

# t0720

---

[华为OD 机考 2025 C卷 - 几何平均值最大子数组 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025 C卷 - 热点网站统计 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD 机试 2025C卷 - 高矮个子排队 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025 C卷 - 精准核酸检测 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

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

# 华为OD 机考 2025 C卷 - 几何平均值最大子数组 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 几何平均值最大子数组

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

华为OD机试2025C卷 200分题型

### 题目描述

从一个长度为N的正数数组numbers中找出长度至少为L且几何平均值最大子数组，并输出其位置和大小。

(K个数的几何平均值为K个数的乘积的K次方根)

若有多个子数组的几何平均值均为**最大值**，则输出长度最小的子数组。

若有多个长度相同的子数组的几何平均值均为最大值，则输出最前面的子数组。

### 输入描述

第一行输入为N、L

- N表示numbers的大小 ( $1 \leq N \leq 100000$ )
- L表示子数组的最小长度 ( $1 \leq L \leq N$ )

之后N行表示numbers中的N个数，每个一行 ( $10^{-9} \leq \text{numbers}[i] \leq 10^9$ )

### 输出描述

输出子数组的位置（从0开始计数）和大小，中间用一个空格隔开。

### 备注

用例保证除几何平均值为最大值的子数组外，其他子数组的几何平均值至少比最大值小 $10^{-10}$ 倍

### 用例1

#### 输入

		Plain Text
1	3 2	
2	2	
3	2	
4	3	

#### 输出

		Plain Text
1	1 2	

## 说明

长度至少为2的子数组共三个，分别是{2,2}、{2,3}、{2,2,3}，其中{2,3}的几何平均值最大，故输出其位置1和长度2

## 用例2

### 输入

		Plain Text
1	10 2	
2	0.2	
3	0.1	
4	0.2	
5	0.2	
6	0.2	
7	0.1	
8	0.2	
9	0.2	
10	0.2	
11	0.2	

### 输出

		Plain Text
1	2 2	

## 说明

有多个长度至少为2的子数组的几何平均值为0.2，其中长度最短的为2，也有多个，长度为2且几何平均值为0.2的子数组最前面的那个为从第二个数开始的两个0.2组成的子数组

## 题解

思路： 数学原理 + 前缀和

- 考虑到几何平均值为  $(a_i * a_{i+1} \dots a_k)^{1/(k+1)}$  ,可以考虑使用对数将计算进行简化。首先需要明白这两个数学原理

- 1  $\log x + \log y = \log(x*y) \Rightarrow$  能够将乘法转换为加法
- 2  $1/n \log(x*y) = \log(x*y)^{1/n} \Rightarrow$  能够将方根计算转换为除法。

- 如果能想明白上面两个数学定理。将输入的数据全部转换为对数，然后构建前缀和数组。
- 枚举起点和终点求值，求其中 若有多个子数组的几何平均值均为最大值，则输出长度最小的子数组。 若有多组长度相同的子数组的几何平均值均为最大值，则输出最前面的子数组 要求结果即可。
- 额外需要注意浮点数计算会存在误差的问题，在进行选择结果时需要考虑在内。

**C++**

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <limits>
5
6  using namespace std;
7
8  // 判断是否更大
9  bool isBetter(double sum, int len, double best_sum, int best_len, int start, int best_start) {
10     //  $\text{sum}/\text{len} = \text{best\_sum} / \text{best\_len} \Rightarrow \text{sum} * \text{best\_len} = \text{best\_sum} * \text{len}$ 
11     double lhs = sum * best_len;
12     double rhs = best_sum * len;
13     // 考虑到误差
14     if (lhs > rhs + 1e-10) return true;
15     if (abs(lhs - rhs) <= 1e-10) {
16         if (len < best_len) return true;
17         if (len == best_len && start < best_start) return true;
18     }
19     return false;
20 }
21
22
23 int main() {
24     int N, L;
25     cin >> N >> L;
26     vector<double> nums(N);
27     for (int i = 0; i < N; ++i) {
28         double x;
29         cin >> x;
30         nums[i] = log(x); // 转换为对数, 几何平均值变为算术平均值
31     }
32
33     // 构建前缀和
34     vector<double> prefix(N + 1, 0.0);
35     for (int i = 0; i < N; ++i) {
36         prefix[i + 1] = prefix[i] + nums[i];
37     }
38
39     int best_start = 0;
40     int best_len = L;
41     double best_sum = prefix[L] - prefix[0];
42
43
44     // 枚举起点和终点
```

```

45     for (int start = 0; start < N; ++start) {
46         for (int end = start + L; end <= N; ++end) {
47             int len = end - start;
48             double sum = prefix[end] - prefix[start];
49
50             if (isBetter(sum, len, best_sum, best_len, start, best_start)) {
51                 best_start = start;
52                 best_len = len;
53                 best_sum = sum;
54             }
55         }
56     }
57
58     cout << best_start << " " << best_len << endl;
59     return 0;
60 }

```

## JAVA

```
1  import java.util.*;
2  import java.io.*;
3
4  public class Main {
5      static boolean isBetter(double sum, int len, double bestSum, int bestLen, int start, int bestStart) {
6          double lhs = sum * bestLen;
7          double rhs = bestSum * len;
8          if (lhs > rhs + 1e-10) return true;
9          // 考虑误差 1e-9 = 1e-10
10         if (Math.abs(lhs - rhs) <= 1e-10) {
11             if (len < bestLen) return true;
12             if (len == bestLen && start < bestStart) return true;
13         }
14         return false;
15     }
16
17     public static void main(String[] args) {
18         Scanner sc = new Scanner(System.in);
19         int N = sc.nextInt();
20         int L = sc.nextInt();
21         double[] nums = new double[N];
22         for (int i = 0; i < N; i++) {
23             double x = sc.nextDouble();
24             nums[i] = Math.log(x); // 转换为对数, 几何平均值变为算术平均值
25         }
26
27         double[] prefix = new double[N + 1];
28         for (int i = 0; i < N; i++) {
29             prefix[i + 1] = prefix[i] + nums[i];
30         }
31
32         int bestStart = 0;
33         int bestLen = L;
34         double bestSum = prefix[L] - prefix[0];
35
36         // 枚举起点和终点
37         for (int start = 0; start < N; start++) {
38             for (int end = start + L; end <= N; end++) {
39                 int len = end - start;
40                 double sum = prefix[end] - prefix[start];
41                 if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
42                     bestStart = start;
43                     bestLen = len;
```

```
44         bestSum = sum;
45     }
46 }
47 }
48 }
49     System.out.println(bestStart + " " + bestLen);
50 }
51 }
```

## Python



```
1  import sys
2  import math
3
4  # 判断最大值
5  def is_better(sum_, length, best_sum, best_length, start, best_start):
6      # 除法转换为乘法
7      lhs = sum_ * best_length
8      rhs = best_sum * length
9      # 考虑误差
10     if lhs > rhs + 1e-10:
11         return True
12
13     if abs(lhs - rhs) <= 1e-10:
14         if length < best_length:
15             return True
16         if length == best_length and start < best_start:
17             return True
18     return False
19
20 def main():
21     N, L = map(int, sys.stdin.readline().split())\
22     # 转换为对数
23     nums = [math.log(float(sys.stdin.readline())) for _ in range(N)]
24
25     prefix = [0.0] * (N + 1)
26     # 前缀和
27     for i in range(N):
28         prefix[i + 1] = prefix[i] + nums[i]
29
30     best_start = 0
31     best_len = L
32     best_sum = prefix[L] - prefix[0]
33     # 枚举终点起点
34     for start in range(N):
35         for end in range(start + L, N + 1):
36             length = end - start
37             sum_ = prefix[end] - prefix[start]
38             if is_better(sum_, length, best_sum, best_len, start, best_start):
39                 best_start = start
40                 best_len = length
41                 best_sum = sum_
42
43     print(f"{best_start} {best_len}")
44
```

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

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const inputLines = [];
9  rl.on('line', line => {
10    inputLines.push(line.trim());
11  });
12
13  rl.on('close', () => {
14    const [N, L] = inputLines[0].split(' ').map(Number);
15    const nums = inputLines.slice(1, N + 1).map(x => Math.log(parseFloat(x)));
16
17    // 构建前缀和
18    const prefix = new Array(N + 1).fill(0);
19    for (let i = 0; i < N; i++) {
20      prefix[i + 1] = prefix[i] + nums[i];
21    }
22
23    let bestStart = 0;
24    let bestLen = L;
25    let bestSum = prefix[L] - prefix[0];
26
27    // 判断是否更好
28    function isBetter(sum, len, bestSum, bestLen, start, bestStart) {
29      // 除法转换为乘法
30      const lhs = sum * bestLen;
31      const rhs = bestSum * len;
32      if (lhs > rhs + 1e-10) return true;
33      if (Math.abs(lhs - rhs) <= 1e-10) {
34        if (len < bestLen) return true;
35        if (len === bestLen && start < bestStart) return true;
36      }
37      return false;
38    }
39
40    // 枚举所有子数组
41    for (let start = 0; start < N; start++) {
42      for (let end = start + L; end <= N; end++) {
43        const len = end - start;
44        const sum = prefix[end] - prefix[start];
```

```
45         if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {  
46             bestStart = start;  
47             bestLen = len;  
48             bestSum = sum;  
49         }  
50     }  
51 }  
52  
53 console.log(`${bestStart} ${bestLen}`);  
54 });
```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "math"
7      "os"
8      "strconv"
9  )
10
11 func isBetter(sum float64, len int, bestSum float64, bestLen int, start in
t, bestStart int) bool {
12     // 除法转换乘法
13     lhs := sum * float64(bestLen)
14     rhs := bestSum * float64(len)
15     // 考虑误差
16     if lhs > rhs+1e-10 {
17         return true
18     }
19     if math.Abs(lhs-rhs) <= 1e-10 {
20         if len < bestLen {
21             return true
22         }
23         if len == bestLen && start < bestStart {
24             return true
25         }
26     }
27     return false
28 }
29
30 func main() {
31     reader := bufio.NewScanner(os.Stdin)
32     reader.Scan()
33     parts := splitInts(reader.Text())
34     N, L := parts[0], parts[1]
35
36     nums := make([]float64, N)
37     for i := 0; i < N; i++ {
38         reader.Scan()
39         f, _ := strconv.ParseFloat(reader.Text(), 64)
40         nums[i] = math.Log(f) // 转换为对数
41     }
42
43     prefix := make([]float64, N+1)
44     // 前缀和
```

```

45     for i := 0; i < N; i++ {
46         prefix[i+1] = prefix[i] + nums[i]
47     }
48
49     bestStart := 0
50     bestLen := L
51     bestSum := prefix[L] - prefix[0]
52     // 枚举起点和终点
53     for start := 0; start < N; start++ {
54         for end := start + L; end <= N; end++ {
55             length := end - start
56             sum := prefix[end] - prefix[start]
57             if isBetter(sum, length, bestSum, bestLen, start, bestStart) {
58                 bestStart = start
59                 bestLen = length
60                 bestSum = sum
61             }
62         }
63     }
64
65     fmt.Printf("%d %d\n", bestStart, bestLen)
66 }
67
68 func splitInts(s string) []int {
69     fields := make([]int, 0)
70     for _, part := range splitStr(s) {
71         v, _ := strconv.Atoi(part)
72         fields = append(fields, v)
73     }
74     return fields
75 }
76
77 func splitStr(s string) []string {
78     result := []string{}
79     current := ""
80     for _, ch := range s {
81         if ch == ' ' || ch == '\t' {
82             if current != "" {
83                 result = append(result, current)
84                 current = ""
85             }
86         } else {
87             current += string(ch)
88         }
89     }
90     if current != "" {
91         result = append(result, current)
92     }

```

