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 ⩽ N ⩽ 100000）
- L表示子数组的最小长度（1 ⩽ L ⩽ N）

之后N行表示numbers中的N个数，每个一行（10^-9 ⩽ numbers[i] ⩽ 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

```
3 2
2
2
3
```

输出

```
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

```
10 2
0.2
0.1
0.2
0.2
0.2
0.1
0.2
0.2
0.2
0.2
```

输出

```
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`(ai*ai+1...+ak)^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

```
logx + logy = log(x*y) => 能够将乘法转换为加法
1/nlog(x*y) = log(x*y)^1/n => 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`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++

Plain Text

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

// 判断是否更大
bool isBetter(double sum, int len, double best_sum, int best_len, int start, int best_start) {
    // sum/len = best_sum / bestLen => sum*best_Len = best_sum * len
    double lhs = sum * best_len;
    double rhs = best_sum * len;
    // 考虑到误差
    if (lhs > rhs + 1e-10) return true;
    if (abs(lhs - rhs) <= 1e-10) {
        if (len < best_len) return true;
        if (len == best_len && start < best_start) return true;
    }
    return false;
}


int main() {
    int N, L;
    cin >> N >> L;
    vector<double> nums(N);
    for (int i = 0; i < N; ++i) {
        double x;
        cin >> x;
        nums[i] = log(x);  // 转换为对数，几何平均值变为算术平均值
    }

    // 构建前缀和
    vector<double> prefix(N + 1, 0.0);
    for (int i = 0; i < N; ++i) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    int best_start = 0;
    int best_len = L;
    double best_sum = prefix[L] - prefix[0];


    // 枚举起点和终点
```

```
    for (int start = 0; start < N; ++start) {
        for (int end = start + L; end <= N; ++end) {
            int len = end - start;
            double sum = prefix[end] - prefix[start];

            if (isBetter(sum, len, best_sum, best_len, start, best_start)) {
                best_start = start;
                best_len = len;
                best_sum = sum;
            }
            }
    }

    cout << best_start << " " << best_len << endl;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;
import java.io.*;

public class Main {
    static boolean isBetter(double sum, int len, double bestSum, int bestLen, int start, int bestStart) {
        double lhs = sum * bestLen;
        double rhs = bestSum * len;
        if (lhs > rhs + 1e-10) return true;
        // 考虑误差 lne10-9 = 1e-10
        if (Math.abs(lhs - rhs) <= 1e-10) {
            if (len < bestLen) return true;
            if (len == bestLen && start < bestStart) return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int L = sc.nextInt();
        double[] nums = new double[N];
        for (int i = 0; i < N; i++) {
            double x = sc.nextDouble();
            nums[i] = Math.log(x); // 转换为对数，几何平均值变为算术平均值
        }

        double[] prefix = new double[N + 1];
        for (int i = 0; i < N; i++) {
            prefix[i + 1] = prefix[i] + nums[i];
        }

        int bestStart = 0;
        int bestLen = L;
        double bestSum = prefix[L] - prefix[0];

        // 枚举起点和终点
        for (int start = 0; start < N; start++) {
            for (int end = start + L; end <= N; end++) {
                int len = end - start;
                double sum = prefix[end] - prefix[start];
                if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
                    bestStart = start;
                    bestLen = len;
```

```
                        bestSum = sum;
                    }
                }
            }

            System.out.println(bestStart + " " + bestLen);
        }
    }
```

Python

Plain Text

```
import sys
import math

# 判断最大值
def is_better(sum_, length, best_sum, best_length, start, best_start):
    # 除法转换为乘法
    lhs = sum_ * best_length
    rhs = best_sum * length
    # 考虑误差
    if lhs > rhs + 1e-10:
        return True

    if abs(lhs - rhs) <= 1e-10:
        if length < best_length:
            return True
        if length == best_length and start < best_start:
            return True
    return False

def main():
    N, L = map(int, sys.stdin.readline().split())\
    # 转换为对数
    nums = [math.log(float(sys.stdin.readline())) for _ in range(N)]

    prefix = [0.0] * (N + 1)
    # 前缀和
    for i in range(N):
        prefix[i + 1] = prefix[i] + nums[i]

    best_start = 0
    best_len = L
    best_sum = prefix[L] - prefix[0]
    # 枚举终点起点
    for start in range(N):
        for end in range(start + L, N + 1):
            length = end - start
            sum_ = prefix[end] - prefix[start]
            if is_better(sum_, length, best_sum, best_len, start, best_sta
rt):
                best_start = start
                best_len = length
                best_sum = sum_

    print(f"{best_start} {best_len}")

