit()
7
8  heights = list(map(int, s.split()))
9
10 i = 0
11 j = 1
12
13 while j < len(heights):
14     if heights[i] != heights[j] and (heights[i] > heights[j]) != (i % 2 =
    = 0):
15         heights[i], heights[j] = heights[j], heights[i]
16
17     i += 1
18     j += 1
19
20 result = " ".join(map(str, heights))
21 print(result)
```

JavaScript

```
1  const readline = require('readline');
2
3
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7  });
8
9
10 rl.on('line', (s) => {
11     // 使用正则表达式检查输入字符串是否只包含数字和空格
12     // 如果字符串中包含非法字符（非数字或空格），则输出"[]"并退出程序
13     if (!/^[0-9\s]+$/.test(s)) {
14         console.log("[]");
15         rl.close(); // 关闭接口
16         return;
17     }
18
19     // 将输入字符串按空格分割，并将每个部分转换为整数，存储在数组heights中
20     let heights = s.split(' ').map(Number);
21
22     // 初始化两个指针i和j，分别指向相邻的两个小朋友
23     let i = 0, j = 1;
24
25     // 遍历数组，调整相邻小朋友的身高顺序以满足"高矮交替"的要求
26     while (j < heights.length) {
27         // 判断当前两个相邻小朋友的身高是否满足要求
28         // 条件解释：如果heights[i] > heights[j]且i是偶数，或者heights[i] < heights[j]且i是奇数
29         // 则需要交换heights[i]和heights[j]的值，以符合"高矮交替"的规则
30         if (heights[i] !== heights[j] && (heights[i] > heights[j]) !== (i % 2 === 0)) {
31             // 交换heights[i]和heights[j]的值
32             [heights[i], heights[j]] = [heights[j], heights[i]];
33         }
34
35         // 移动指针，检查下一个相邻的小朋友
36         i++;
37         j++;
38     }
39
40     // 将排序后的身高数组转换为字符串，并以空格分隔
41     console.log(heights.join(' '));
42 }
```

```
43
```

```
44     }):
```

C++

```
1  #include <iostream>      // 包含输入输出流库
2  #include <sstream>       // 包含字符串流库，用于处理字符串
3  #include <string>        // 包含字符串库
4  #include <vector>        // 包含向量库，用于动态数组
5  using namespace std;
6
7  int main() {
8      string s;
9      // 从标准输入读取一行字符串，存储在变量s中
10     getline(cin, s);
11
12     // 检查输入字符串中是否包含非数字或空格的字符
13     // 如果发现非法字符（非数字或空格），则输出"[]"并退出程序
14     if (s.find_first_not_of("0123456789 ") != string::npos) {
15         cout << "[]" << endl;
16         return 0;
17     }
18
19     // 使用字符串流将字符串s按空格分割，并依次转化为整数存入向量heights中
20     istreamstringstream iss(s);
21     vector<int> heights; // 定义一个整型向量用于存储小朋友的身高
22     int height;
23     while (iss >> height) { // 从字符串流中读取一个整数，并存入heights向量
24         heights.push_back(height);
25     }
26
27     // 初始化两个索引i和j，分别指向相邻的两个小朋友
28     int i = 0, j = 1;
29     while (j < heights.size()) { // 遍历向量，直到处理完所有元素
30         // 检查当前两个相邻位置是否满足"高矮高矮"的排列要求
31         // 如果heights[i] > heights[j] 且 i 是偶数，或者 heights[i] < heights
148 [j] 且 i 是奇数
32         // 则说明当前排列不符合要求，需要交换两个元素的位置
33         if (heights[i] != heights[j] && (heights[i] > heights[j]) != (i %
149 2 == 0)) {
34             // 交换 heights[i] 和 heights[j] 的值
35             int tmp = heights[i];
36             heights[i] = heights[j];
37             heights[j] = tmp;
38         }
39         // 移动索引i和j，继续检查下一个相邻的元素对
40         i++;
41         j++;
42     }
```

```
43
44     // 将调整后的向量heights中的元素转换为字符串，准备输出
45     string result;
46     for (int h : heights) { // 遍历向量中的每一个元素
47         result += to_string(h) + " "; // 将元素转换为字符串并拼接到result中，
以空格分隔
48     }
49     result.pop_back(); // 移除最后一个多余的空格
50     cout << result << endl; // 输出最终的排列结果
51
52     return 0; // 程序结束
53 }
```

C语言

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <ctype.h>
5
6  // 函数声明
7  int is_valid_input(const char* s);
8  int* split_and_convert(const char* s, int* size);
9  void swap(int* a, int* b);
10
11 int main() {
12     char s[1024];
13     fgets(s, sizeof(s), stdin); // 读取用户输入
14
15     // 检查输入字符串是否只包含数字和空格
16     if (!is_valid_input(s)) {
17         printf("[]\n");
18         return 0;
19     }
20
21     int size;
22     int* heights = split_and_convert(s, &size); // 将输入字符串分割并转换为整
数数组
23
24     // 初始化两个指针i和j, 分别指向相邻的两个小朋友
25     int i = 0, j = 1;
26
27     // 遍历数组, 调整相邻小朋友的身高顺序以满足"高矮交替"的要求
28     while (j < size) {
29         // 判断当前两个相邻小朋友的身高是否满足要求
30         // 条件解释: 如果heights[i] > heights[j]且i是偶数, 或者heights[i] < he
ights[j]且i是奇数
31         // 则需要交换heights[i]和heights[j]的值, 以符合"高矮交替"的规则
32         if (heights[i] != heights[j] && (heights[i] > heights[j]) != (i %
2 == 0)) {
33             // 交换heights[i]和heights[j]的值
34             swap(&heights[i], &heights[j]);
35         }
36
37         // 移动指针, 检查下一个相邻的小朋友
38         i++;
39         j++;
40     }
41 }
```

```

42     // 输出最终排序结果
43     for (int k = 0; k < size; k++) {
44         if (k > 0) {
45             printf(" ");
46         }
47         printf("%d", heights[k]);
48     }
49     printf("\n");
50
51     free(heights); // 释放动态分配的内存
52     return 0;
53 }
54
55 // 检查输入字符串是否只包含数字和空格
56 int is_valid_input(const char* s) {
57     while (*s) {
58         if (!isdigit(*s) && !isspace(*s)) {
59             return 0; // 非法字符
60         }
61         s++;
62     }
63     return 1;
64 }
65
66 // 将输入字符串按空格分割并转换为整数数组
67 int* split_and_convert(const char* s, int* size) {
68     int* heights = malloc(1024 * sizeof(int)); // 假设数组最大长度为1024
69     *size = 0;
70
71     char* token = strtok(strdup(s), " ");
72     while (token != NULL) {
73         heights[(*size)++] = atoi(token);
74         token = strtok(NULL, " ");
75     }
76     return heights;
77 }
78
79 // 交换两个整数的值
80 void swap(int* a, int* b) {
81     int temp = *a;
82     *a = *b;
83     *b = temp;
84 }