```
93     return result
94 }
```

## 几何平均值最大子数组

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

华为OD机试2025C卷 200分题型

### 题目描述

从一个长度为N的正数数组numbers中找出长度至少为L且几何平均值最大子数组，并输出其位置和大小。

(K个数的几何平均值为K个数的乘积的K次方根)

若有多个子数组的几何平均值均为**最大值**，则输出长度最小的子数组。

若有多组长度相同的子数组的几何平均值均为最大值，则输出最前面的子数组。

### 输入描述

第一行输入为N、L

- N表示numbers的大小 ( $1 \leq N \leq 100000$ )
- L表示子数组的最小长度 ( $1 \leq L \leq N$ )

之后N行表示numbers中的N个数，每个一行 ( $10^{-9} \leq \text{numbers}[i] \leq 10^9$ )

### 输出描述

输出子数组的位置（从0开始计数）和大小，中间用一个空格隔开。

### 备注

用例保证除几何平均值为最大值的子数组外，其他子数组的几何平均值至少比最大值小 $10^{-10}$ 倍

### 用例1

#### 输入

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

Plain Text

#### 输出

	▼	Plain Text
1	1 2	

## 说明

长度至少为2的子数组共三个，分别是{2,2}、{2,3}、{2,2,3}，其中{2,3}的几何平均值最大，故输出其位置1和长度2

## 用例2

### 输入

	▼	Plain Text
1	10 2	
2	0.2	
3	0.1	
4	0.2	
5	0.2	
6	0.2	
7	0.1	
8	0.2	
9	0.2	
10	0.2	
11	0.2	

### 输出

	▼	Plain Text
1	2 2	

## 说明

有多个长度至少为2的子数组的几何平均值为0.2，其中长度最短的为2，也有多个，长度为2且几何平均值为0.2的子数组最前面的那个为从第二个数开始的两个0.2组成的子数组

## 题解

思路： 数学原理 + 前缀和

- 考虑到几何平均值为  $(a_i * a_{i+1} \dots a_k)^{1/(k+1)}$  ,可以考虑使用对数将计算进行简化。首先需要明白这两个数学原理



- 1  $\log x + \log y = \log(x*y) \Rightarrow$  能够将乘法转换为加法
- 2  $1/n \log(x*y) = \log(x*y)^{1/n} \Rightarrow$  能够将方根计算转换为除法。

- 如果能想明白上面两个数学定理。将输入的数据全部转换为对数，然后构建前缀和数组。
- 枚举起点和终点求值，求其中 若有多个子数组的几何平均值均为最大值，则输出长度最小的子数组。 若有多组长度相同的子数组的几何平均值均为最大值，则输出最前面的子数组 要求结果即可。
- 额外需要注意浮点数计算会存在误差的问题，在进行选择结果时需要考虑在内。

**C++**

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <limits>
5
6  using namespace std;
7
8  // 判断是否更大
9  bool isBetter(double sum, int len, double best_sum, int best_len, int start, int best_start) {
10     //  $\text{sum}/\text{len} = \text{best\_sum} / \text{best\_len} \Rightarrow \text{sum} * \text{best\_len} = \text{best\_sum} * \text{len}$ 
11     double lhs = sum * best_len;
12     double rhs = best_sum * len;
13     // 考虑到误差
14     if (lhs > rhs + 1e-10) return true;
15     if (abs(lhs - rhs) <= 1e-10) {
16         if (len < best_len) return true;
17         if (len == best_len && start < best_start) return true;
18     }
19     return false;
20 }
21
22
23 int main() {
24     int N, L;
25     cin >> N >> L;
26     vector<double> nums(N);
27     for (int i = 0; i < N; ++i) {
28         double x;
29         cin >> x;
30         nums[i] = log(x); // 转换为对数, 几何平均值变为算术平均值
31     }
32
33     // 构建前缀和
34     vector<double> prefix(N + 1, 0.0);
35     for (int i = 0; i < N; ++i) {
36         prefix[i + 1] = prefix[i] + nums[i];
37     }
38
39     int best_start = 0;
40     int best_len = L;
41     double best_sum = prefix[L] - prefix[0];
42
43
44     // 枚举起点和终点
```

```

45     for (int start = 0; start < N; ++start) {
46         for (int end = start + L; end <= N; ++end) {
47             int len = end - start;
48             double sum = prefix[end] - prefix[start];
49
50             if (isBetter(sum, len, best_sum, best_len, start, best_start)) {
51                 best_start = start;
52                 best_len = len;
53                 best_sum = sum;
54             }
55         }
56     }
57
58     cout << best_start << " " << best_len << endl;
59     return 0;
60 }

```

## JAVA

```
1  import java.util.*;
2  import java.io.*;
3
4  public class Main {
5      static boolean isBetter(double sum, int len, double bestSum, int bestLen, int start, int bestStart) {
6          double lhs = sum * bestLen;
7          double rhs = bestSum * len;
8          if (lhs > rhs + 1e-10) return true;
9          // 考虑误差 1e-9 = 1e-10
10         if (Math.abs(lhs - rhs) <= 1e-10) {
11             if (len < bestLen) return true;
12             if (len == bestLen && start < bestStart) return true;
13         }
14         return false;
15     }
16
17     public static void main(String[] args) {
18         Scanner sc = new Scanner(System.in);
19         int N = sc.nextInt();
20         int L = sc.nextInt();
21         double[] nums = new double[N];
22         for (int i = 0; i < N; i++) {
23             double x = sc.nextDouble();
24             nums[i] = Math.log(x); // 转换为对数, 几何平均值变为算术平均值
25         }
26
27         double[] prefix = new double[N + 1];
28         for (int i = 0; i < N; i++) {
29             prefix[i + 1] = prefix[i] + nums[i];
30         }
31
32         int bestStart = 0;
33         int bestLen = L;
34         double bestSum = prefix[L] - prefix[0];
35
36         // 枚举起点和终点
37         for (int start = 0; start < N; start++) {
38             for (int end = start + L; end <= N; end++) {
39                 int len = end - start;
40                 double sum = prefix[end] - prefix[start];
41                 if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
42                     bestStart = start;
43                     bestLen = len;
```

```
44         bestSum = sum;
45     }
46 }
47 }
48 }
49     System.out.println(bestStart + " " + bestLen);
50 }
51 }
```

## Python

```
1  import sys
2  import math
3
4  # 判断最大值
5  def is_better(sum_, length, best_sum, best_length, start, best_start):
6      # 除法转换为乘法
7      lhs = sum_ * best_length
8      rhs = best_sum * length
9      # 考虑误差
10     if lhs > rhs + 1e-10:
11         return True
12
13     if abs(lhs - rhs) <= 1e-10:
14         if length < best_length:
15             return True
16         if length == best_length and start < best_start:
17             return True
18     return False
19
20 def main():
21     N, L = map(int, sys.stdin.readline().split())\
22     # 转换为对数
23     nums = [math.log(float(sys.stdin.readline())) for _ in range(N)]
24
25     prefix = [0.0] * (N + 1)
26     # 前缀和
27     for i in range(N):
28         prefix[i + 1] = prefix[i] + nums[i]
29
30     best_start = 0
31     best_len = L
32     best_sum = prefix[L] - prefix[0]
33     # 枚举终点起点
34     for start in range(N):
35         for end in range(start + L, N + 1):
36             length = end - start
37             sum_ = prefix[end] - prefix[start]
38             if is_better(sum_, length, best_sum, best_len, start, best_start):
39                 best_start = start
40                 best_len = length
41                 best_sum = sum_
42
43     print(f"{best_start} {best_len}")
44
```

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

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const inputLines = [];
9  rl.on('line', line => {
10    inputLines.push(line.trim());
11  });
12
13  rl.on('close', () => {
14    const [N, L] = inputLines[0].split(' ').map(Number);
15    const nums = inputLines.slice(1, N + 1).map(x => Math.log(parseFloat(x)));
16
17    // 构建前缀和
18    const prefix = new Array(N + 1).fill(0);
19    for (let i = 0; i < N; i++) {
20      prefix[i + 1] = prefix[i] + nums[i];
21    }
22
23    let bestStart = 0;
24    let bestLen = L;
25    let bestSum = prefix[L] - prefix[0];
26
27    // 判断是否更好
28    function isBetter(sum, len, bestSum, bestLen, start, bestStart) {
29      // 除法转换为乘法
30      const lhs = sum * bestLen;
31      const rhs = bestSum * len;
32      if (lhs > rhs + 1e-10) return true;
33      if (Math.abs(lhs - rhs) <= 1e-10) {
34        if (len < bestLen) return true;
35        if (len === bestLen && start < bestStart) return true;
36      }
37      return false;
38    }
39
40    // 枚举所有子数组
41    for (let start = 0; start < N; start++) {
42      for (let end = start + L; end <= N; end++) {
43        const len = end - start;
44        const sum = prefix[end] - prefix[start];
```



```
45         if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
46             bestStart = start;
47             bestLen = len;
48             bestSum = sum;
49         }
50     }
51 }
52
53 console.log(`${bestStart} ${bestLen}`);
54 });
```

**Go**

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "math"
7      "os"
8      "strconv"
9  )
10
11 func isBetter(sum float64, len int, bestSum float64, bestLen int, start in
t, bestStart int) bool {
12     // 除法转换乘法
13     lhs := sum * float64(bestLen)
14     rhs := bestSum * float64(len)
15     // 考虑误差
16     if lhs > rhs+1e-10 {
17         return true
18     }
19     if math.Abs(lhs-rhs) <= 1e-10 {
20         if len < bestLen {
21             return true
22         }
23         if len == bestLen && start < bestStart {
24             return true
25         }
26     }
27     return false
28 }
29
30 func main() {
31     reader := bufio.NewScanner(os.Stdin)
32     reader.Scan()
33     parts := splitInts(reader.Text())
34     N, L := parts[0], parts[1]
35
36     nums := make([]float64, N)
37     for i := 0; i < N; i++ {
38         reader.Scan()
39         f, _ := strconv.ParseFloat(reader.Text(), 64)
40         nums[i] = math.Log(f) // 转换为对数
41     }
42
43     prefix := make([]float64, N+1)
44     // 前缀和
```

```

45     for i := 0; i < N; i++ {
46         prefix[i+1] = prefix[i] + nums[i]
47     }
48
49     bestStart := 0
50     bestLen := L
51     bestSum := prefix[L] - prefix[0]
52     // 枚举起点和终点
53     for start := 0; start < N; start++ {
54         for end := start + L; end <= N; end++ {
55             length := end - start
56             sum := prefix[end] - prefix[start]
57             if isBetter(sum, length, bestSum, bestLen, start, bestStart) {
58                 bestStart = start
59                 bestLen = length
60                 bestSum = sum
61             }
62         }
63     }
64
65     fmt.Printf("%d %d\n", bestStart, bestLen)
66 }
67
68 func splitInts(s string) []int {
69     fields := make([]int, 0)
70     for _, part := range splitStr(s) {
71         v, _ := strconv.Atoi(part)
72         fields = append(fields, v)
73     }
74     return fields
75 }
76
77 func splitStr(s string) []string {
78     result := []string{}
79     current := ""
80     for _, ch := range s {
81         if ch == ' ' || ch == '\t' {
82             if current != "" {
83                 result = append(result, current)
84                 current = ""
85             }
86         } else {
87             current += string(ch)
88         }
89     }
90     if current != "" {
91         result = append(result, current)
92     }

```

```
93     return result
94 }
```

来自: [华为OD 机考 2025 C卷 – 几何平均值最大子数组 \(C++ & Python & JAVA & JS & GO\)–CSDN 博客](#)

## 几何平均值最大子数组

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

华为OD机试2025C卷 200分题型

### 题目描述

从一个长度为N的正数数组numbers中找出长度至少为L且几何平均值最大子数组，并输出其位置和大小。

(K个数的几何平均值为K个数的乘积的K次方根)

若有多个子数组的几何平均值均为**最大值**，则输出长度最小的子数组。

若有多个长度相同的子数组的几何平均值均为最大值，则输出最前面的子数组。

### 输入描述

第一行输入为N、L

- N表示numbers的大小 ( $1 \leq N \leq 100000$ )
- L表示子数组的最小长度 ( $1 \leq L \leq N$ )

之后N行表示numbers中的N个数，每个一行 ( $10^{-9} \leq \text{numbers}[i] \leq 10^9$ )

### 输出描述

输出子数组的位置 (从0开始计数) 和大小，中间用一个空格隔开。

### 备注

用例保证除几何平均值为最大值的子数组外，其他子数组的几何平均值至少比最大值小 $10^{-10}$ 倍

### 用例1

#### 输入

▼ Plain Text

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

输出

	Plain Text
1	1 2

说明

长度至少为2的子数组共三个，分别是{2,2}、{2,3}、{2,2,3}，其中{2,3}的几何平均值最大，故输出其位置1和长度2

用例2

输入

	Plain Text
1	10 2
2	0.2
3	0.1
4	0.2
5	0.2
6	0.2
7	0.1
8	0.2
9	0.2
10	0.2
11	0.2

输出

	Plain Text
1	2 2

说明

有多个长度至少为2的子数组的几何平均值为0.2，其中长度最短的为2，也有多个，长度为2且几何平均值为0.2的子数组最前面的那个为从第二个数开始的两个0.2组成的子数组

题解

思路： 数学原理 + 前缀和

- 考虑到几何平均值为  $(a_i \times a_{i+1} \dots + a_k)^{1/(k+1)}$  ,可以考虑使用对数将计算进行简化。首先需要

## 明白这两个数学原理

Plain Text

- 1  $\log x + \log y = \log(x*y) \Rightarrow$  能够将乘法转换为加法
- 2  $1/n \log(x*y) = \log(x*y)^{1/n} \Rightarrow$  能够将方根计算转换为除法。

- 如果能想明白上面两个数学定理。将输入的数据全部转换为对数，然后构建前缀和数组。
- 枚举起点和终点求值，求其中 若有多个子数组的几何平均值均为最大值，则输出长度最小的子数组。 若有多组长度相同的子数组的几何平均值均为最大值，则输出最前面的子数组 要求结果即可。
- 额外需要注意浮点数计算会存在误差的问题，在进行选择结果时需要考虑在内。