```

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

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const inputLines = [];
rl.on('line', line => {
    inputLines.push(line.trim());
});

rl.on('close', () => {
    const [N, L] = inputLines[0].split(' ').map(Number);
    const nums = inputLines.slice(1, N + 1).map(x => Math.log(parseFloat(x)));

    // 构建前缀和
    const prefix = new Array(N + 1).fill(0);
    for (let i = 0; i < N; i++) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    let bestStart = 0;
    let bestLen = L;
    let bestSum = prefix[L] - prefix[0];

    // 判断是否更好
    function isBetter(sum, len, bestSum, bestLen, start, bestStart) {
        // 除法转换为乘法
        const lhs = sum * bestLen;
        const rhs = bestSum * len;
        if (lhs > rhs + 1e-10) return true;
        if (Math.abs(lhs - rhs) <= 1e-10) {
            if (len < bestLen) return true;
            if (len === bestLen && start < bestStart) return true;
        }
        return false;
    }

    // 枚举所有子数组
    for (let start = 0; start < N; start++) {
        for (let end = start + L; end <= N; end++) {
            const len = end - start;
            const sum = prefix[end] - prefix[start];
```

```
            if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
                bestStart = start;
                bestLen = len;
                bestSum = sum;
            }
        }
    }

    console.log(`${bestStart} ${bestLen}`);
});
```

Go

Plain Text

```
package main

import (
    "bufio"
    "fmt"
    "math"
    "os"
    "strconv"
)

func isBetter(sum float64, len int, bestSum float64, bestLen int, start int, bestStart int) bool {
    // 除法转换乘法
    lhs := sum * float64(bestLen)
    rhs := bestSum * float64(len)
    // 考虑误差
    if lhs > rhs+1e-10 {
        return true
    }
    if math.Abs(lhs-rhs) <= 1e-10 {
        if len < bestLen {
            return true
        }
        if len == bestLen && start < bestStart {
            return true
        }
    }
    return false
}

func main() {
    reader := bufio.NewScanner(os.Stdin)
    reader.Scan()
    parts := splitInts(reader.Text())
    N, L := parts[0], parts[1]

    nums := make([]float64, N)
    for i := 0; i < N; i++ {
        reader.Scan()
        f, _ := strconv.ParseFloat(reader.Text(), 64)
        nums[i] = math.Log(f) // 转换为对数
    }

    prefix := make([]float64, N+1)
    // 前缀和
```

```
    for i := 0; i < N; i++ {
        prefix[i+1] = prefix[i] + nums[i]
    }

    bestStart := 0
    bestLen := L
    bestSum := prefix[L] - prefix[0]
    // 枚举起点和终点
    for start := 0; start < N; start++ {
        for end := start + L; end <= N; end++ {
            length := end - start
            sum := prefix[end] - prefix[start]
            if isBetter(sum, length, bestSum, bestLen, start, bestStart) {
                bestStart = start
                bestLen = length
                bestSum = sum
            }
        }
    }

    fmt.Printf("%d %d\n", bestStart, bestLen)
}

func splitInts(s string) []int {
    fields := make([]int, 0)
    for _, part := range splitStr(s) {
        v, _ := strconv.Atoi(part)
        fields = append(fields, v)
    }
    return fields
}

func splitStr(s string) []string {
    result := []string{}
    current := ""
    for _, ch := range s {
        if ch == ' ' || ch == '\t' {
            if current != "" {
                result = append(result, current)
                current = ""
            }
        } else {
            current += string(ch)
        }
    }
    if current != "" {
        result = append(result, current)
    }
```

```
        return result
    }
```

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 ⩽ N ⩽ 100000）
- L表示子数组的最小长度（1 ⩽ L ⩽ N）

之后N行表示numbers中的N个数，每个一行（10^-9 ⩽ numbers[i] ⩽ 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

```
3 2
2
2
3
```

输出

```
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

```
10 2
0.2
0.1
0.2
0.2
0.2
0.1
0.2
0.2
0.2
0.2
```