```

完整用例

用例1

▼	Plain Text	
1	4 3 5 7 8	

用例2

▼	Plain Text	
1	4 1 3 5 2	

用例3

▼	Plain Text	
1	1 1 1 1 1	

用例4

▼	Plain Text	
1	xxx	

用例5

▼	Plain Text	
1	5 4 3 2 1	

用例6

▼	Plain Text	
1	1 1 1 2 2	

用例7

▼	Plain Text	
1	10 9 8 7 6 5 4 3 2 1	

用例8

▼ Plain Text |

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

用例9

▼ Plain Text |

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

用例10

▼ Plain Text |

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

文章目录

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

- [用例8](#)
- [用例9](#)
- [用例10](#)



来自: [华为OD机考2025B卷 – 高矮个子排队 \(Java & Python& JS & C++ & C \)](#) –CSDN博客

华为OD机考2025B卷 - 分班问题/幼儿园分班 (Java & Python& JS & C++ & C) -CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

华为OD机考2025B卷

题目描述

儿园两个班的小朋友在排队时混在了一起，每位小朋友都知道自己是否与前面一位小朋友同班，请你帮忙把同班的小朋友找出来。

小朋友的编号是整数，与前一位小朋友同班用Y表示，不同班用N表示。

输入描述

输入为空格分开的小朋友编号和是否同班标志。

比如：6/N 2/Y 3/N 4/Y，表示4位小朋友，2和6同班，3和2不同班，4和3同班。

其中，小朋友总数不超过999，每个小朋友编号大于0，小于等于999。

不考虑输入格式错误问题。

输出描述

输出为两行，每一行记录一个班小朋友的编号，编号用空格分开，且：

1. 编号需按照大小升序排列，分班记录中第一个编号小的排在第一行。
2. 若只有一个班的小朋友，第二行为空行。
3. 若输入不符合要求，则直接输出字符串ERROR。

示例1

输入

▼	Plain Text
1	1/N 2/Y 3/N 4/Y

输出

▼		Plain Text
1	1 2	
2	3 4	

说明

2的同班标记为Y，因此和1同班。
3的同班标记为N，因此和1、2不同班。
4的同班标记为Y，因此和3同班。
所以1、2同班，3、4同班，输出为

```
1 2
3 4
```

示例2

输入

▼		Plain Text
1	1/N 2/Y 3/N 4/Y 5/Y	

输出

▼		Plain Text
1	1 2	
2	3 4 5	

说明

无

解题思路

题目的要求是将一组小朋友按班级进行分类。输入由小朋友的编号和他们是否与前一位小朋友同班的标志组成。任务是根据这些标志将同班的小朋友归类并输出，遵循以下规则：

1. 输入是小朋友编号与他们是否与前一位小朋友同班的标志，用空格分隔。编号后面跟随一个标志：
 - Y：表示与前一个小朋友同班。
 - N：表示与前一个小朋友不同班。
2. 根据这些标志，小朋友们要被分成不同的班级。

示例分析

示例 1

输入：

▼	Plain Text
1	1/N 2/Y 3/N 4/Y

解释：

- 小朋友1是第一个，所以他是班级1的第一个成员。
- 小朋友2与小朋友1同班（因为标志是Y），因此小朋友1和2同班，形成班级1。
- 小朋友3与小朋友2不同班（标志是N），因此小朋友3在另一个班级，形成班级2。
- 小朋友4与小朋友3同班（标志是Y），所以小朋友3和4同班，属于班级2。

输出：

▼	Plain Text
1	1 2
2	3 4

示例 2

输入：

▼	Plain Text
1	1/N 2/Y 3/N 4/Y 5/Y

解释：

- 小朋友1和2同班，属于班级1。
- 小朋友3和4同班，属于班级2。
- 小朋友5与4同班，属于班级2。

输出：

▼	Plain Text
1	1 2
2	3 4 5

```
1  import java.util.ArrayList;
2  import java.util.Collections;
3  import java.util.List;
4  import java.util.Scanner;
5
6  public class Main {
7      public static void main(String[] args) {
8          Scanner scanner = new Scanner(System.in);
9          String[] nums = scanner.nextLine().split(" ");
10
11          String[] start = nums[0].split("/");
12          List<String> class_A = new ArrayList<>();
13          class_A.add(start[0]);
14          List<String> class_B = new ArrayList<>();
15
16          List<List<String>> temp = new ArrayList<>();
17          temp.add(class_A);
18          temp.add(class_B);
19
20          for (int i = 1; i < nums.length; i++) {
21              String[] current = nums[i].split("/");
22              String id_ = current[0];
23              String f = current[1];
24
25              if (f.equals("Y")) {
26                  temp = temp;
27              } else {
28                  Collections.reverse(temp);
29              }
30
31              temp.get(0).add(id_);
32          }
33
34          if (!class_A.isEmpty()) {
35              Collections.sort(class_A, (a, b) -> Integer.parseInt(a) - Integer.parseInt(b));
36              System.out.println(String.join(" ", class_A));
37          }
38
39          if (!class_B.isEmpty()) {
40              Collections.sort(class_B, (a, b) -> Integer.parseInt(a) - Integer.parseInt(b));
41              System.out.println(String.join(" ", class_B));
42          }
```

```
43     }
44 }
```