**C++**

```
1  #include <iostream>
2  #include <vector>
3  #include <cmath>
4  #include <limits>
5
6  using namespace std;
7
8  // 判断是否更大
9  bool isBetter(double sum, int len, double best_sum, int best_len, int start, int best_start) {
10     //  $\text{sum}/\text{len} = \text{best\_sum} / \text{best\_len} \Rightarrow \text{sum} * \text{best\_len} = \text{best\_sum} * \text{len}$ 
11     double lhs = sum * best_len;
12     double rhs = best_sum * len;
13     // 考虑到误差
14     if (lhs > rhs + 1e-10) return true;
15     if (abs(lhs - rhs) <= 1e-10) {
16         if (len < best_len) return true;
17         if (len == best_len && start < best_start) return true;
18     }
19     return false;
20 }
21
22
23 int main() {
24     int N, L;
25     cin >> N >> L;
26     vector<double> nums(N);
27     for (int i = 0; i < N; ++i) {
28         double x;
29         cin >> x;
30         nums[i] = log(x); // 转换为对数, 几何平均值变为算术平均值
31     }
32
33     // 构建前缀和
34     vector<double> prefix(N + 1, 0.0);
35     for (int i = 0; i < N; ++i) {
36         prefix[i + 1] = prefix[i] + nums[i];
37     }
38
39     int best_start = 0;
40     int best_len = L;
41     double best_sum = prefix[L] - prefix[0];
42
43
44     // 枚举起点和终点
```

```

45     for (int start = 0; start < N; ++start) {
46         for (int end = start + L; end <= N; ++end) {
47             int len = end - start;
48             double sum = prefix[end] - prefix[start];
49
50             if (isBetter(sum, len, best_sum, best_len, start, best_start)) {
51                 best_start = start;
52                 best_len = len;
53                 best_sum = sum;
54             }
55         }
56     }
57
58     cout << best_start << " " << best_len << endl;
59     return 0;
60 }

```

## JAVA



```
1  import java.util.*;
2  import java.io.*;
3
4  public class Main {
5      static boolean isBetter(double sum, int len, double bestSum, int bestLen, int start, int bestStart) {
6          double lhs = sum * bestLen;
7          double rhs = bestSum * len;
8          if (lhs > rhs + 1e-10) return true;
9          // 考虑误差 1e-9 = 1e-10
10         if (Math.abs(lhs - rhs) <= 1e-10) {
11             if (len < bestLen) return true;
12             if (len == bestLen && start < bestStart) return true;
13         }
14         return false;
15     }
16
17     public static void main(String[] args) {
18         Scanner sc = new Scanner(System.in);
19         int N = sc.nextInt();
20         int L = sc.nextInt();
21         double[] nums = new double[N];
22         for (int i = 0; i < N; i++) {
23             double x = sc.nextDouble();
24             nums[i] = Math.log(x); // 转换为对数, 几何平均值变为算术平均值
25         }
26
27         double[] prefix = new double[N + 1];
28         for (int i = 0; i < N; i++) {
29             prefix[i + 1] = prefix[i] + nums[i];
30         }
31
32         int bestStart = 0;
33         int bestLen = L;
34         double bestSum = prefix[L] - prefix[0];
35
36         // 枚举起点和终点
37         for (int start = 0; start < N; start++) {
38             for (int end = start + L; end <= N; end++) {
39                 int len = end - start;
40                 double sum = prefix[end] - prefix[start];
41                 if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
42                     bestStart = start;
43                     bestLen = len;
```

```
44         bestSum = sum;
45     }
46 }
47 }
48 }
49 System.out.println(bestStart + " " + bestLen);
50 }
51 }
```

## Python

```
1  import sys
2  import math
3
4  # 判断最大值
5  def is_better(sum_, length, best_sum, best_length, start, best_start):
6      # 除法转换为乘法
7      lhs = sum_ * best_length
8      rhs = best_sum * length
9      # 考虑误差
10     if lhs > rhs + 1e-10:
11         return True
12
13     if abs(lhs - rhs) <= 1e-10:
14         if length < best_length:
15             return True
16         if length == best_length and start < best_start:
17             return True
18     return False
19
20 def main():
21     N, L = map(int, sys.stdin.readline().split())\
22     # 转换为对数
23     nums = [math.log(float(sys.stdin.readline())) for _ in range(N)]
24
25     prefix = [0.0] * (N + 1)
26     # 前缀和
27     for i in range(N):
28         prefix[i + 1] = prefix[i] + nums[i]
29
30     best_start = 0
31     best_len = L
32     best_sum = prefix[L] - prefix[0]
33     # 枚举终点起点
34     for start in range(N):
35         for end in range(start + L, N + 1):
36             length = end - start
37             sum_ = prefix[end] - prefix[start]
38             if is_better(sum_, length, best_sum, best_len, start, best_start):
39                 best_start = start
40                 best_len = length
41                 best_sum = sum_
42
43     print(f"{best_start} {best_len}")
44
```

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

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const inputLines = [];
9  rl.on('line', line => {
10    inputLines.push(line.trim());
11  });
12
13  rl.on('close', () => {
14    const [N, L] = inputLines[0].split(' ').map(Number);
15    const nums = inputLines.slice(1, N + 1).map(x => Math.log(parseFloat(x)));
16
17    // 构建前缀和
18    const prefix = new Array(N + 1).fill(0);
19    for (let i = 0; i < N; i++) {
20      prefix[i + 1] = prefix[i] + nums[i];
21    }
22
23    let bestStart = 0;
24    let bestLen = L;
25    let bestSum = prefix[L] - prefix[0];
26
27    // 判断是否更好
28    function isBetter(sum, len, bestSum, bestLen, start, bestStart) {
29      // 除法转换为乘法
30      const lhs = sum * bestLen;
31      const rhs = bestSum * len;
32      if (lhs > rhs + 1e-10) return true;
33      if (Math.abs(lhs - rhs) <= 1e-10) {
34        if (len < bestLen) return true;
35        if (len === bestLen && start < bestStart) return true;
36      }
37      return false;
38    }
39
40    // 枚举所有子数组
41    for (let start = 0; start < N; start++) {
42      for (let end = start + L; end <= N; end++) {
43        const len = end - start;
44        const sum = prefix[end] - prefix[start];
```

```
45         if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {  
46             bestStart = start;  
47             bestLen = len;  
48             bestSum = sum;  
49         }  
50     }  
51 }  
52  
53 console.log(`${bestStart} ${bestLen}`);  
54 });
```

**Go**

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "math"
7     "os"
8     "strconv"
9 )
10
11 func isBetter(sum float64, len int, bestSum float64, bestLen int, start in
t, bestStart int) bool {
12     // 除法转换乘法
13     lhs := sum * float64(bestLen)
14     rhs := bestSum * float64(len)
15     // 考虑误差
16     if lhs > rhs+1e-10 {
17         return true
18     }
19     if math.Abs(lhs-rhs) <= 1e-10 {
20         if len < bestLen {
21             return true
22         }
23         if len == bestLen && start < bestStart {
24             return true
25         }
26     }
27     return false
28 }
29
30 func main() {
31     reader := bufio.NewScanner(os.Stdin)
32     reader.Scan()
33     parts := splitInts(reader.Text())
34     N, L := parts[0], parts[1]
35
36     nums := make([]float64, N)
37     for i := 0; i < N; i++ {
38         reader.Scan()
39         f, _ := strconv.ParseFloat(reader.Text(), 64)
40         nums[i] = math.Log(f) // 转换为对数
41     }
42
43     prefix := make([]float64, N+1)
44     // 前缀和
```

```

45     for i := 0; i < N; i++ {
46         prefix[i+1] = prefix[i] + nums[i]
47     }
48
49     bestStart := 0
50     bestLen := L
51     bestSum := prefix[L] - prefix[0]
52     // 枚举起点和终点
53     for start := 0; start < N; start++ {
54         for end := start + L; end <= N; end++ {
55             length := end - start
56             sum := prefix[end] - prefix[start]
57             if isBetter(sum, length, bestSum, bestLen, start, bestStart) {
58                 bestStart = start
59                 bestLen = length
60                 bestSum = sum
61             }
62         }
63     }
64
65     fmt.Printf("%d %d\n", bestStart, bestLen)
66 }
67
68 func splitInts(s string) []int {
69     fields := make([]int, 0)
70     for _, part := range splitStr(s) {
71         v, _ := strconv.Atoi(part)
72         fields = append(fields, v)
73     }
74     return fields
75 }
76
77 func splitStr(s string) []string {
78     result := []string{}
79     current := ""
80     for _, ch := range s {
81         if ch == ' ' || ch == '\t' {
82             if current != "" {
83                 result = append(result, current)
84                 current = ""
85             }
86         } else {
87             current += string(ch)
88         }
89     }
90     if current != "" {
91         result = append(result, current)
92     }

```



```
93     return result
94 }
```

来自: [华为OD 机考 2025 C卷 – 几何平均值最大子数组 \(C++ & Python & JAVA & JS & GO\)–CSDN 博客](#)

来自: [华为OD 机考 2025 C卷 – 几何平均值最大子数组 \(C++ & Python & JAVA & JS & GO\)–CSDN 博客](#)

# 华为OD机试2025 C卷 - 热点网站统计 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 热点网站统计

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

华为OD机试2025C卷 100分题型

## 题目描述

企业路由器的统计页面，有一个功能需要动态统计公司访问最多的网页URL top N。请设计一个算法，可以高效动态统计Top N的页面。

## 输入描述

每一行都是一个URL或一个数字，如果是URL，代表一段时间内的网页访问； 如果是一个数字N，代表本次需要输出的Top N个URL。

输入约束：

1. 总访问网页数量小于5000个，单网页访问次数小于65535次；
2. 网页URL仅由字母，数字和点分隔符组成，且长度小于等于127字节；
3. 数字是正整数，小于等于10且小于当前总访问网页数；

## 输出描述

行输入要对应一行输出，输出按访问次数排序的前N个URL，用逗号分隔。

输出要求：

1. 每次输出要统计之前所有输入，不仅是本次输入；
2. 如果有访问次数相等的URL，按URL的字符串字典序升序排列，输出排序靠前的URL；

## 示例1

### 输入

	▼	Plain Text
1	news.qq.com	
2	news.sina.com.cn	
3	news.qq.com	
4	news.qq.com	
5	game.163.com	
6	game.163.com	
7	www.huawei.com	
8	www.cctv.com	
9	3	
10	www.huawei.com	
11	www.cctv.com	
12	www.huawei.com	
13	www.cctv.com	
14	www.huawei.com	
15	www.cctv.com	
16	www.huawei.com	
17	www.cctv.com	
18	www.huawei.com	
19	3	

## 输出

	▼	Plain Text
1	news.qq.com,game.163.com,news.sina.com.cn	
2	www.huawei.com,www.cctv.com,news.qq.com	

## 示例2

### 输入

	▼	Plain Text
1	news.qq.com	
2	www.cctv.com	
3	1	
4	www.huawei.com	
5	www.huawei.com	
6	2	
7	3	

## 输出

▼ Plain Text

```
1 news.qq.com
2 www.huawei.com,news.qq.com
3 www.huawei.com,news.qq.com,www.cctv.com
```

## 题解

思路： 模拟 ,由于本题的数据量比较小，可以按照下面这种代码通过。当数据量较大时可以进行使用 优先队列 排序。

- 使用哈希表统计出现的网站的次数。
- 当遇到数字时，进行自定义排序，对出现网站进行排序先按照次数排序升序，当出现次数相同时按照字典序升序。
- 输出前N个网站即可。

**C++**

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <algorithm>
5  #include <regex>
6
7  using namespace std;
8
9  string sortURL(int n, map<string, int>& cache);
10
11 int main() {
12     map<string, int> cache; // 创建一个哈希表, 用于存储每个 URL 出现的次数
13     string line;
14     while (getline(cin, line)) { // 不断读取输入, 直到没有下一行
15         if (regex_match(line, regex("^\\d+$"))) { // 如果该行数据只包含数字,
说明已经读取完了一个测试用例
16             cout << sortURL(stoi(line), cache) << endl;
17             continue;
18         }
19         cache[line]++;
20     }
21     return 0;
22 }
23
24 string sortURL(int n, map<string, int>& cache) {
25     vector<pair<string, int>> list(cache.begin(), cache.end()); // 将哈希表
中的每一项转换成一个键值对, 并存入一个列表中
26
27     sort(list.begin(), list.end(), [](const pair<string, int>& a, const pa
ir<string, int>& b) { // 对列表进行排序, 按照计数从大到小排序, 如果计数相同则按照字
典序从小到大排序
28         if (a.second != b.second) {
29             return a.second > b.second;
30         } else {
31             return a.first < b.first;
32         }
33     });
34
35     string res;
36     for (int i = 0; i < n && i < list.size(); i++) { // 取出前 n 个 URL, 并
将它们拼接成一个字符串
37         res += list[i].first + ",";
38     }
39     if (!res.empty()) { // 如果字符串不为空, 则删除最后一个逗号
40         res.pop_back();
```