输出

```
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 `(ai*ai+1...+ak)^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  logx + logy = log(x*y) => 能够将乘法转换为加法
2  1/nlog(x*y) = log(x*y)^1/n => 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`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++

Plain Text

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

// 判断是否更大
bool isBetter(double sum, int len, double best_sum, int best_len, int start, int best_start) {
    // sum/len = best_sum / bestLen => sum*best_Len = best_sum * len
    double lhs = sum * best_len;
    double rhs = best_sum * len;
    // 考虑到误差
    if (lhs > rhs + 1e-10) return true;
    if (abs(lhs - rhs) <= 1e-10) {
        if (len < best_len) return true;
        if (len == best_len && start < best_start) return true;
    }
    return false;
}


int main() {
    int N, L;
    cin >> N >> L;
    vector<double> nums(N);
    for (int i = 0; i < N; ++i) {
        double x;
        cin >> x;
        nums[i] = log(x);  // 转换为对数，几何平均值变为算术平均值
    }

    // 构建前缀和
    vector<double> prefix(N + 1, 0.0);
    for (int i = 0; i < N; ++i) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    int best_start = 0;
    int best_len = L;
    double best_sum = prefix[L] - prefix[0];


    // 枚举起点和终点
```

```
    for (int start = 0; start < N; ++start) {
        for (int end = start + L; end <= N; ++end) {
            int len = end - start;
            double sum = prefix[end] - prefix[start];

            if (isBetter(sum, len, best_sum, best_len, start, best_start)) {
                best_start = start;
                best_len = len;
                best_sum = sum;
            }
            }
    }

    cout << best_start << " " << best_len << endl;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;
import java.io.*;

public class Main {
    static boolean isBetter(double sum, int len, double bestSum, int bestLen, int start, int bestStart) {
        double lhs = sum * bestLen;
        double rhs = bestSum * len;
        if (lhs > rhs + 1e-10) return true;
        // 考虑误差 lne10-9 = 1e-10
        if (Math.abs(lhs - rhs) <= 1e-10) {
            if (len < bestLen) return true;
            if (len == bestLen && start < bestStart) return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int L = sc.nextInt();
        double[] nums = new double[N];
        for (int i = 0; i < N; i++) {
            double x = sc.nextDouble();
            nums[i] = Math.log(x); // 转换为对数，几何平均值变为算术平均值
        }

        double[] prefix = new double[N + 1];
        for (int i = 0; i < N; i++) {
            prefix[i + 1] = prefix[i] + nums[i];
        }

        int bestStart = 0;
        int bestLen = L;
        double bestSum = prefix[L] - prefix[0];

        // 枚举起点和终点
        for (int start = 0; start < N; start++) {
            for (int end = start + L; end <= N; end++) {
                int len = end - start;
                double sum = prefix[end] - prefix[start];
                if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
                    bestStart = start;
                    bestLen = len;
```

```
                        bestSum = sum;
                    }
                }
            }

            System.out.println(bestStart + " " + bestLen);
        }
    }
```

Python

```
import sys
import math

# 判断最大值
def is_better(sum_, length, best_sum, best_length, start, best_start):
    # 除法转换为乘法
    lhs = sum_ * best_length
    rhs = best_sum * length
    # 考虑误差
    if lhs > rhs + 1e-10:
        return True

    if abs(lhs - rhs) <= 1e-10:
        if length < best_length:
            return True
        if length == best_length and start < best_start:
            return True
    return False

def main():
    N, L = map(int, sys.stdin.readline().split())\
    # 转换为对数
    nums = [math.log(float(sys.stdin.readline())) for _ in range(N)]

    prefix = [0.0] * (N + 1)
    # 前缀和
    for i in range(N):
        prefix[i + 1] = prefix[i] + nums[i]

    best_start = 0
    best_len = L
    best_sum = prefix[L] - prefix[0]
    # 枚举终点起点
    for start in range(N):
        for end in range(start + L, N + 1):
            length = end - start
            sum_ = prefix[end] - prefix[start]
            if is_better(sum_, length, best_sum, best_len, start, best_sta
rt):
                best_start = start
                best_len = length
                best_sum = sum_

    print(f"{best_start} {best_len}")