Python

```
▼ Plain Text |
1  nums = input().split()
2
3  # 将第一个元素以 '/' 分隔成两部分，第一部分表示小朋友的编号，第二部分表示是否与前一位小朋友同班
4  start = nums[0].split('/')
5  # 创建一个列表class_A，用于存放同班的小朋友编号
6  class_A = [start[0]]
7  # 创建一个列表class_B，用于存放不同班的小朋友编号
8  class_B = []
9
10 # 创建一个临时列表temp，用于存放两个班级的小朋友编号列表
11 temp = [class_A, class_B]
12
13 # 遍历nums列表中的每一个元素
14 for n in nums[1:]:
15     # 将当前元素以 '/' 分隔成两部分，第一部分表示小朋友的编号，第二部分表示是否与前一位小朋友同班
16     id_, f = n.split("/")
17
18     # 如果与前一位小朋友同班，则temp不变
19     if f == "Y":
20         temp = temp
21     else:
22         # 如果与前一位小朋友不同班，则将temp列表中的两个班级的小朋友编号列表颠倒顺序
23         temp = temp[::-1]
24
25     # 将当前小朋友的编号添加到temp列表的第一个班级的小朋友编号列表中
26     temp[0].append(id_)
27
28 # 如果class_A列表不为空，则按照编号的大小升序排列，并用空格分隔成字符串输出
29 if class_A:
30     print(" ".join(sorted(class_A, key=lambda x: int(x))))
31 # 如果class_B列表不为空，则按照编号的大小升序排列，并用空格分隔成字符串输出
32 if class_B:
33     print(" ".join(sorted(class_B, key=lambda x: int(x))))
```

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', (nums) => {
9    nums = nums.split(' ');
10
11    let start = nums[0].split('/');
12    let class_A = [start[0]];
13    let class_B = [];
14    let temp = [class_A, class_B];
15
16    for (let i = 1; i < nums.length; i++) {
17      let [id_, f] = nums[i].split('/');
18
19      if (f === 'Y') {
20        temp = temp;
21      } else {
22        temp = temp.reverse();
23      }
24
25      temp[0].push(id_);
26    }
27
28    if (class_A.length > 0) {
29      console.log(class_A.sort((a, b) => parseInt(a) - parseInt(b)).join('
30    '));
31    }
32    if (class_B.length > 0) {
33      console.log(class_B.sort((a, b) => parseInt(a) - parseInt(b)).join('
34    '));
35    }
36    rl.close();
37  });
```

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <sstream>
5  using namespace std;
6
7  vector<string> split(const string& s, char delimiter) {
8      vector<string> tokens;
9      string token;
10     istringstream tokenStream(s);
11     while (getline(tokenStream, token, delimiter)) {
12         tokens.push_back(token);
13     }
14     return tokens;
15 }
16
17 int main() {
18     string nums;
19     getline(cin, nums);
20
21     vector<string> nums_vec = split(nums, ' ');
22
23     vector<string> start = split(nums_vec[0], '/');
24     vector<string> class_A = {start[0]};
25     vector<string> class_B;
26     vector<vector<string>> temp = {class_A, class_B};
27
28     for (int i = 1; i < nums_vec.size(); i++) {
29         vector<string> temp_vec = split(nums_vec[i], '/');
30
31         string id_ = temp_vec[0];
32         string f = temp_vec[1];
33
34         if (f == "Y") {
35             temp = temp;
36         } else {
37             reverse(temp.begin(), temp.end());
38         }
39
40         temp[0].push_back(id_);
41     }
42
43     class_A = temp[0];
44     class_B = temp[1];
```

```

45     if (class_B.size() > 0) {
46         sort(class_B.begin(), class_B.end(), [](string a, string b) { return stoi(a) < stoi(b); });
47         for (string s : class_B) {
48             cout << s << " ";
49         }
50         cout << endl;
51     }
52     if (class_A.size() > 0) {
53         sort(class_A.begin(), class_A.end(), [](string a, string b) { return stoi(a) < stoi(b); });
54         for (string s : class_A) {
55             cout << s << " ";
56         }
57         cout << endl;
58     }
59
60
61     return 0;
62 }

```

C语言

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  // 定义最大小朋友数量
6  #define MAX_KIDS 1000
7
8  // 辅助函数，用于比较两个字符串表示的数字
9  int cmp(const void *a, const void *b) {
10     return atoi(*(const char **)a) - atoi(*(const char **)b);
11 }
12
13 // 主函数
14 int main() {
15     char input[5000]; // 假设输入不超过5000字符
16     fgets(input, sizeof(input), stdin); // 读取整行输入
17
18     char *nums[MAX_KIDS]; // 保存小朋友编号和同班标志
19     int count = 0; // 输入的条目数量
20
21     // 分割输入的每个小朋友编号和同班标志
22     char *token = strtok(input, " ");
23     while (token != NULL) {
24         nums[count++] = token;
25         token = strtok(NULL, " ");
26     }
27
28     // 定义两个班级的数组
29     char *class_A[MAX_KIDS];
30     int class_A_count = 0;
31     char *class_B[MAX_KIDS];
32     int class_B_count = 0;
33
34     // 初始化第一个小朋友
35     char *start = strtok(nums[0], "/");
36     class_A[class_A_count++] = start;
37
38     // 定义指向两个班级的数组
39     char ***temp[2] = { &class_A, &class_B }; // 指向class_A和class_B的指针
40     int temp_index = 0; // 当前处理的班级
41
42     // 遍历输入的每个小朋友，从第二个开始
43     for (int i = 1; i < count; i++) {
44         char *id_ = strtok(nums[i], "/"); // 小朋友编号
```