```
41     }  
42     return res; // 返回拼接好的字符串  
43 }
```

## Java

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          Map<String, Integer> cache = new HashMap<>(); // 哈希表存储 URL 及其
出现次数
7
8          while (scanner.hasNextLine()) {
9              String line = scanner.nextLine().trim();
10             if (line.matches("^\\d+$")) { // 如果输入是纯数字，处理排序并输出结
果
11                 System.out.println(sortURL(Integer.parseInt(line), cach
e));
12                 continue;
13             }
14             cache.put(line, cache.getOrDefault(line, 0) + 1);
15         }
16         scanner.close();
17     }
18
19     private static String sortURL(int n, Map<String, Integer> cache) {
20         List<Map.Entry<String, Integer>> list = new ArrayList<>(cache.entr
ySet());
21
22         // 按出现次数降序排列，次数相同则按字典序升序排列
23         list.sort((a, b) -> {
24             if (!a.getValue().equals(b.getValue())) {
25                 return b.getValue() - a.getValue();
26             }
27             return a.getKey().compareTo(b.getKey());
28         });
29
30         // 取前 n 个 URL 并拼接成字符串
31         StringBuilder res = new StringBuilder();
32         for (int i = 0; i < Math.min(n, list.size()); i++) {
33             res.append(list.get(i).getKey()).append(",");
34         }
35         if (res.length() > 0) {
36             res.setLength(res.length() - 1); // 删除末尾的逗号
37         }
38         return res.toString();
39     }
40 }
```

## Python

```
▼ Plain Text |
1  import sys
2  import re
3
4  def sort_url(n, cache):
5      # 按出现次数降序排序, 次数相同则按字典序升序
6      sorted_urls = sorted(cache.items(), key=lambda x: (-x[1], x[0]))
7
8      # 取前 n 个 URL 并拼接成字符串
9      return ",".join(url for url, _ in sorted_urls[:n])
10
11 def main():
12     cache = {} # 哈希表存储 URL 及其出现次数
13
14     for line in sys.stdin:
15         line = line.strip()
16         if re.fullmatch(r"\d+", line): # 如果输入是纯数字, 处理排序并输出结果
17             print(sort_url(int(line), cache))
18             continue
19         cache[line] = cache.get(line, 0) + 1
20
21 if __name__ == "__main__":
22     main()
```

## JavaScript



```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let cache = {}; // 哈希表存储 URL 及其出现次数
9  let lines = [];
10
11  rl.on("line", (line) => {
12    line = line.trim();
13    if (/^\d+$/ .test(line)) { // 如果输入是纯数字，处理排序并输出结果
14      console.log(sortURL(parseInt(line), cache));
15      return;
16    }
17    cache[line] = (cache[line] || 0) + 1;
18  });
19
20  function sortURL(n, cache) {
21    // 转换为数组并排序，按次数降序，次数相同则按字典序升序
22    let sortedList = Object.entries(cache).sort((a, b) => {
23      if (b[1] !== a[1]) {
24        return b[1] - a[1];
25      }
26      return a[0].localeCompare(b[0]);
27    });
28
29    // 取前 n 个 URL 并拼接成字符串
30    return sortedList.slice(0, n).map(item => item[0]).join(",");
31  }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12 func sortURL(n int, cache map[string]int) string {
13     // 将哈希表转换为切片
14     type urlCount struct {
15         url    string
16         count  int
17     }
18     list := make([]urlCount, 0, len(cache))
19     for url, count := range cache {
20         list = append(list, urlCount{url, count})
21     }
22
23     // 按出现次数降序, 次数相同则按字典序升序
24     sort.Slice(list, func(i, j int) bool {
25         if list[i].count != list[j].count {
26             return list[i].count > list[j].count
27         }
28         return list[i].url < list[j].url
29     })
30
31     // 取前 n 个 URL 并拼接成字符串
32     var res []string
33     for i := 0; i < n && i < len(list); i++ {
34         res = append(res, list[i].url)
35     }
36     return strings.Join(res, ",")
37 }
38
39 func main() {
40     cache := make(map[string]int) // 哈希表存储 URL 及其出现次数
41     scanner := bufio.NewScanner(os.Stdin)
42
43     for scanner.Scan() {
44         line := strings.TrimSpace(scanner.Text())
45     }
```

```

46         if num, err := strconv.Atoi(line); err == nil { // 如果输入是纯数字，处理
47             排序并输出结果
48                 fmt.Println(sortURL(num, cache))
49                 continue
50             }
51             cache[line]++
52         }

```

## 热点网站统计

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

华为OD机试2025C卷 100分题型

## 题目描述

企业路由器的统计页面，有一个功能需要动态统计公司访问最多的网页URL top N。请设计一个算法，可以高效动态统计Top N的页面。

## 输入描述

每一行都是一个URL或一个数字，如果是URL，代表一段时间内的网页访问； 如果是一个数字N，代表本次需要输出的Top N个URL。

输入约束：

1. 总访问网页数量小于5000个，单网页访问次数小于65535次；
2. 网页URL仅由字母，数字和点分隔符组成，且长度小于等于127字节； 3、数字是正整数，小于等于10且小于当前总访问网页数；

## 输出描述

行输入要对应一行输出，输出按访问次数排序的前N个URL，用逗号分隔。

输出要求：

1. 每次输出要统计之前所有输入，不仅是本次输入；
2. 如果有访问次数相等的URL，按URL的字符串字典序升序排列，输出排序靠前的URL；

## 示例1

### 输入

```
1 news.qq.com
2 news.sina.com.cn
3 news.qq.com
4 news.qq.com
5 game.163.com
6 game.163.com
7 www.huawei.com
8 www.cctv.com
9 3
10 www.huawei.com
11 www.cctv.com
12 www.huawei.com
13 www.cctv.com
14 www.huawei.com
15 www.cctv.com
16 www.huawei.com
17 www.cctv.com
18 www.huawei.com
19 3
```

## 输出

```
1 news.qq.com,game.163.com,news.sina.com.cn
2 www.huawei.com,www.cctv.com,news.qq.com
```

## 示例2

### 输入

```
1 news.qq.com
2 www.cctv.com
3 1
4 www.huawei.com
5 www.huawei.com
6 2
7 3
```

## 输出

▼ Plain Text

```
1 news.qq.com
2 www.huawei.com,news.qq.com
3 www.huawei.com,news.qq.com,www.cctv.com
```

## 题解

思路： 模拟 ,由于本题的数据量比较小，可以按照下面这种代码通过。当数据量较大时可以进行使用 优先队列 排序。

- 使用哈希表统计出现的网站的次数。
- 当遇到数字时，进行自定义排序，对出现网站进行排序先按照次数排序升序，当出现次数相同时按照字典序升序。
- 输出前N个网站即可。

**C++**

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <algorithm>
5  #include <regex>
6
7  using namespace std;
8
9  string sortURL(int n, map<string, int>& cache);
10
11 int main() {
12     map<string, int> cache; // 创建一个哈希表, 用于存储每个 URL 出现的次数
13     string line;
14     while (getline(cin, line)) { // 不断读取输入, 直到没有下一行
15         if (regex_match(line, regex("^\\d+$"))) { // 如果该行数据只包含数字,
说明已经读取完了一个测试用例
16             cout << sortURL(stoi(line), cache) << endl;
17             continue;
18         }
19         cache[line]++;
20     }
21     return 0;
22 }
23
24 string sortURL(int n, map<string, int>& cache) {
25     vector<pair<string, int>> list(cache.begin(), cache.end()); // 将哈希表
中的每一项转换成一个键值对, 并存入一个列表中
26
27     sort(list.begin(), list.end(), [](const pair<string, int>& a, const pa
ir<string, int>& b) { // 对列表进行排序, 按照计数从大到小排序, 如果计数相同则按照字
典序从小到大排序
28         if (a.second != b.second) {
29             return a.second > b.second;
30         } else {
31             return a.first < b.first;
32         }
33     });
34
35     string res;
36     for (int i = 0; i < n && i < list.size(); i++) { // 取出前 n 个 URL, 并
将它们拼接成一个字符串
37         res += list[i].first + ",";
38     }
39     if (!res.empty()) { // 如果字符串不为空, 则删除最后一个逗号
40         res.pop_back();
```

```
41     }  
42     return res; // 返回拼接好的字符串  
43 }
```

## Java

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          Map<String, Integer> cache = new HashMap<>(); // 哈希表存储 URL 及其
出现次数
7
8          while (scanner.hasNextLine()) {
9              String line = scanner.nextLine().trim();
10             if (line.matches("^\\d+$")) { // 如果输入是纯数字，处理排序并输出结
果
11                 System.out.println(sortURL(Integer.parseInt(line), cach
e));
12                 continue;
13             }
14             cache.put(line, cache.getOrDefault(line, 0) + 1);
15         }
16         scanner.close();
17     }
18
19     private static String sortURL(int n, Map<String, Integer> cache) {
20         List<Map.Entry<String, Integer>> list = new ArrayList<>(cache.entr
ySet());
21
22         // 按出现次数降序排列，次数相同则按字典序升序排列
23         list.sort((a, b) -> {
24             if (!a.getValue().equals(b.getValue())) {
25                 return b.getValue() - a.getValue();
26             }
27             return a.getKey().compareTo(b.getKey());
28         });
29
30         // 取前 n 个 URL 并拼接成字符串
31         StringBuilder res = new StringBuilder();
32         for (int i = 0; i < Math.min(n, list.size()); i++) {
33             res.append(list.get(i).getKey()).append(",");
34         }
35         if (res.length() > 0) {
36             res.setLength(res.length() - 1); // 删除末尾的逗号
37         }
38         return res.toString();
39     }
40 }
```



## Python

```
▼ Plain Text |
1  import sys
2  import re
3
4  def sort_url(n, cache):
5      # 按出现次数降序排序, 次数相同则按字典序升序
6      sorted_urls = sorted(cache.items(), key=lambda x: (-x[1], x[0]))
7
8      # 取前 n 个 URL 并拼接成字符串
9      return ",".join(url for url, _ in sorted_urls[:n])
10
11 def main():
12     cache = {} # 哈希表存储 URL 及其出现次数
13
14     for line in sys.stdin:
15         line = line.strip()
16         if re.fullmatch(r"\d+", line): # 如果输入是纯数字, 处理排序并输出结果
17             print(sort_url(int(line), cache))
18             continue
19         cache[line] = cache.get(line, 0) + 1
20
21 if __name__ == "__main__":
22     main()
```

## JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let cache = {}; // 哈希表存储 URL 及其出现次数
9  let lines = [];
10
11  rl.on("line", (line) => {
12      line = line.trim();
13      if (/^\d+$/ .test(line)) { // 如果输入是纯数字，处理排序并输出结果
14          console.log(sortURL(parseInt(line), cache));
15          return;
16      }
17      cache[line] = (cache[line] || 0) + 1;
18  });
19
20  function sortURL(n, cache) {
21      // 转换为数组并排序，按次数降序，次数相同则按字典序升序
22      let sortedList = Object.entries(cache).sort((a, b) => {
23          if (b[1] !== a[1]) {
24              return b[1] - a[1];
25          }
26          return a[0].localeCompare(b[0]);
27      });
28
29      // 取前 n 个 URL 并拼接成字符串
30      return sortedList.slice(0, n).map(item => item[0]).join(",");
31  }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12 func sortURL(n int, cache map[string]int) string {
13     // 将哈希表转换为切片
14     type urlCount struct {
15         url    string
16         count  int
17     }
18     list := make([]urlCount, 0, len(cache))
19     for url, count := range cache {
20         list = append(list, urlCount{url, count})
21     }
22
23     // 按出现次数降序, 次数相同则按字典序升序
24     sort.Slice(list, func(i, j int) bool {
25         if list[i].count != list[j].count {
26             return list[i].count > list[j].count
27         }
28         return list[i].url < list[j].url
29     })
30
31     // 取前 n 个 URL 并拼接成字符串
32     var res []string
33     for i := 0; i < n && i < len(list); i++ {
34         res = append(res, list[i].url)
35     }
36     return strings.Join(res, ",")
37 }
38
39 func main() {
40     cache := make(map[string]int) // 哈希表存储 URL 及其出现次数
41     scanner := bufio.NewScanner(os.Stdin)
42
43     for scanner.Scan() {
44         line := strings.TrimSpace(scanner.Text())
45     }
```