```

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

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const inputLines = [];
rl.on('line', line => {
    inputLines.push(line.trim());
});

rl.on('close', () => {
    const [N, L] = inputLines[0].split(' ').map(Number);
    const nums = inputLines.slice(1, N + 1).map(x => Math.log(parseFloat(x)));

    // 构建前缀和
    const prefix = new Array(N + 1).fill(0);
    for (let i = 0; i < N; i++) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    let bestStart = 0;
    let bestLen = L;
    let bestSum = prefix[L] - prefix[0];

    // 判断是否更好
    function isBetter(sum, len, bestSum, bestLen, start, bestStart) {
        // 除法转换为乘法
        const lhs = sum * bestLen;
        const rhs = bestSum * len;
        if (lhs > rhs + 1e-10) return true;
        if (Math.abs(lhs - rhs) <= 1e-10) {
            if (len < bestLen) return true;
            if (len === bestLen && start < bestStart) return true;
        }
        return false;
    }

    // 枚举所有子数组
    for (let start = 0; start < N; start++) {
        for (let end = start + L; end <= N; end++) {
            const len = end - start;
            const sum = prefix[end] - prefix[start];
```

```
            if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
                bestStart = start;
                bestLen = len;
                bestSum = sum;
            }
        }
    }

    console.log(`${bestStart} ${bestLen}`);
});
```

Go

Plain Text

```
package main

import (
    "bufio"
    "fmt"
    "math"
    "os"
    "strconv"
)

func isBetter(sum float64, len int, bestSum float64, bestLen int, start int, bestStart int) bool {
    // 除法转换乘法
    lhs := sum * float64(bestLen)
    rhs := bestSum * float64(len)
    // 考虑误差
    if lhs > rhs+1e-10 {
        return true
    }
    if math.Abs(lhs-rhs) <= 1e-10 {
        if len < bestLen {
            return true
        }
        if len == bestLen && start < bestStart {
            return true
        }
    }
    return false
}

func main() {
    reader := bufio.NewScanner(os.Stdin)
    reader.Scan()
    parts := splitInts(reader.Text())
    N, L := parts[0], parts[1]

    nums := make([]float64, N)
    for i := 0; i < N; i++ {
        reader.Scan()
        f, _ := strconv.ParseFloat(reader.Text(), 64)
        nums[i] = math.Log(f) // 转换为对数
    }

    prefix := make([]float64, N+1)
    // 前缀和
```

```
    for i := 0; i < N; i++ {
        prefix[i+1] = prefix[i] + nums[i]
    }

    bestStart := 0
    bestLen := L
    bestSum := prefix[L] - prefix[0]
    // 枚举起点和终点
    for start := 0; start < N; start++ {
        for end := start + L; end <= N; end++ {
            length := end - start
            sum := prefix[end] - prefix[start]
            if isBetter(sum, length, bestSum, bestLen, start, bestStart) {
                bestStart = start
                bestLen = length
                bestSum = sum
            }
        }
    }

    fmt.Printf("%d %d\n", bestStart, bestLen)
}

func splitInts(s string) []int {
    fields := make([]int, 0)
    for _, part := range splitStr(s) {
        v, _ := strconv.Atoi(part)
        fields = append(fields, v)
    }
    return fields
}

func splitStr(s string) []string {
    result := []string{}
    current := ""
    for _, ch := range s {
        if ch == ' ' || ch == '\t' {
            if current != "" {
                result = append(result, current)
                current = ""
            }
        } else {
            current += string(ch)
        }
    }
    if current != "" {
        result = append(result, current)
    }
```

```
        return result
    }
```

> 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 ≤ N ≤ 100000）
- L表示子数组的最小长度（1 ≤ L ≤ N）

之后N行表示numbers中的N个数，每个一行（10^-9 ≤ numbers[i] ≤ 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

```
3 2
2
2
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 `(ai*ai+1...+ak)^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

```
logx + logy = log(x*y) => 能够将乘法转换为加法
1/nlog(x*y) = log(x*y)^1/n => 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` 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++

Plain Text

```
#include <iostream>
#include <vector>
#include <cmath>
#include <limits>

using namespace std;