```

45         char *f = strtok(NULL, "/"); // 同班标志
46
47         if (strcmp(f, "N") == 0) {
48             temp_index = 1 - temp_index; // 切换到另一个班
49         }
50
51         // 将当前小朋友编号添加到当前班级
52         if (temp_index == 0) {
53             class_A[class_A_count++] = id_;
54         } else {
55             class_B[class_B_count++] = id_;
56         }
57     }
58
59     // 输出班级A的编号，升序排列
60     if (class_A_count > 0) {
61         qsort(class_A, class_A_count, sizeof(char *), cmp);
62         for (int i = 0; i < class_A_count; i++) {
63             if (i > 0) printf(" ");
64             printf("%s", class_A[i]);
65         }
66         printf("\n");
67     }
68
69     // 输出班级B的编号，升序排列
70     if (class_B_count > 0) {
71         qsort(class_B, class_B_count, sizeof(char *), cmp);
72         for (int i = 0; i < class_B_count; i++) {
73             if (i > 0) printf(" ");
74             printf("%s", class_B[i]);
75         }
76         printf("\n");
77     }
78
79     return 0;
80 }

```

完整用例

用例1

	▼	Plain Text
1	1/N 2/Y 3/N 4/Y	

用例2

▼	Plain Text
1	1/N 2/Y 3/N 4/Y 5/Y

用例3

▼	Plain Text
1	6/N 7/Y

用例4

▼	Plain Text
1	1/N 2/N 3/N 4/Y 5/Y 6/Y 7/N 8/Y 9/Y 10/Y

用例5

▼	Plain Text
1	1/N 2/Y 3/N 4/Y 5/N 6/Y 7/N 8/Y 9/N 10/Y 11/N 12/Y 13/N 14/Y 15/N 16/Y 17/N 18/Y 19/N 20/Y

用例6

▼	Plain Text
1	1/N 2/N 3/N 4/N 5/N

用例7

▼	Plain Text
1	1/N 2/Y 3/N 4/Y 5/Y 100/Y 101/N 102/Y 103/N 104/Y 105/Y

用例8

▼	Plain Text
1	1/N

用例9

▼	Plain Text
1	6/N 7/Y 8/N 9/Y

用例10

▼	Plain Text
1	10/N 11/Y 12/N 13/Y 14/N

文章目录

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

机考真题 华为OD



CSDN @算法大师

来自: [华为OD机考2025B卷 – 分班问题/幼儿园分班 \(Java & Python& JS & C++ & C \)](#) –CSDN博客

华为OD机考2025B卷 - 模拟消息队列（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)

华为OD面试真题精选：[点击立即查看](#)

华为OD机考2025B卷

题目描述

让我们来模拟一个消息队列的运作，有一个发布者和若干消费者，发布者会在给定的时刻向消息队列发送消息，

- 若此时消息队列有消费者订阅，这个消息会被发送到订阅的消费者中优先级最高（输入中消费者按优先级升序排列）的一个；
- 若此时没有订阅的消费者，该消息被消息队列丢弃。

消费者则会在给定的时刻订阅消息队列或取消订阅。

- 当消息发送和订阅发生在同一时刻时，先处理订阅操作，即同一时刻订阅的消费者成为消息发送的候选。
- 当消息发送和取消订阅发生在同一时刻时，先处理取消订阅操作，即消息不会被发送到同一时刻取消订阅的消费者。

输入描述

输入为两行。

第一行为 $2N$ 个正整数，代表发布者发送的 N 个消息的时刻和内容（为方便解折，消息内容也用正整数表示）。第一个数字是第一个消息的发送时刻，第二个数字是第一个消息的内容，以此类推。用例保证发送时刻不会重复，但注意消息并没有按照发送时刻排列。

第二行为 $2M$ 个正整数，代表 M 个消费者订阅和取消订阅的时刻。第一个数字是第一个消费者订阅的时刻，第二个数字是第一个消费者取消订阅的时刻，以此类推。用例保证每个消费者的取消订阅时刻大于订阅时刻，消费者按优先级升序排列。

两行的数字都由空格分隔。 N 不超过100， M 不超过10，每行的长度不超过1000字符。

输出描述

输出为 M 行，依次为 M 个消费者收到的消息内容，消息内容按收到的顺序排列，且由空格分隔；若某个消费者没有收到任何消息，则对应的行输出-1。

示例1

输入

		Plain Text										
1	2	22	1	11	4	44	5	55	3	33		
2	1	7	2	3								

输出

		Plain Text										
1	11	33	44	55								
2	22											

说明

消息11在1时刻到达，此时只有第一个消费者订阅，消息发送给它；
消息22在2时刻到达，此时两个消费者都订阅了，消息发送给优先级最高的第二个消费者；
消息33在时刻3到达，此时只有第一个消费者订阅，消息发送给它；
余下的消息按规则也是发送给第一个消费者。

示例2

输入

		Plain Text										
1	5	64	11	64	9	97						
2	9	11	4	9								

输出

		Plain Text										
1	97											
2	64											

说明

消息64在5时刻到达，此时只有第二个消费者订阅，消息发送给它；
消息97在9时刻到达，此时只有第一消费者订阅(因为第二个消费者刚好在9时刻取消订阅)，消息发送给它；
11时刻也到达了一个内容为64的消息，不过因为没有消费者订阅，消息被丢弃。

解题思路

```
1  import java.util.ArrayList;
2  import java.util.Arrays;
3  import java.util.List;
4  import java.util.Scanner;
5  import java.util.stream.Collectors;
6
7  public class Main {
8      public static void main(String[] args) {
9          Scanner scanner = new Scanner(System.in);
10
11         // 输入消息和消费者信息
12         String[] newsInput = scanner.nextLine().split(" ");
13         String[] consumersInput = scanner.nextLine().split(" ");
14
15         // 将消息和消费者信息转化为列表
16         List<Integer> news = new ArrayList<>();
17         List<Integer> consumers = new ArrayList<>();
18         for (String s : newsInput) {
19             news.add(Integer.parseInt(s));
20         }
21         for (String s : consumersInput) {
22             consumers.add(Integer.parseInt(s));
23         }
24
25         // 将消息和消费者信息转化为列表
26         List<List<Integer>> newsList = new ArrayList<>();
27         List<List<Integer>> consumersList = new ArrayList<>();
28         for (int i = 0; i < news.size(); i += 2) {
29             newsList.add(Arrays.asList(news.get(i), news.get(i+1)));
30         }
31         for (int i = 0; i < consumers.size(); i += 2) {
32             consumersList.add(Arrays.asList(consumers.get(i), consumers.ge
33 t(i+1)));
34         }
35
36         // 消息按照到达时刻排序
37         newsList.sort((a, b) -> a.get(0) - b.get(0));
38
39         // 定义消费者收到消息内容的列表
40         List<List<Integer>> resList = new ArrayList<>();
41         for (int i = 0; i < consumersList.size(); i++) {
42             resList.add(new ArrayList<>());
43         }
```