```

46         if num, err := strconv.Atoi(line); err == nil { // 如果输入是纯数字，处理
47             排序并输出结果
48                 fmt.Println(sortURL(num, cache))
49                 continue
50             }
51             cache[line]++
52         }

```

来自: [华为OD机试2025 C卷 - 热点网站统计 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

## 热点网站统计

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

华为OD机试2025C卷 100分题型

## 题目描述

企业路由器的统计页面，有一个功能需要动态统计公司访问最多的网页URL top N。请设计一个算法，可以高效动态统计Top N的页面。

## 输入描述

每一行都是一个URL或一个数字，如果是URL，代表一段时间内的网页访问； 如果是一个数字N，代表本次需要输出的Top N个URL。

输入约束：

1. 总访问网页数量小于5000个，单网页访问次数小于65535次；
2. 网页URL仅由字母，数字和点分隔符组成，且长度小于等于127字节；
3. 数字是正整数，小于等于10且小于当前总访问网页数；

## 输出描述

行输入要对应一行输出，输出按访问次数排序的前N个URL，用逗号分隔。

输出要求：

1. 每次输出要统计之前所有输入，不仅是本次输入；
2. 如果有访问次数相等的URL，按URL的字符串字典序升序排列，输出排序靠前的URL；

## 示例1

### 输入

	▼	Plain Text
1	news.qq.com	
2	news.sina.com.cn	
3	news.qq.com	
4	news.qq.com	
5	game.163.com	
6	game.163.com	
7	www.huawei.com	
8	www.cctv.com	
9	3	
10	www.huawei.com	
11	www.cctv.com	
12	www.huawei.com	
13	www.cctv.com	
14	www.huawei.com	
15	www.cctv.com	
16	www.huawei.com	
17	www.cctv.com	
18	www.huawei.com	
19	3	

## 输出

	▼	Plain Text
1	news.qq.com,game.163.com,news.sina.com.cn	
2	www.huawei.com,www.cctv.com,news.qq.com	

## 示例2

### 输入

	▼	Plain Text
1	news.qq.com	
2	www.cctv.com	
3	1	
4	www.huawei.com	
5	www.huawei.com	
6	2	
7	3	

### 输出

```
1 news.qq.com
2 www.huawei.com,news.qq.com
3 www.huawei.com,news.qq.com,www.cctv.com
```

## 题解

思路：模拟，由于本题的数据量比较小，可以按照下面这种代码通过。当数据量较大时可以进行使用优先队列排序。

- 使用哈希表统计出现的网站的次数。
- 当遇到数字时，进行自定义排序，对出现网站进行排序先按照次数排序升序，当出现次数相同时按照字典序升序。
- 输出前N个网站即可。

**C++**

```
1  #include <iostream>
2  #include <vector>
3  #include <map>
4  #include <algorithm>
5  #include <regex>
6
7  using namespace std;
8
9  string sortURL(int n, map<string, int>& cache);
10
11 int main() {
12     map<string, int> cache; // 创建一个哈希表, 用于存储每个 URL 出现的次数
13     string line;
14     while (getline(cin, line)) { // 不断读取输入, 直到没有下一行
15         if (regex_match(line, regex("^\\d+$"))) { // 如果该行数据只包含数字,
说明已经读取完了一个测试用例
16             cout << sortURL(stoi(line), cache) << endl;
17             continue;
18         }
19         cache[line]++;
20     }
21     return 0;
22 }
23
24 string sortURL(int n, map<string, int>& cache) {
25     vector<pair<string, int>> list(cache.begin(), cache.end()); // 将哈希表
中的每一项转换成一个键值对, 并存入一个列表中
26
27     sort(list.begin(), list.end(), [](const pair<string, int>& a, const pa
ir<string, int>& b) { // 对列表进行排序, 按照计数从大到小排序, 如果计数相同则按照字
典序从小到大排序
28         if (a.second != b.second) {
29             return a.second > b.second;
30         } else {
31             return a.first < b.first;
32         }
33     });
34
35     string res;
36     for (int i = 0; i < n && i < list.size(); i++) { // 取出前 n 个 URL, 并
将它们拼接成一个字符串
37         res += list[i].first + ",";
38     }
39     if (!res.empty()) { // 如果字符串不为空, 则删除最后一个逗号
40         res.pop_back();
```

```
41     }  
42     return res; // 返回拼接好的字符串  
43 }
```

## Java



```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          Map<String, Integer> cache = new HashMap<>(); // 哈希表存储 URL 及其
出现次数
7
8          while (scanner.hasNextLine()) {
9              String line = scanner.nextLine().trim();
10             if (line.matches("^\\d+$")) { // 如果输入是纯数字, 处理排序并输出结
果
11                 System.out.println(sortURL(Integer.parseInt(line), cach
e));
12                 continue;
13             }
14             cache.put(line, cache.getOrDefault(line, 0) + 1);
15         }
16         scanner.close();
17     }
18
19     private static String sortURL(int n, Map<String, Integer> cache) {
20         List<Map.Entry<String, Integer>> list = new ArrayList<>(cache.entr
ySet());
21
22         // 按出现次数降序排列, 次数相同则按字典序升序排列
23         list.sort((a, b) -> {
24             if (!a.getValue().equals(b.getValue())) {
25                 return b.getValue() - a.getValue();
26             }
27             return a.getKey().compareTo(b.getKey());
28         });
29
30         // 取前 n 个 URL 并拼接成字符串
31         StringBuilder res = new StringBuilder();
32         for (int i = 0; i < Math.min(n, list.size()); i++) {
33             res.append(list.get(i).getKey()).append(",");
34         }
35         if (res.length() > 0) {
36             res.setLength(res.length() - 1); // 删除末尾的逗号
37         }
38         return res.toString();
39     }
40 }
```

## Python

```
▼ Plain Text |
1  import sys
2  import re
3
4  def sort_url(n, cache):
5      # 按出现次数降序排序, 次数相同则按字典序升序
6      sorted_urls = sorted(cache.items(), key=lambda x: (-x[1], x[0]))
7
8      # 取前 n 个 URL 并拼接成字符串
9      return ",".join(url for url, _ in sorted_urls[:n])
10
11 def main():
12     cache = {} # 哈希表存储 URL 及其出现次数
13
14     for line in sys.stdin:
15         line = line.strip()
16         if re.fullmatch(r"\d+", line): # 如果输入是纯数字, 处理排序并输出结果
17             print(sort_url(int(line), cache))
18             continue
19         cache[line] = cache.get(line, 0) + 1
20
21 if __name__ == "__main__":
22     main()
```

## JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  let cache = {}; // 哈希表存储 URL 及其出现次数
9  let lines = [];
10
11  rl.on("line", (line) => {
12      line = line.trim();
13      if (/^\d+$/ .test(line)) { // 如果输入是纯数字，处理排序并输出结果
14          console.log(sortURL(parseInt(line), cache));
15          return;
16      }
17      cache[line] = (cache[line] || 0) + 1;
18  });
19
20  function sortURL(n, cache) {
21      // 转换为数组并排序，按次数降序，次数相同则按字典序升序
22      let sortedList = Object.entries(cache).sort((a, b) => {
23          if (b[1] !== a[1]) {
24              return b[1] - a[1];
25          }
26          return a[0].localeCompare(b[0]);
27      });
28
29      // 取前 n 个 URL 并拼接成字符串
30      return sortedList.slice(0, n).map(item => item[0]).join(",");
31  }
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12 func sortURL(n int, cache map[string]int) string {
13     // 将哈希表转换为切片
14     type urlCount struct {
15         url    string
16         count  int
17     }
18     list := make([]urlCount, 0, len(cache))
19     for url, count := range cache {
20         list = append(list, urlCount{url, count})
21     }
22
23     // 按出现次数降序, 次数相同则按字典序升序
24     sort.Slice(list, func(i, j int) bool {
25         if list[i].count != list[j].count {
26             return list[i].count > list[j].count
27         }
28         return list[i].url < list[j].url
29     })
30
31     // 取前 n 个 URL 并拼接成字符串
32     var res []string
33     for i := 0; i < n && i < len(list); i++ {
34         res = append(res, list[i].url)
35     }
36     return strings.Join(res, ",")
37 }
38
39 func main() {
40     cache := make(map[string]int) // 哈希表存储 URL 及其出现次数
41     scanner := bufio.NewScanner(os.Stdin)
42
43     for scanner.Scan() {
44         line := strings.TrimSpace(scanner.Text())
45     }
```

```
46         if num, err := strconv.Atoi(line); err == nil { // 如果输入是纯数字, 处理
47             排序并输出结果
48                 fmt.Println(sortURL(num, cache))
49                 continue
50             }
51             cache[line]++
52         }
```

来自: [华为OD机试2025 C卷 – 热点网站统计 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

来自: [华为OD机试2025 C卷 – 热点网站统计 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

# 华为OD 机试 2025C卷 - 高矮个子排队 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 高矮个子排队

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

华为OD机试2025C卷 100分题型

### 题目描述

现在有一队小朋友，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：

▼ Plain Text

1 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

## 示例2

输入

▼	Plain Text
1	1 1 1 1 1

输出

▼	Plain Text
1	1 1 1 1 1

## 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。

## 题解

思路：贪心算法实现

1. 这个题其实有点坑的，要求按照 高矮高矮 的方式进行排序，并要求移动距离最小，但是根据示例1来看并不是这样。对于这种情况一般是根据示例1来编写代码。
2. 根据示例1的结果来推断，这道题的逻辑排序需要先把前面的人排好，再去考虑后面。
3. 那我们可以使用 贪心 进行排序，处理逻辑如下，遍历到i位置时
  - a. 如果 `ans[i] > ans[i-1] && (i % 2) == 1` : 交换 `ans[i], ans[i-1]` 元素。
  - b. `ans[i] < ans[i-1] && (i % 2) == 0` : 交换 `ans[i] < ans[i-1]` 元素。
4. 不存在异常数据的情况，按照3的逻辑进行排序，排序之后的序列就是结果，直接输出即可。额外需要注意输入异常的情况，对于出现不合法的情况直接输出 `[]` .

C++



```
1  #include <cctype>
2  #include<iostream>
3  #include<vector>
4  #include<string>
5  #include <utility>
6  #include <sstream>
7  #include<algorithm>
8  using namespace std;
9
10 // 解析字符串到int
11 int parseStringToInt(string s) {
12     int res = 0;
13     for (int i = 0; i < s.size();i++) {
14         if ( '0' <=s[i] && s[i] <= '9') {
15             res = res * 10 + s[i] -'0';
16         } else {
17             return -1;
18         }
19     }
20     return res;
21 }
22
23
24 int main() {
25     vector<int> ans;
26     string s;
27     while (cin >> s) {
28         if (s == "") {
29             break;
30         }
31         int tmp = parseStringToInt(s);
32         // 出现不合法数据
33         if (tmp == -1) {
34             cout << "[ ]";
35             return 0;
36         }
37         ans.push_back(tmp);
38     }
39
40     // 贪心算法
41     for (int i = 1; i < ans.size(); i++) {
42         if (ans[i] > ans[i-1] && (i % 2)) {
43             swap(ans[i], ans[i-1]);
44         }
45     }
```

```
46         if (ans[i] < ans[i-1] && !(i % 2)) {
47             swap(ans[i], ans[i-1]);
48         }
49     }
50     for (int i = 0 ; i < ans.size(); i++) {
51         cout << ans[i] << " ";
52     }
53     return 0;
54 }
```

## Java

```
1  import java.util.*;
2
3  public class Main {
4      // 解析字符串到 int
5      public static int parseStringToInt(String s) {
6          int res = 0;
7          for (char c : s.toCharArray()) {
8              if (Character.isDigit(c)) {
9                  res = res * 10 + (c - '0');
10             } else {
11                 return -1; // 非数字字符, 返回 -1
12             }
13         }
14         return res;
15     }
16
17     public static void main(String[] args) {
18         Scanner scanner = new Scanner(System.in);
19         List<Integer> ans = new ArrayList<>();
20
21         while (scanner.hasNext()) {
22             String s = scanner.next();
23             int tmp = parseStringToInt(s);
24             if (tmp == -1) {
25                 System.out.println("[ ]");
26                 scanner.close();
27                 return;
28             }
29             ans.add(tmp);
30         }
31         scanner.close();
32
33         // 贪心算法
34         for (int i = 1; i < ans.size(); i++) {
35             if (ans.get(i) > ans.get(i - 1) && (i % 2 == 1)) {
36                 Collections.swap(ans, i, i - 1);
37             }
38             if (ans.get(i) < ans.get(i - 1) && (i % 2 == 0)) {
39                 Collections.swap(ans, i, i - 1);
40             }
41         }
42
43         for (int i = 0; i < ans.size(); i++) {
44             if (i > 0) System.out.print(" ");
45             System.out.print(ans.get(i));
```

```
46         }
47     }
48 }
```

## Python

▼ Plain Text

```
1  import sys
2
3  # 解析字符串到 int
4  def parse_string_to_int(s):
5      if s.isdigit():
6          return int(s)
7      return -1 # 非数字字符返回 -1
8
9  def main():
10     # 读取整行输入
11     line = sys.stdin.readline().strip()
12
13     # 处理空输入
14     if not line:
15         print("[ ]")
16         return
17
18     # 分割输入并解析
19     ans = []
20     for s in line.split():
21         tmp = parse_string_to_int(s)
22         if tmp == -1:
23             print("[ ]")
24             return
25         ans.append(tmp)
26
27     # 贪心算法
28     for i in range(1, len(ans)):
29         if ans[i] > ans[i - 1] and (i % 2 == 1):
30             ans[i], ans[i - 1] = ans[i - 1], ans[i]
31         if ans[i] < ans[i - 1] and (i % 2 == 0):
32             ans[i], ans[i - 1] = ans[i - 1], ans[i]
33
34     print(" ".join(map(str, ans)))
35
36 if __name__ == "__main__":
37     main()
```