```

44         // 遍历每条消息
45         for (int i = 0; i < newsList.size(); i++) {
46             // 查询每条消息，按照优先级顺序是否到达（反向遍历）
47             for (int j = consumersList.size(); j > 0; j--) {
48                 // 判断消息达到时刻是否满足消费者要求
49                 if (consumersList.get(j-1).get(0) <= newsList.get(i).get
(0) && newsList.get(i).get(0) < consumersList.get(j-1).get(1)) {
50                     resList.get(j-1).add(newsList.get(i).get(1));
51                     break;
52                 }
53             }
54         }
55
56         resList.stream().forEach(contents -> {
57             if (contents.size() == 0) {
58                 System.out.println("-1");
59             } else {
60                 String result = contents.stream().map(Object::toString).col
lect(Collectors.joining(" "));
61                 System.out.println(result);
62             }
63         });
64     }
65 }

```

Python

```
1  # 输入消息和消费者信息
2  news = list(map(int, input().split()))
3  consumers = list(map(int, input().split()))
4
5  # 将消息和消费者信息转化为列表
6  news_list = []
7  consumers_list = []
8  for i in range(0, len(news), 2):
9      news_list.append([news[i], news[i+1]])
10 for i in range(0, len(consumers), 2):
11     consumers_list.append([consumers[i], consumers[i+1]])
12
13 # 消息按照到达时刻排序
14 news_list.sort(key=lambda x: x[0])
15
16 # 定义消费者收到消息内容的列表
17 res_list = [[] for _ in range(len(consumers_list))]
18
19 # 遍历每条消息
20 for i in range(len(news_list)):
21     # 查询每条消息, 按照优先级顺序是否到达 (反向遍历)
22     for j in range(len(consumers_list), 0, -1):
23         # 判断消息达到时刻是否满足消费者要求
24         if consumers_list[j-1][0] <= news_list[i][0] < consumers_list[j-1]
[1]:
25             res_list[j-1].append(news_list[i][1])
26             break
27
28 # 逐个打印消费者消息
29 for contents in res_list:
30     if len(contents) == 0:
31         print("-1")
32     else:
33         print(" ".join(map(str, contents)))
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let publisherSubscriberArray, subscriberArray;
9  let numPublisherSubscriber, numSubscriber;
10 let publishers, subscribers, subscriberContents;
11
12 rl.on('line', (line) => {
13   if (!publisherSubscriberArray) {
14     publisherSubscriberArray = line.split(' ').map(Number);
15     numPublisherSubscriber = publisherSubscriberArray.length;
16   } else if (!subscriberArray) {
17     subscriberArray = line.split(' ').map(Number);
18     numSubscriber = subscriberArray.length;
19
20     // 将发布者的信息存储到二维数组中
21     publishers = new Array(numPublisherSubscriber / 2);
22     for (let i = 0, k = 0; i < numPublisherSubscriber; i += 2) {
23       publishers[k++] = [publisherSubscriberArray[i], publisherSubscriberA
24         rray[i + 1]];
25     }
26
27     // 将订阅者的信息存储到二维数组中
28     subscribers = new Array(numSubscriber / 2);
29     for (let j = 0, k = 0; j < numSubscriber; j += 2) {
30       subscribers[k++] = [subscriberArray[j], subscriberArray[j + 1]];
31     }
32
33     // 将发布者的信息按照时间升序排序
34     publishers.sort((a, b) => a[0] - b[0]);
35
36     // 用一个列表存储每个订阅者收到的消息
37     subscriberContents = new Array(subscribers.length);
38     for (let i = 0; i < subscribers.length; i++) {
39       subscriberContents[i] = [];
40     }
41
42     // 遍历每条消息
43     for (let i = 0; i < publishers.length; i++) {
```

```

44         // 查询每条消息，按照优先级顺序是否到达（反向遍历）
45         for (let j = subscribers.length; j > 0; j--) {
46             // 判断消息达到时刻是否满足消费者要求
47             if (subscribers[j-1][0] <= publishers[i][0] && publishers[i]
[0] < subscribers[j-1][1]) {
48                 subscriberContents[j-1].push(publishers[i][1]);
49                 break;
50             }
51         }
52     }
53
54     // 输出每个订阅者收到的消息
55     for (let i = 0; i < subscriberContents.length; i++) {
56         const subscriberContent = subscriberContents[i];
57         if (subscriberContent.length === 0) {
58             console.log('-1');
59         } else {
60             console.log(subscriberContent.join(' '));
61         }
62     }
63 }
64 }):

```

C++


```
1  #include <iostream>
2  #include <vector>
3  #include <sstream>
4  #include <algorithm>
5
6  using namespace std;
7
8  int main() {
9      string newsInput, consumersInput;
10     getline(cin, newsInput);
11     getline(cin, consumersInput);
12
13     // 将输入的消息和消费者信息转化为列表
14     vector<int> news;
15     vector<int> consumers;
16     stringstream newsStream(newsInput);
17     stringstream consumersStream(consumersInput);
18     int num;
19     while (newsStream >> num) {
20         news.push_back(num);
21     }
22     while (consumersStream >> num) {
23         consumers.push_back(num);
24     }
25
26     // 将消息和消费者信息转化为列表
27     vector<vector<int>> newsList;
28     vector<vector<int>> consumersList;
29     for (int i = 0; i < news.size(); i += 2) {
30         newsList.push_back({news[i], news[i+1]});
31     }
32     for (int i = 0; i < consumers.size(); i += 2) {
33         consumersList.push_back({consumers[i], consumers[i+1]});
34     }
35
36     // 消息按照到达时刻排序
37     sort(newsList.begin(), newsList.end(), [](const vector<int>& a, const
vector<int>& b) {
38         return a[0] < b[0];
39     });
40
41     // 定义消费者收到消息内容的列表
42     vector<vector<int>> resList(consumersList.size(), vector<int>());
43
```