# JavaScript

Plain Text

```
1  const readline = require('readline');
2
3  // 解析字符串到 int
4  function parseStringToInt(s) {
5      if (/^\d+$/.test(s)) {
6          return parseInt(s, 10);
7      }
8      return -1; // 非数字字符返回 -1
9  }
10
11 const rl = readline.createInterface({
12     input: process.stdin,
13     output: process.stdout
14 });
15
16 rl.on('line', (line) => {
17     let tokens = line.trim().split(/\s+/);
18
19     let ans = [];
20     for (let s of tokens) {
21         let num = parseStringToInt(s);
22         if (num === -1) {
23             console.log("[ ]");
24             rl.close();
25             return;
26         }
27         ans.push(num);
28     }
29
30     // 贪心算法
31     for (let i = 1; i < ans.length; i++) {
32         if (ans[i] > ans[i - 1] && (i % 2 === 1)) {
33             [ans[i], ans[i - 1]] = [ans[i - 1], ans[i]];
34         }
35         if (ans[i] < ans[i - 1] && (i % 2 === 0)) {
36             [ans[i], ans[i - 1]] = [ans[i - 1], ans[i]];
37         }
38     }
39
40     console.log(ans.join(" "));
41     rl.close();
42 });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9  )
10
11 // 解析字符串到 int
12 func parseStringToInt(s string) (int, bool) {
13     num, err := strconv.Atoi(s)
14     if err != nil {
15         return -1, false // 解析失败返回 -1, 并标记为非法输入
16     }
17     return num, true
18 }
19
20 func main() {
21     // 读取整行输入
22     scanner := bufio.NewScanner(os.Stdin)
23     if !scanner.Scan() {
24         fmt.Println("[ ]")
25         return
26     }
27     line := strings.TrimSpace(scanner.Text())
28
29     // 处理空输入
30     if line == "" {
31         fmt.Println("[ ]")
32         return
33     }
34
35     // 解析输入
36     tokens := strings.Fields(line)
37     ans := make([]int, 0, len(tokens))
38
39     for _, s := range tokens {
40         num, valid := parseStringToInt(s)
41         if !valid {
42             fmt.Println("[ ]")
43             return
44         }
45         ans = append(ans, num)
```

```

46     }
47
48     // 贪心算法
49     for i := 1; i < len(ans); i++ {
50         if ans[i] > ans[i-1] && (i%2 == 1) {
51             ans[i], ans[i-1] = ans[i-1], ans[i]
52         }
53         if ans[i] < ans[i-1] && (i%2 == 0) {
54             ans[i], ans[i-1] = ans[i-1], ans[i]
55         }
56     }
57
58     // 输出结果
59     fmt.Println(strings.Trim(fmt.Sprint(ans), "[]"))
60 }

```

来自: [华为OD 机试 2025C卷 – 高矮个子排队 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)



# 华为OD机试 2025 C卷 - 精准核酸检测 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 精准核酸检测

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

华为OD机试2025C卷 100分题型

## 题目描述

为了达到新冠疫情精准防控的需要，为了避免全员核酸检测带来的浪费，需要精准圈定可能被感染的人群。

现在根据传染病流调以及[大数据分析](#)，得到了每个人之间在时间、空间上是否存在轨迹交叉。

现在给定一组确诊人员编号 ( $X_1, X_2, X_3, \dots, X_n$ )，在所有人当中，找出哪些人需要进行核酸检测，输出需要进行核酸检测的人数。（注意：确诊病例自身不需要再做核酸检测）

需要进行核酸检测的人，是病毒传播链条上的所有人员，即有可能通过确诊病例所能传播到的所有人。

例如：A是确诊病例，A和B有接触、B和C有接触、C和D有接触、D和E有接触，那么B\C\D\E都是需要进行核酸检测的人。

## 输入描述

第一行为总人数  $N$

第二行为确诊病例人员编号（确诊病例人员数量  $< N$ ），用逗号分割

第三行开始，为一个  $N * N$  的矩阵，表示每个人员之间是否有接触，0表示没有接触，1表示有接触。

## 备注

- 人员编号从0开始
- $0 < N < 100$

## 输出描述

整数：需要做核酸检测的人数

## 用例1

### 输入

	Plain Text
1	5
2	1,2
3	1,1,0,1,0
4	1,1,0,0,0
5	0,0,1,0,1
6	1,0,0,1,0
7	0,0,1,0,1

## 输出

	Plain Text
1	3

## 说明

编号为1、2号的人员，为确诊病例。1号和0号有接触，0号和3号有接触。2号和4号有接触。所以，需要做核酸检测的人是0号、3号、4号，总计3人需要进行核酸检测。

## 题解

思路：并查集 算法实现

1. 题目说明 需要进行核酸检测的人，是病毒传播链条上的所有人员，即有可能通过确诊病例所能传播到的所有人。 ，可以直接使用 并查集算法 将直接和间接接触的人员放入到同一个组中。
2. 至于判断 需要做核酸检测的人数 ，当使用并查集算法处理完所有输入数据之后。如果一个人和 任何一个确诊病例 处于同一个组种他就需要进行核酸检测。
3. 需要额外注意最终结果需要减去病例本身数量。题目种明确说明了 确诊病例自身不需要再做核酸检测

优化点建议：

1. 矩阵是对称的，进行并查集合并的时候，可以只遍历 矩阵的左半部分 。
1. 推荐使用 并查集 算法一定要使用路径压缩。这可以加速下一次查询的速度。

建议： 并查集算法是机试中高频考点，一定要掌握。而且并查集的算法属于模板算法，尽可能背一下代码模板。

## C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<list>
8  #include<queue>
9  #include<map>
10 #include<set>
11 using namespace std;
12
13 // 通用 split 函数
14 vector<int> split(const string& str, const string& delimiter) {
15     vector<int> result;
16     size_t start = 0;
17     size_t end = str.find(delimiter);
18     while (end != string::npos) {
19         result.push_back(stoi(str.substr(start, end - start)));
20         start = end + delimiter.length();
21         end = str.find(delimiter, start);
22     }
23     // 添加最后一个部分
24     result.push_back(stoi(str.substr(start)));
25     return result;
26 }
27
28 // 找到组中的代表人
29 int find(int a, vector<int> &ans) {
30     if (ans[a] != a) {
31         ans[a] = find(ans[a], ans);
32     }
33     return ans[a];
34 }
35
36 // 合并两个组
37 void merge(int a, int b, vector<int> &ans) {
38     int rootA = find(a, ans);
39     int rootB = find(b, ans);
40     int newRoot = min(rootA, rootB);
41     ans[rootA] = newRoot;
42     ans[rootB] = newRoot;
43 }
44
45
```

```

46 int main() {
47     int n;
48     cin >> n;
49
50     string confirm;
51     // 忽略换行符
52     cin.ignore();
53     getline(cin, confirm);
54     if (confirm.empty()) {
55         cout << 0;
56         return 0;
57     }
58     vector<int> confirmId = split(confirm, ",");
59     vector<vector<int>> grid(n);
60     for (int i = 0; i < n; i++) {
61         string tmp;
62         getline(cin, tmp);
63         grid[i] = split(tmp, ",");
64     }
65
66     vector<int> ans(n);
67     for (int i = 0; i < n; i++) {
68         ans[i] = i;
69     }
70
71     // 对称的, 只需要遍历一半除对角线
72     for (int i = 0; i < n; i++) {
73         for (int j = 0; j < i; j++) {
74             // 合并两个组
75             if (grid[i][j] == 1) {
76                 merge(i, j, ans);
77             }
78         }
79     }
80
81     int res = 0;
82     set<int> s;
83     // 使用集合存储所有组长
84     for (int i = 0; i < confirmId.size(); i++) {
85         s.insert(find(confirmId[i], ans));
86     }
87
88     for (int i = 0; i < n; i++) {
89         // 处于密切接触组
90         if (s.find(find(i, ans)) != s.end()) {
91             res++;
92         }
93     }

```

```
94 // 减去病例人数
95 res -= confirmId.size();
96 cout << res;
97 return 0;
98 }
```

## JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 查找代表人
5      public static int find(int a, int[] ans) {
6          if (ans[a] != a) {
7              ans[a] = find(ans[a], ans);
8          }
9          return ans[a];
10     }
11
12     // 合并两个组
13     public static void merge(int a, int b, int[] ans) {
14         int rootA = find(a, ans);
15         int rootB = find(b, ans);
16         int newRoot = Math.min(rootA, rootB);
17         ans[rootA] = newRoot;
18         ans[rootB] = newRoot;
19     }
20
21     public static void main(String[] args) {
22         Scanner sc = new Scanner(System.in);
23         int n = Integer.parseInt(sc.nextLine());
24         String confirm = sc.nextLine().trim();
25         // 没有病例的情况
26         if (confirm.isEmpty()) {
27             System.out.println(0);
28             return;
29         }
30
31         String[] confirmParts = confirm.split(",");
32         List<Integer> confirmIds = new ArrayList<>();
33         for (String s : confirmParts) {
34             confirmIds.add(Integer.parseInt(s));
35         }
36
37         int[][] grid = new int[n][n];
38         for (int i = 0; i < n; i++) {
39             String[] parts = sc.nextLine().split(",");
40             for (int j = 0; j < n; j++) {
41                 grid[i][j] = Integer.parseInt(parts[j]);
42             }
43         }
44
45         int[] ans = new int[n];
```

```

46 // 初始化
47 for (int i = 0; i < n; i++) ans[i] = i;
48
49 // 合并组
50 for (int i = 0; i < n; i++) {
51     for (int j = 0; j < i; j++) {
52         if (grid[i][j] == 1) {
53             merge(i, j, ans);
54         }
55     }
56 }
57
58 Set<Integer> infectedGroups = new HashSet<>();
59 // 使用集合存储病例所在组
60 for (int id : confirmIds) {
61     infectedGroups.add(find(id, ans));
62 }
63
64 int res = 0;
65 for (int i = 0; i < n; i++) {
66     if (infectedGroups.contains(find(i, ans))) {
67         res++;
68     }
69 }
70
71 // 减去已知病例人数
72 System.out.println(res - confirmIds.size());
73 }
74 }

```

## Python

```
1  # 并查集查找代表人
2  def find(a, parent):
3      if parent[a] != a:
4          parent[a] = find(parent[a], parent)
5      return parent[a]
6
7  # 合并两个组
8  def merge(a, b, parent):
9      rootA = find(a, parent)
10     rootB = find(b, parent)
11     new_root = min(rootA, rootB)
12     parent[rootA] = new_root
13     parent[rootB] = new_root
14
15 if __name__ == "__main__":
16     n = int(input())
17     confirm = input().strip()
18     # 不存在病例的情况
19     if not confirm:
20         print(0)
21         exit()
22
23     confirm_ids = list(map(int, confirm.split(",")))
24     grid = []
25     for _ in range(n):
26         row = list(map(int, input().strip().split(",")))
27         grid.append(row)
28
29     parent = list(range(n))
30
31     # 只遍历一半（对称矩阵）
32     for i in range(n):
33         for j in range(i):
34             # 合并组
35             if grid[i][j] == 1:
36                 merge(i, j, parent)
37
38     # 存储病例所在组
39     infected_groups = set(find(id, parent) for id in confirm_ids)
40
41     res = 0
42     for i in range(n):
43         if find(i, parent) in infected_groups:
44             res += 1
45
46     # 减去已知病例人数
```



```
46     print(res - len(confirm_ids))
```

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let inputLines = [];
9  rl.on('line', (line) => {
10    inputLines.push(line.trim());
11  }).on('close', () => {
12    let n = parseInt(inputLines[0]); // 读取矩阵大小
13
14    // 若无确诊病例
15    if (inputLines.length < 2 || inputLines[1] === '') {
16      console.log(0);
17      return;
18    }
19
20    // 读取确诊病例编号
21    let confirmId = inputLines[1].split(',').map(Number);
22
23    // 构造邻接矩阵
24    let grid = [];
25    for (let i = 0; i < n; i++) {
26      grid.push(inputLines[2 + i].split(',').map(Number));
27    }
28
29    // 并查集初始化，每人是自己的代表人
30    let parent = Array.from({ length: n }, (_, i) => i);
31
32    // 查找组的代表人
33    function find(a) {
34      if (parent[a] !== a) {
35        parent[a] = find(parent[a]);
36      }
37      return parent[a];
38    }
39
40    // 合并两个组
41    function merge(a, b) {
42      let rootA = find(a);
43      let rootB = find(b);
44      let newRoot = Math.min(rootA, rootB);
45      parent[rootA] = newRoot;
```

```

46         parent[rootB] = newRoot;
47     }
48
49     // 遍历下三角区域（不含对角线）合并接触者
50     for (let i = 0; i < n; i++) {
51         for (let j = 0; j < i; j++) {
52             if (grid[i][j] === 1) {
53                 merge(i, j);
54             }
55         }
56     }
57
58     // 收集确诊病例所在的组
59     let infectedGroups = new Set();
60     for (let id of confirmId) {
61         infectedGroups.add(find(id));
62     }
63
64     // 统计密接人数（不含确诊者）
65     let result = 0;
66     for (let i = 0; i < n; i++) {
67         if (infectedGroups.has(find(i)) && !confirmId.includes(i)) {
68             result++;
69         }
70     }
71
72     console.log(result);
73 });

```

**Go**

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 查找代表人
12 func find(a int, parent []int) int {
13     if parent[a] != a {
14         parent[a] = find(parent[a], parent)
15     }
16     return parent[a]
17 }
18
19 // 合并两个组
20 func merge(a, b int, parent []int) {
21     rootA := find(a, parent)
22     rootB := find(b, parent)
23     newRoot := min(rootA, rootB)
24     parent[rootA] = newRoot
25     parent[rootB] = newRoot
26 }
27
28 func min(a, b int) int {
29     if a < b {
30         return a
31     }
32     return b
33 }
34
35 func main() {
36     reader := bufio.NewReader(os.Stdin)
37
38     line, _ := reader.ReadString('\n')
39     n, _ := strconv.Atoi(strings.TrimSpace(line))
40
41     confirmLine, _ := reader.ReadString('\n')
42     confirmLine = strings.TrimSpace(confirmLine)
43     // 没有病例
44     if confirmLine == "" {
45         fmt.Println(0)
```

```

46         return
47     }
48
49     confirmStrs := strings.Split(confirmLine, ",")
50     confirmIds := make([]int, len(confirmStrs))
51     for i, s := range confirmStrs {
52         confirmIds[i], _ = strconv.Atoi(s)
53     }
54
55     grid := make([][]int, n)
56     for i := 0; i < n; i++ {
57         line, _ := reader.ReadString('\n')
58         parts := strings.Split(strings.TrimSpace(line), ",")
59         grid[i] = make([]int, n)
60         for j := range parts {
61             grid[i][j], _ = strconv.Atoi(parts[j])
62         }
63     }
64
65     parent := make([]int, n)
66     // 初始化
67     for i := range parent {
68         parent[i] = i
69     }
70     // 对称矩阵 遍历一半
71     for i := 0; i < n; i++ {
72         for j := 0; j < i; j++ {
73             // 合并组
74             if grid[i][j] == 1 {
75                 merge(i, j, parent)
76             }
77         }
78     }
79     // 病例所在组
80     infected := make(map[int]bool)
81     for _, id := range confirmIds {
82         infected[find(id, parent)] = true
83     }
84
85     res := 0
86     // 统计和病例处于同一组的人数(包含病例本身)
87     for i := 0; i < n; i++ {
88         if infected[find(i, parent)] {
89             res++
90         }
91     }
92     // 减去病例数量
93     fmt.Println(res - len(confirmIds))

```

## 精准核酸检测

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

华为OD机试2025C卷 100分题型

### 题目描述

为了达到新冠疫情精准防控的需要, 为了避免全员核酸检测带来的浪费, 需要精准圈定可能被感染的人群。

现在根据传染病流调以及[大数据分析](#), 得到了每个人之间在时间、空间上是否存在轨迹交叉。

现在给定一组确诊人员编号 ( $X_1, X_2, X_3, \dots, X_n$ ), 在所有人当中, 找出哪些人需要进行核酸检测, 输出需要进行核酸检测的人数。(注意: 确诊病例自身不需要再做核酸检测)

需要进行核酸检测的人, 是病毒传播链条上的所有人员, 即有可能通过确诊病例所能传播到的所有人。

例如: A是确诊病例, A和B有接触、B和C有接触、C和D有接触、D和E有接触, 那么B\C\D\E都是需要进行核酸检测的人。

### 输入描述

第一行为总人数  $N$

第二行为确认病例人员编号 (确诊病例人员数量  $< N$ ), 用逗号分割

第三行开始, 为一个  $N * N$  的矩阵, 表示每个人员之间是否有接触, 0表示没有接触, 1表示有接触。

### 备注

- 人员编号从0开始
- $0 < N < 100$

### 输出描述

整数: 需要做核酸检测的人数

### 用例1

#### 输入

```
5
1,2
1,1,0,1,0
1,1,0,0,0
0,0,1,0,1
1,0,0,1,0
0,0,1,0,1
```

Plain Text

# 输出

▼

Plain Text

13

# 说明

编号为1、2号的人员，为确诊病例。1号和0号有接触，0号和3号有接触。2号和4号有接触。所以，需要做核酸检测的人是0号、3号、4号，总计3人需要进行核酸检测。

# 题解

思路：并查集 算法实现

- 1. 题目说明 需要进行核酸检测的人，是病毒传播链条上的所有人员，即有可能通过确诊病例所能传播到的所有人。 ，可以直接使用 并查集算法 将直接和间接接触的人员放入到同一个组中。
- 2. 至于判断 需要做核酸检测的人数 ，当使用并查集算法处理完所有输入数据之后。如果一个人和 任何一个确诊病例 处于同一个组种他就需要进行核酸检测。
- 3. 需要额外注意最终结果需要减去病例本身数量。题目种明确说明了 确诊病例自身不需要再做核酸检测

优化点建议：

- 1. 矩阵是对称的，进行并查集合并的时候，可以只遍历 矩阵的左半部分 。
- 1. 推荐使用 并查集 算法一定要使用路径压缩。这可以加速下一次查询的速度。

建议： 并查集算法是机试中高频考点，一定要掌握。而且并查集的算法属于模板算法，尽可能背一下代码模板。

# C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<list>
8  #include<queue>
9  #include<map>
10 #include<set>
11 using namespace std;
12
13 // 通用 split 函数
14 vector<int> split(const string& str, const string& delimiter) {
15     vector<int> result;
16     size_t start = 0;
17     size_t end = str.find(delimiter);
18     while (end != string::npos) {
19         result.push_back(stoi(str.substr(start, end - start)));
20         start = end + delimiter.length();
21         end = str.find(delimiter, start);
22     }
23     // 添加最后一个部分
24     result.push_back(stoi(str.substr(start)));
25     return result;
26 }
27
28 // 找到组中的代表人
29 int find(int a, vector<int> &ans) {
30     if (ans[a] != a) {
31         ans[a] = find(ans[a], ans);
32     }
33     return ans[a];
34 }
35
36 // 合并两个组
37 void merge(int a, int b, vector<int> &ans) {
38     int rootA = find(a, ans);
39     int rootB = find(b, ans);
40     int newRoot = min(rootA, rootB);
41     ans[rootA] = newRoot;
42     ans[rootB] = newRoot;
43 }
44
45
```



```

46 int main() {
47     int n;
48     cin >> n;
49
50     string confirm;
51     // 忽略换行符
52     cin.ignore();
53     getline(cin, confirm);
54     if (confirm.empty()) {
55         cout << 0;
56         return 0;
57     }
58     vector<int> confirmId = split(confirm, ",");
59     vector<vector<int>> grid(n);
60     for (int i = 0; i < n; i++) {
61         string tmp;
62         getline(cin, tmp);
63         grid[i] = split(tmp, ",");
64     }
65
66     vector<int> ans(n);
67     for (int i = 0; i < n; i++) {
68         ans[i] = i;
69     }
70
71     // 对称的, 只需要遍历一半除对角线
72     for (int i = 0; i < n; i++) {
73         for (int j = 0; j < i; j++) {
74             // 合并两个组
75             if (grid[i][j] == 1) {
76                 merge(i, j, ans);
77             }
78         }
79     }
80
81     int res = 0;
82     set<int> s;
83     // 使用集合存储所有组长
84     for (int i = 0; i < confirmId.size(); i++) {
85         s.insert(find(confirmId[i], ans));
86     }
87
88     for (int i = 0; i < n; i++) {
89         // 处于密切接触组
90         if (s.find(find(i, ans)) != s.end()) {
91             res++;
92         }
93     }

```

```
94 // 减去病例人数
95 res -= confirmId.size();
96 cout << res;
97 return 0;
98 }
```

## JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 查找代表人
5      public static int find(int a, int[] ans) {
6          if (ans[a] != a) {
7              ans[a] = find(ans[a], ans);
8          }
9          return ans[a];
10     }
11
12     // 合并两个组
13     public static void merge(int a, int b, int[] ans) {
14         int rootA = find(a, ans);
15         int rootB = find(b, ans);
16         int newRoot = Math.min(rootA, rootB);
17         ans[rootA] = newRoot;
18         ans[rootB] = newRoot;
19     }
20
21     public static void main(String[] args) {
22         Scanner sc = new Scanner(System.in);
23         int n = Integer.parseInt(sc.nextLine());
24         String confirm = sc.nextLine().trim();
25         // 没有病例的情况
26         if (confirm.isEmpty()) {
27             System.out.println(0);
28             return;
29         }
30
31         String[] confirmParts = confirm.split(",");
32         List<Integer> confirmIds = new ArrayList<>();
33         for (String s : confirmParts) {
34             confirmIds.add(Integer.parseInt(s));
35         }
36
37         int[][] grid = new int[n][n];
38         for (int i = 0; i < n; i++) {
39             String[] parts = sc.nextLine().split(",");
40             for (int j = 0; j < n; j++) {
41                 grid[i][j] = Integer.parseInt(parts[j]);
42             }
43         }
44
45         int[] ans = new int[n];
```

```

46 // 初始化
47 for (int i = 0; i < n; i++) ans[i] = i;
48
49 // 合并组
50 for (int i = 0; i < n; i++) {
51     for (int j = 0; j < i; j++) {
52         if (grid[i][j] == 1) {
53             merge(i, j, ans);
54         }
55     }
56 }
57
58 Set<Integer> infectedGroups = new HashSet<>();
59 // 使用集合存储病例所在组
60 for (int id : confirmIds) {
61     infectedGroups.add(find(id, ans));
62 }
63
64 int res = 0;
65 for (int i = 0; i < n; i++) {
66     if (infectedGroups.contains(find(i, ans))) {
67         res++;
68     }
69 }
70
71 // 减去已知病例人数
72 System.out.println(res - confirmIds.size());
73 }
74 }

```

## Python

```
1  # 并查集查找代表人
2  def find(a, parent):
3      if parent[a] != a:
4          parent[a] = find(parent[a], parent)
5      return parent[a]
6
7  # 合并两个组
8  def merge(a, b, parent):
9      rootA = find(a, parent)
10     rootB = find(b, parent)
11     new_root = min(rootA, rootB)
12     parent[rootA] = new_root
13     parent[rootB] = new_root
14
15  if __name__ == "__main__":
16     n = int(input())
17     confirm = input().strip()
18     # 不存在病例的情况
19     if not confirm:
20         print(0)
21         exit()
22
23     confirm_ids = list(map(int, confirm.split(",")))
24     grid = []
25     for _ in range(n):
26         row = list(map(int, input().strip().split(",")))
27         grid.append(row)
28
29     parent = list(range(n))
30
31     # 只遍历一半（对称矩阵）
32     for i in range(n):
33         for j in range(i):
34             # 合并组
35             if grid[i][j] == 1:
36                 merge(i, j, parent)
37
38     # 存储病例所在组
39     infected_groups = set(find(id, parent) for id in confirm_ids)
40
41     res = 0
42     for i in range(n):
43         if find(i, parent) in infected_groups:
44             res += 1
45
46     # 减去已知病例人数
```

```
46     print(res - len(confirm_ids))
```

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let inputLines = [];
9  rl.on('line', (line) => {
10    inputLines.push(line.trim());
11  }).on('close', () => {
12    let n = parseInt(inputLines[0]); // 读取矩阵大小
13
14    // 若无确诊病例
15    if (inputLines.length < 2 || inputLines[1] === '') {
16      console.log(0);
17      return;
18    }
19
20    // 读取确诊病例编号
21    let confirmId = inputLines[1].split(',').map(Number);
22
23    // 构造邻接矩阵
24    let grid = [];
25    for (let i = 0; i < n; i++) {
26      grid.push(inputLines[2 + i].split(',').map(Number));
27    }
28
29    // 并查集初始化，每人是自己的代表人
30    let parent = Array.from({ length: n }, (_, i) => i);
31
32    // 查找组的代表人
33    function find(a) {
34      if (parent[a] !== a) {
35        parent[a] = find(parent[a]);
36      }
37      return parent[a];
38    }
39
40    // 合并两个组
41    function merge(a, b) {
42      let rootA = find(a);
43      let rootB = find(b);
44      let newRoot = Math.min(rootA, rootB);
45      parent[rootA] = newRoot;
```

```

46     parent[rootB] = newRoot;
47 }
48
49 // 遍历下三角区域（不含对角线）合并接触者
50 for (let i = 0; i < n; i++) {
51     for (let j = 0; j < i; j++) {
52         if (grid[i][j] === 1) {
53             merge(i, j);
54         }
55     }
56 }
57
58 // 收集确诊病例所在的组
59 let infectedGroups = new Set();
60 for (let id of confirmId) {
61     infectedGroups.add(find(id));
62 }
63
64 // 统计密接人数（不含确诊者）
65 let result = 0;
66 for (let i = 0; i < n; i++) {
67     if (infectedGroups.has(find(i)) && !confirmId.includes(i)) {
68         result++;
69     }
70 }
71
72 console.log(result);
73 });

```

**Go**