```

44     // 遍历每条消息
45     for (int i = 0; i < newsList.size(); i++) {
46         // 查询每条消息，按照优先级顺序是否到达（反向遍历）
47         for (int j = consumersList.size(); j > 0; j--) {
48             // 判断消息达到时刻是否满足消费者要求
49             if (consumersList[j-1][0] <= newsList[i][0] && newsList[i][0]
< consumersList[j-1][1]) {
50                 resList[j-1].push_back(newsList[i][1]);
51                 break;
52             }
53         }
54     }
55
56     // 逐个打印消费者消息
57     for (const vector<int>& contents : resList) {
58         if (contents.size() == 0) {
59             cout << "-1" << endl;
60         } else {
61             for (int i : contents) {
62                 cout << i << " ";
63             }
64             cout << endl;
65         }
66     }
67
68     return 0;
69 }

```

C语言

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #define MAX_NEWS 100
5  #define MAX_CONSUMERS 100
6
7  int main() {
8      char newsInput[256], consumersInput[256];
9      int news[MAX_NEWS][2];
10     int consumers[MAX_CONSUMERS][2];
11     int newsCount = 0, consumersCount = 0;
12
13     // 读取输入
14     fgets(newsInput, sizeof(newsInput), stdin);
15     fgets(consumersInput, sizeof(consumersInput), stdin);
16
17     // 将输入转化为整数对
18     char *token = strtok(newsInput, " ");
19     while (token != NULL) {
20         if (newsCount % 2 == 0) {
21             news[newsCount / 2][0] = atoi(token);
22         } else {
23             news[newsCount / 2][1] = atoi(token);
24         }
25         newsCount++;
26         token = strtok(NULL, " ");
27     }
28
29     token = strtok(consumersInput, " ");
30     while (token != NULL) {
31         if (consumersCount % 2 == 0) {
32             consumers[consumersCount / 2][0] = atoi(token);
33         } else {
34             consumers[consumersCount / 2][1] = atoi(token);
35         }
36         consumersCount++;
37         token = strtok(NULL, " ");
38     }
39
40     // 按到达时刻排序 (冒泡排序)
41     for (int i = 0; i < newsCount / 2 - 1; i++) {
42         for (int j = 0; j < newsCount / 2 - i - 1; j++) {
43             if (news[j][0] > news[j + 1][0]) {
44                 int temp0 = news[j][0];
```

```

45         int temp1 = news[j][1];
46         news[j][0] = news[j + 1][0];
47         news[j][1] = news[j + 1][1];
48         news[j + 1][0] = temp0;
49         news[j + 1][1] = temp1;
50     }
51 }
52 }
53
54 // 定义消费者收到消息内容的列表
55 int resList[MAX_CONSUMERS][MAX_NEWS];
56 int resCount[MAX_CONSUMERS] = {0};
57
58 // 遍历每条消息
59 for (int i = 0; i < newsCount / 2; i++) {
60     // 查询每条消息
61     for (int j = consumersCount / 2 - 1; j >= 0; j--) {
62         // 判断消息达到时刻是否满足消费者要求
63         if (consumers[j][0] <= news[i][0] && news[i][0] < consumers[j]
[1]) {
64             resList[j][resCount[j]] = news[i][1];
65             resCount[j]++;
66             break;
67         }
68     }
69 }
70
71 // 逐个打印消费者消息
72 for (int i = 0; i < consumersCount / 2; i++) {
73     if (resCount[i] == 0) {
74         printf("-1\n");
75     } else {
76         for (int j = 0; j < resCount[i]; j++) {
77             printf("%d ", resList[i][j]);
78         }
79         printf("\n");
80     }
81 }
82
83 return 0;
84 }

```

完整用例

用例1

▼	Plain Text
1	2 22 1 11 4 44 5 55 3 33
2	1 7 2 3

用例2

▼	Plain Text
1	1 10 2 20 3 30 4 40 5 50
2	1 3 2 4

用例3

▼	Plain Text
1	1 10 2 20 3 30 4 40 5 50
2	1 3 2 4 5 6

用例4

▼	Plain Text
1	1 10 2 20 3 30 4 40 5 50
2	1 3 2 4 5 6 7 8

用例5

▼	Plain Text
1	1 1 2 2 3 3 4 4 5 5
2	1 2 3 4

用例6

▼	Plain Text
1	1 5 2 4 3 3 4 2 5 1
2	1 5 2 4

用例7

▼	Plain Text
1	1 10 2 20 3 30 4 40 5 50
2	1 5 2 3

用例8

▼	Plain Text
1	1 10 2 20 3 30 4 40 5 50
2	1 5 2 3

用例9

▼	Plain Text
1	5 64 11 64 9 97
2	9 11 4 9

用例10

▼	Plain Text
1	1 10 2 20 3 30 4 40 5 50
2	1 5 2 4 3 4 5 6 6 7

文章目录

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

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



来自: [华为OD机考2025B卷 – 模拟消息队列（Java & Python& JS & C++ & C）](#) -CSDN博客

华为OD机考2025B卷 - 最小循环子数组（Java & Python& JS & C++ & C）-CSDN博客

最新华为OD机试

真题目录：[点击查看目录](#)
华为OD面试真题精选：[点击立即查看](#)

题目描述

给定一个由若干整数组成的数组nums，请检查数组是否是由某个子数组重复循环拼接而成，请输出这个最小的子数组。

输入描述

第一行输入数组中元素个数n， $1 \leq n \leq 100000$
第二行输入数组的数字序列nums，以空格分割， $0 \leq \text{nums}[i] < 10$

备注

数组本身是其最大的子数组，循环1次可生成的自身；

输出描述

输出最小的子数组的数字序列，以空格分割；

示例1

输入

▼

Plain Text

19

2121121121

输出

▼

Plain Text

1121

说明

数组[1,2,1,1,2,1,1,2,1] 可由子数组[1,2,1]重复循环3次拼接而成

解题思路

问题理解

这道题要求我们找到一个最小的子数组，该子数组通过循环拼接可以构成原始数组。例如，对于数组 [1, 2, 1, 1, 2, 1, 1, 2, 1]，最小的循环子数组是 [1, 2, 1]，它重复3次可以生成原数组。

算法原理

这个问题可以使用 [KMP 算法](#) 的思想来解决。KMP 算法主要用于字符串匹配，但其中的"前缀函数" (next 数组) 可以帮助我们找到循环模式。

KMP 的 next 数组在这里的应用

1. next数组记录了每个位置的最长相等前后缀长度
2. 对于数组末尾位置 i ， $\text{next}[i]$ 表示数组的前 $\text{next}[i]$ 个元素与末尾的 $\text{next}[i]$ 个元素相同
3. 如果数组是由某个子数组循环构成的，那么 $(\text{数组长度} - \text{next}[\text{末尾位置}])$ 就是可能的循环子数组的长度

关键思路

如果一个数组是循环构成的，假设循环子数组长度为 k ，数组长度为 n ，则有：

- n 必须是 k 的整数倍
- 如果 $(n \% (n - \text{next}[n-1])) == 0$ ，说明数组可以被分成整数个循环子数组
- 此时最小循环子数组长度为 $(n - \text{next}[n-1])$
- 否则，无法找到真正的循环子数组，整个数组本身就是最小的"循环子数组" (循环1次)

示例分析

对于输入 [1, 2, 1, 1, 2, 1, 1, 2, 1]：

1. 构建 next 数组：[0,0,1,1,2,3,4,5,6]
2. 最后一个元素的 next 值是 6
3. 计算： $9 \% (9 - 6) = 9 \% 3 = 0$ ，可以整除
4. 因此最小循环子数组长度为 3
5. 输出前 3 个元素：[1,2,1]

```
1  import java.util.Arrays;
2  import java.util.Scanner;
3  import java.util.StringJoiner;
4
5  public class Main {
6      public static void main(String[] args) {
7
8          Scanner sc = new Scanner(System.in);
9          // 读取数组长度
10         int arrayLength = Integer.parseInt(sc.nextLine());
11         // 读取数组元素并将其转换为整数数组
12         int[] nums = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
13
14         // 初始化next数组, 用于存储KMP算法中的前缀信息
15         int[] next = new int[arrayLength];
16         // 初始化前缀索引
17         int prefixIndex = 0;
18         // 初始化当前索引
19         int currentIndex = 1;
20
21         // 使用KMP算法的思想构建next数组
22         while (currentIndex < arrayLength) {
23             // 如果当前元素与前缀元素相等, 则更新next数组
24             if (nums[currentIndex] == nums[prefixIndex]) {
25                 next[currentIndex++] = ++prefixIndex;
26             } else if (prefixIndex > 0) {
27                 // 如果前缀索引大于0, 则回退到前一个前缀
28                 prefixIndex = next[prefixIndex - 1];
29             } else {
30                 // 如果前缀索引为0, 则移动到下一个元素
31                 currentIndex++;
32             }
33         }
34
35         // 获取next数组的最后一个值
36         int lastNextValue = next[arrayLength - 1];
37         // 计算子数组长度
38         int subArrayLength = arrayLength % (arrayLength - lastNextValue) =
= 0 ? arrayLength - lastNextValue : arrayLength;
39
40         // 使用StringJoiner构建输出字符串
41         StringJoiner sj = new StringJoiner(" ");
42         // 将子数组的元素添加到输出字符串中
```

```
43         for (int i = 0; i < subArrayLength; i++) sj.add(nums[i] + "");
44         // 输出结果
45         System.out.println(sj.toString());
46     }
47 }
```

Python

```
1  import sys
2
3  # 从控制台读取输入
4  input_lines = sys.stdin.readlines()
5  # 读取数组长度
6  array_length = int(input_lines[0].strip())
7  # 读取数组元素并将其转换为整数数组
8  nums = list(map(int, input_lines[1].strip().split()))
9
10 # 初始化next数组，用于存储KMP算法中的前缀信息
11 next = [0] * array_length
12 # 初始化前缀索引
13 prefix_index = 0
14 # 初始化当前索引
15 current_index = 1
16
17 # 使用KMP算法的思想构建next数组
18 while current_index < array_length:
19     # 如果当前元素与前缀元素相等，则更新next数组
20     if nums[current_index] == nums[prefix_index]:
21         prefix_index += 1
22         next[current_index] = prefix_index
23         current_index += 1
24     elif prefix_index > 0:
25         # 如果前缀索引大于0，则回退到前一个前缀
26         prefix_index = next[prefix_index - 1]
27     else:
28         # 如果前缀索引为0，则移动到下一个元素
29         current_index += 1
30
31 # 获取next数组的最后一个值
32 last_next_value = next[array_length - 1]
33 # 计算子数组长度
34 sub_array_length = array_length - last_next_value if array_length % (array_length - last_next_value) == 0 else array_length
35
36 # 使用列表构建输出字符串
37 output = ' '.join(str(nums[i]) for i in range(sub_array_length))
38 # 输出结果
39 print(output)
```

JavaScript

```
1  const readline = require('readline');
2
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout
7  });
8
9  // 读取用户输入
10 rl.on('line', arrayLength => {
11   rl.on('line', arrayElements => {
12     // 将输入的字符串转换为整数数组
13     const nums = arrayElements.split(' ').map(Number);
14     const next = new Array(parseInt(arrayLength)).fill(0);
15
16     let prefixIndex = 0;
17     let currentIndex = 1;
18
19     // 使用KMP算法的思想构建next数组
20     while (currentIndex < nums.length) {
21       if (nums[currentIndex] === nums[prefixIndex]) {
22         next[currentIndex++] = ++prefixIndex;
23       } else if (prefixIndex > 0) {
24         prefixIndex = next[prefixIndex - 1];
25       } else {
26         currentIndex++;
27       }
28     }
29
30     // 获取next数组的最后一个值
31     const lastNextValue = next[nums.length - 1];
32     // 计算子数组长度
33     const subArrayLength = nums.length % (nums.length - lastNextValue) ==
34     = 0 ? nums.length - lastNextValue : nums.length;
35
36     // 构建输出字符串
37     const output = nums.slice(0, subArrayLength).join(' ');
38     console.log(output);
39
40     rl.close();
41   });
42 }
```