```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strconv"
8      "strings"
9  )
10
11 // 查找代表人
12 func find(a int, parent []int) int {
13     if parent[a] != a {
14         parent[a] = find(parent[a], parent)
15     }
16     return parent[a]
17 }
18
19 // 合并两个组
20 func merge(a, b int, parent []int) {
21     rootA := find(a, parent)
22     rootB := find(b, parent)
23     newRoot := min(rootA, rootB)
24     parent[rootA] = newRoot
25     parent[rootB] = newRoot
26 }
27
28 func min(a, b int) int {
29     if a < b {
30         return a
31     }
32     return b
33 }
34
35 func main() {
36     reader := bufio.NewReader(os.Stdin)
37
38     line, _ := reader.ReadString('\n')
39     n, _ := strconv.Atoi(strings.TrimSpace(line))
40
41     confirmLine, _ := reader.ReadString('\n')
42     confirmLine = strings.TrimSpace(confirmLine)
43     // 没有病例
44     if confirmLine == "" {
45         fmt.Println(0)
```

```

46         return
47     }
48
49     confirmStrs := strings.Split(confirmLine, ",")
50     confirmIds := make([]int, len(confirmStrs))
51     for i, s := range confirmStrs {
52         confirmIds[i], _ = strconv.Atoi(s)
53     }
54
55     grid := make([][]int, n)
56     for i := 0; i < n; i++ {
57         line, _ := reader.ReadString('\n')
58         parts := strings.Split(strings.TrimSpace(line), ",")
59         grid[i] = make([]int, n)
60         for j := range parts {
61             grid[i][j], _ = strconv.Atoi(parts[j])
62         }
63     }
64
65     parent := make([]int, n)
66     // 初始化
67     for i := range parent {
68         parent[i] = i
69     }
70     // 对称矩阵 遍历一半
71     for i := 0; i < n; i++ {
72         for j := 0; j < i; j++ {
73             // 合并组
74             if grid[i][j] == 1 {
75                 merge(i, j, parent)
76             }
77         }
78     }
79     // 病例所在组
80     infected := make(map[int]bool)
81     for _, id := range confirmIds {
82         infected[find(id, parent)] = true
83     }
84
85     res := 0
86     // 统计和病例处于同一组的人数(包含病例本身)
87     for i := 0; i < n; i++ {
88         if infected[find(i, parent)] {
89             res++
90         }
91     }
92     // 减去病例数量
93     fmt.Println(res - len(confirmIds))

```

来自: [华为OD机试 2025 C卷 – 精准核酸检测 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

来自: [华为OD机试 2025 C卷 – 精准核酸检测 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

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

## 国际移动用户识别码(IMSI)匹配

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

华为OD机试2025C卷 100分题型

### 题目描述

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

输入用户的IMSI(国际移动用户识别码), 根据以下规则匹配配置列表:

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

### 输入描述

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

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

### 备注

保证输入格式正确, 无需考虑格式错误。

### 输出描述

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

### 用例1

#### 输入

	▼	Plain Text
1	1234567,1234567*	
2	123456789012345	

## 输出

	▼	Plain Text
1	1234567*	

## 说明

\*可以匹配0或多个任意字符，故输出1234567 \*

## 用例2

### 输入

	▼	Plain Text
1	123?????????345,123????*????345	
2	123456789012345	

## 输出

	▼	Plain Text
1	null	

## 说明

'?'字符只能匹配IMSI中为奇数下标的字符，故都不符合要求，返回null

## 题解

思路： 模拟实现

1. 此题主要注意三个点
  - ? 能且只能匹配一个字符，并且匹配字符要位于待匹配字符奇数位置。
  - \* 能匹配0或多个任意字符，并且 \* 最多只会出现依次。
  - 根据上面两个点，其实也可以推断出如果模式字符串中不包含 \*，两个字符串长度需要相等。
1. 明白上面两个点之后，从前往后进行一一匹配就行，处理逻辑如下，我们使用 `pattern` 表示配置，`str` 比较用户的IMSI：

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

**C++**

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

```

46         return true;
47     }
48     string suffix = pattern.substr(i+1);
49     int pos = suffix.size() - 1;
50     // 后缀进行匹配
51     for (int j = n - 1; j >= i && pos >= 0; j--, pos--) {
52         if (suffix[pos] == str[j]) {
53             continue;
54         } else if (suffix[pos] == '?') {
55             if (j % 2 == 0) {
56                 break;
57             }
58         }
59     }
60     // 判断后缀是否匹配成功
61     return pos < 0;
62 } else {
63     return false;
64 }
65 }
66 // 从这里返回说明不存在* 所以必需两者长度相同
67 return n == m;
68 }
69
70 int main() {
71     string input;
72     getline(cin, input);
73     vector<string> settings = split(input, ",");
74     string number;
75     getline(cin, number);
76
77     vector<string> res;
78     int n = settings.size();
79     for (int i = 0; i < n; i++) {
80         string pattern = settings[i];
81         bool flag = judge(pattern, number);
82         if (flag) {
83             res.push_back(pattern);
84         }
85     }
86     if (res.empty()) {
87         cout << "null";
88         return 0;
89     }
90     // 字典序升序
91     sort(res.begin(), res.end());
92     int m = res.size();
93     for (int i = 0; i < m; i++) {

```



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

## JAVA

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

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

## Python

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

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

## JavaScript

```
1  const rl = require('readline').createInterface({
2    input: process.stdin,
3    output: process.stdout
4  });
5
6  let lines = [];
7  rl.on('line', line => {
8    lines.push(line);
9    if (lines.length === 2) {
10      const settings = lines[0].split(',');
11      const number = lines[1];
12      const res = [];
13
14      // 匹配函数
15      function judge(pattern, str) {
16        if (pattern === '*') return true;
17        let n = str.length, m = pattern.length;
18        let i = 0;
19        for (; i < n && i < m; i++) {
20          if (pattern[i] === str[i]) continue;
21          else if (pattern[i] === '?') {
22            // 偶数位置不能匹配又不相同
23            if (i % 2 === 0) return false;
24          } else if (pattern[i] === '*') {
25            // 达到pattern的末尾并且为*肯定可以匹配
26            if (i === m - 1) return true;
27            const suffix = pattern.slice(i + 1);
28            // 后缀匹配
29            let pos = suffix.length - 1;
30            for (let j = n - 1; j >= i && pos >= 0; j--, pos--) {
31              if (suffix[pos] === str[j]) continue;
32              else if (suffix[pos] === '?') {
33                if (j % 2 === 0) break;
34              } else break;
35            }
36            return pos < 0;
37          } else return false;
38        }
39        return m === n;
40      }
41
42      for (const pattern of settings) {
43        if (judge(pattern, number)) {
44          res.push(pattern);
45        }
46      }
47    }
48  });
```

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

**Go**

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



```

46         break
47     }
48 }
49     return pos < 0
50 } else {
51     return false
52 }
53     i++
54 }
55     return m == n
56 }
57
58 func main() {
59     scanner := bufio.NewScanner(os.Stdin)
60     scanner.Scan()
61     settings := strings.Split(scanner.Text(), ",")
62     scanner.Scan()
63     number := scanner.Text()
64
65     var res []string
66     for _, pattern := range settings {
67         if judge(pattern, number) {
68             res = append(res, pattern)
69         }
70     }
71
72     if len(res) == 0 {
73         fmt.Println("null")
74     } else {
75         sort.Strings(res)
76         fmt.Println(strings.Join(res, ","))
77     }
78 }

```

## 国际移动用户识别码(IMSI)匹配

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

华为OD机试2025C卷 100分题型

### 题目描述

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

输入用户的IMSI(国际移动用户识别码)，根据以下规则匹配配置列表：

- “\*”匹配0个或连续多个任意字符。

- "?"“匹配下标为奇数的单个字符，比如123?中的”?"可以匹配123456789012345下标为3的字符’4’，下标从0开始。

## 输入描述

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

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

## 备注

保证输入格式正确，无需考虑格式错误。

## 输出描述

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

## 用例1

### 输入

	Plain Text
1	1234567,1234567*
2	123456789012345

### 输出

	Plain Text
1	1234567*

## 说明

\*可以匹配0或多个任意字符，故输出1234567 \*

## 用例2

### 输入

	Plain Text
1	123?????????345,123????*????345
2	123456789012345

## 输出

	Plain Text
1	null

## 说明

'?'字符只能匹配IMSI中为奇数下标的字符，故都不符合要求，返回null

## 题解

思路：模拟实现

1. 此题主要注意三个点

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

1. 明白上面两个点之后，从前往后进行一一匹配就行，处理逻辑如下，我们使用 pattern 表示配置，str 比较用户的IMSI：

- pattern[i] == str[i] : 跳过，比较下一个字符。
- pattern[i] != str[i] && pattern[i] == '?' : 此时只需要判断i是否为偶数，为偶数的话说明不能匹配，直接匹配失败。
- pattern[i] != str[i] && pattern[i] == '\*' : 其实遇到 \* 之后，并且题目已知 \* 最多只会出现1次，此时只需要比较后缀即可。举个例子 pattern = 123%45, str = 123456789012345 ,知道\*可以匹配任意多个字符，此时我们只需要考虑pattern在 \* 之后后缀 45 能否和str末尾能否匹配上即可。能匹配上就说明能匹配成功，否则会失败。后缀匹配阶段也需要考虑？

2. 根据上面两点进行模拟即可，找出匹配的配置加入数组。如果不存在满足条件的输出 null ,否则把满足要求的配置升序之后输出。

## C++

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

```

46         return true;
47     }
48     string suffix = pattern.substr(i+1);
49     int pos = suffix.size() - 1;
50     // 后缀进行匹配
51     for (int j = n - 1; j >= i && pos >= 0; j--, pos--) {
52         if (suffix[pos] == str[j]) {
53             continue;
54         } else if (suffix[pos] == '?') {
55             if (j % 2 == 0) {
56                 break;
57             }
58         }
59     }
60     // 判断后缀是否匹配成功
61     return pos < 0;
62 } else {
63     return false;
64 }
65 }
66 // 从这里返回说明不存在* 所以必需两者长度相同
67 return n == m;
68 }
69
70 int main() {
71     string input;
72     getline(cin, input);
73     vector<string> settings = split(input, ",");
74     string number;
75     getline(cin, number);
76
77     vector<string> res;
78     int n = settings.size();
79     for (int i = 0; i < n; i++) {
80         string pattern = settings[i];
81         bool flag = judge(pattern, number);
82         if (flag) {
83             res.push_back(pattern);
84         }
85     }
86     if (res.empty()) {
87         cout << "null";
88         return 0;
89     }
90     // 字典序升序
91     sort(res.begin(), res.end());
92     int m = res.size();
93     for (int i = 0; i < m; i++) {

```

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

## JAVA

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

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

## Python



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

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

## JavaScript

```
1  const rl = require('readline').createInterface({
2    input: process.stdin,
3    output: process.stdout
4  });
5
6  let lines = [];
7  rl.on('line', line => {
8    lines.push(line);
9    if (lines.length === 2) {
10      const settings = lines[0].split(',');
11      const number = lines[1];
12      const res = [];
13
14      // 匹配函数
15      function judge(pattern, str) {
16        if (pattern === '*') return true;
17        let n = str.length, m = pattern.length;
18        let i = 0;
19        for (; i < n && i < m; i++) {
20          if (pattern[i] === str[i]) continue;
21          else if (pattern[i] === '?') {
22            // 偶数位置不能匹配又不相同
23            if (i % 2 === 0) return false;
24          } else if (pattern[i] === '*') {
25            // 达到pattern的末尾并且为*肯定可以匹配
26            if (i === m - 1) return true;
27            const suffix = pattern.slice(i + 1);
28            // 后缀匹配
29            let pos = suffix.length - 1;
30            for (let j = n - 1; j >= i && pos >= 0; j--, pos--) {
31              if (suffix[pos] === str[j]) continue;
32              else if (suffix[pos] === '?') {
33                if (j % 2 === 0) break;
34              } else break;
35            }
36            return pos < 0;
37          } else return false;
38        }
39        return m === n;
40      }
41
42      for (const pattern of settings) {
43        if (judge(pattern, number)) {
44          res.push(pattern);
45        }
46      }
47    }
48  });
```

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

**Go**

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

```

46         break
47     }
48 }
49     return pos < 0
50 } else {
51     return false
52 }
53     i++
54 }
55     return m == n
56 }
57
58 func main() {
59     scanner := bufio.NewScanner(os.Stdin)
60     scanner.Scan()
61     settings := strings.Split(scanner.Text(), ",")
62     scanner.Scan()
63     number := scanner.Text()
64
65     var res []string
66     for _, pattern := range settings {
67         if judge(pattern, number) {
68             res = append(res, pattern)
69         }
70     }
71
72     if len(res) == 0 {
73         fmt.Println("null")
74     } else {
75         sort.Strings(res)
76         fmt.Println(strings.Join(res, ","))
77     }
78 }

```

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

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