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <sstream>
4  #include <string>
5  using namespace std;
6  int main() {
7      // 创建输入流对象以读取用户输入
8      string line;
9      // 读取数组长度
10     int arrayLength;
11     getline(cin, line);
12     arrayLength = stoi(line);
13
14     // 读取数组元素并将其转换为整数数组
15     getline(cin, line);
16     istringstream iss(line);
17     vector<int> nums(arrayLength);
18     for (int i = 0; i < arrayLength; ++i) {
19         iss >> nums[i];
20     }
21
22     // 初始化next数组，用于存储KMP算法中的前缀信息
23     vector<int> next(arrayLength, 0);
24     // 初始化前缀索引
25     int prefixIndex = 0;
26     // 初始化当前索引
27     int currentIndex = 1;
28
29     // 使用KMP算法的思想构建next数组
30     while (currentIndex < arrayLength) {
31         // 如果当前元素与前缀元素相等，则更新next数组
32         if (nums[currentIndex] == nums[prefixIndex]) {
33             next[currentIndex++] = ++prefixIndex;
34         } else if (prefixIndex > 0) {
35             // 如果前缀索引大于0，则回退到前一个前缀
36             prefixIndex = next[prefixIndex - 1];
37         } else {
38             // 如果前缀索引为0，则移动到下一个元素
39             currentIndex++;
40         }
41     }
42
43     // 获取next数组的最后一个值
44     int lastNextValue = next[arrayLength - 1];
```

```

45     // 计算子数组长度
46     int subArrayLength = arrayLength % (arrayLength - lastNextValue) == 0
    ? arrayLength - lastNextValue : arrayLength;
47
48     // 使用ostringstream构建输出字符串
49     stringstream oss;
50     // 将子数组的元素添加到输出字符串中
51     for (int i = 0; i < subArrayLength; i++) {
52         oss << nums[i];
53         if (i < subArrayLength - 1) {
54             oss << " ";
55         }
56     }
57     // 输出结果
58     cout << oss.str() << endl;
59
60     return 0;
61 }

```

C语言


```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  int main() {
6      // 读取数组长度
7      int arrayLength;
8      scanf("%d", &arrayLength);
9
10     // 读取数组元素
11     int* nums = (int*)malloc(arrayLength * sizeof(int));
12     for (int i = 0; i < arrayLength; i++) {
13         scanf("%d", &nums[i]);
14     }
15
16     // 初始化next数组, 用于存储KMP算法中的前缀信息
17     int* next = (int*)calloc(arrayLength, sizeof(int));
18     // 初始化前缀索引
19     int prefixIndex = 0;
20     // 初始化当前索引
21     int currentIndex = 1;
22
23     // 使用KMP算法的思想构建next数组
24     while (currentIndex < arrayLength) {
25         // 如果当前元素与前缀元素相等, 则更新next数组
26         if (nums[currentIndex] == nums[prefixIndex]) {
27             prefixIndex++;
28             next[currentIndex] = prefixIndex;
29             currentIndex++;
30         } else if (prefixIndex > 0) {
31             // 如果前缀索引大于0, 则回退到前一个前缀
32             prefixIndex = next[prefixIndex - 1];
33         } else {
34             // 如果前缀索引为0, 则移动到下一个元素
35             next[currentIndex] = 0;
36             currentIndex++;
37         }
38     }
39
40     // 获取next数组的最后一个值
41     int lastNextValue = next[arrayLength - 1];
42     // 计算子数组长度
43     int subArrayLength = arrayLength % (arrayLength - lastNextValue) == 0
        ? arrayLength - lastNextValue : arrayLength;
```

```

44
45     // 输出结果
46     for (int i = 0; i < subArrayLength; i++) {
47         printf("%d", nums[i]);
48         if (i < subArrayLength - 1) {
49             printf(" ");
50         }
51     }
52     printf("\n");
53
54     // 释放内存
55     free(nums);
56     free(next);
57
58     return 0;
59 }

```

完整用例

用例1

		Plain Text
1	9	
2	1 2 1 1 2 1 1 2 1	

用例2

		Plain Text
1	6	
2	1 2 3 1 2 3	

用例3

		Plain Text
1	8	
2	1 2 3 1 2 3 1 2	

用例4

▼	Plain Text
1 10	
2 1 2 3 1 2 3 1 2 3 1	

用例5

▼	Plain Text
1 5	
2 1 2 3 4 5	

用例6

▼	Plain Text
1 1	
2 1	

用例7

▼	Plain Text
1 6	
2 1 1 1 2 2 2	

用例8

▼	Plain Text
1 8	
2 1 2 1 1 2 1 1 2	

用例9

▼	Plain Text
1 8	
2 1 2 1 2 1 2 1 2	

用例10

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

文章目录

- [最新华为OD机试](#)
- [题目描述](#)
- [输入描述](#)
 - [备注](#)
- [输出描述](#)
- [示例1](#)
- [解题思路](#)
 - [问题理解](#)
 - [算法原理](#)
 - [KMP 的 next 数组在这里的应用](#)
 - [关键思路](#)
 - [示例分析](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [C++](#)
- [C语言](#)
 - [完整用例](#)
 - [用例1](#)
 - [用例2](#)
 - [用例3](#)
 - [用例4](#)
 - [用例5](#)
 - [用例6](#)
 - [用例7](#)
 - [用例8](#)
 - [用例9](#)
 - [用例10](#)

机考真题 华为OD



CSDN @算法大师

来自: [华为OD机考2025B卷 – 最小循环子数组 \(Java & Python& JS & C++ & C \)](#) -CSDN博客