// 判断是否更大
bool isBetter(double sum, int len, double best_sum, int best_len, int start, int best_start) {
    // sum/len = best_sum / bestLen => sum*best_Len = best_sum * len
    double lhs = sum * best_len;
    double rhs = best_sum * len;
    // 考虑到误差
    if (lhs > rhs + 1e-10) return true;
    if (abs(lhs - rhs) <= 1e-10) {
        if (len < best_len) return true;
        if (len == best_len && start < best_start) return true;
    }
    return false;
}


int main() {
    int N, L;
    cin >> N >> L;
    vector<double> nums(N);
    for (int i = 0; i < N; ++i) {
        double x;
        cin >> x;
        nums[i] = log(x);  // 转换为对数，几何平均值变为算术平均值
    }

    // 构建前缀和
    vector<double> prefix(N + 1, 0.0);
    for (int i = 0; i < N; ++i) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    int best_start = 0;
    int best_len = L;
    double best_sum = prefix[L] - prefix[0];


    // 枚举起点和终点
```

```
    for (int start = 0; start < N; ++start) {
        for (int end = start + L; end <= N; ++end) {
            int len = end - start;
            double sum = prefix[end] - prefix[start];

            if (isBetter(sum, len, best_sum, best_len, start, best_start)) {
                best_start = start;
                best_len = len;
                best_sum = sum;
            }
            }
    }

    cout << best_start << " " << best_len << endl;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;
import java.io.*;

public class Main {
    static boolean isBetter(double sum, int len, double bestSum, int bestLen, int start, int bestStart) {
        double lhs = sum * bestLen;
        double rhs = bestSum * len;
        if (lhs > rhs + 1e-10) return true;
        // 考虑误差 lne10-9 = 1e-10
        if (Math.abs(lhs - rhs) <= 1e-10) {
            if (len < bestLen) return true;
            if (len == bestLen && start < bestStart) return true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int N = sc.nextInt();
        int L = sc.nextInt();
        double[] nums = new double[N];
        for (int i = 0; i < N; i++) {
            double x = sc.nextDouble();
            nums[i] = Math.log(x); // 转换为对数，几何平均值变为算术平均值
        }

        double[] prefix = new double[N + 1];
        for (int i = 0; i < N; i++) {
            prefix[i + 1] = prefix[i] + nums[i];
        }

        int bestStart = 0;
        int bestLen = L;
        double bestSum = prefix[L] - prefix[0];

        // 枚举起点和终点
        for (int start = 0; start < N; start++) {
            for (int end = start + L; end <= N; end++) {
                int len = end - start;
                double sum = prefix[end] - prefix[start];
                if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
                    bestStart = start;
                    bestLen = len;
```

```
                        bestSum = sum;
                    }
                }
            }

            System.out.println(bestStart + " " + bestLen);
        }
    }
```

Python

```
import sys
import math

# 判断最大值
def is_better(sum_, length, best_sum, best_length, start, best_start):
    # 除法转换为乘法
    lhs = sum_ * best_length
    rhs = best_sum * length
    # 考虑误差
    if lhs > rhs + 1e-10:
        return True

    if abs(lhs - rhs) <= 1e-10:
        if length < best_length:
            return True
        if length == best_length and start < best_start:
            return True
    return False

def main():
    N, L = map(int, sys.stdin.readline().split())\
    # 转换为对数
    nums = [math.log(float(sys.stdin.readline())) for _ in range(N)]

    prefix = [0.0] * (N + 1)
    # 前缀和
    for i in range(N):
        prefix[i + 1] = prefix[i] + nums[i]

    best_start = 0
    best_len = L
    best_sum = prefix[L] - prefix[0]
    # 枚举终点起点
    for start in range(N):
        for end in range(start + L, N + 1):
            length = end - start
            sum_ = prefix[end] - prefix[start]
            if is_better(sum_, length, best_sum, best_len, start, best_sta
rt):
                best_start = start
                best_len = length
                best_sum = sum_

    print(f"{best_start} {best_len}")

```

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

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const inputLines = [];
rl.on('line', line => {
    inputLines.push(line.trim());
});

rl.on('close', () => {
    const [N, L] = inputLines[0].split(' ').map(Number);
    const nums = inputLines.slice(1, N + 1).map(x => Math.log(parseFloat(x)));

    // 构建前缀和
    const prefix = new Array(N + 1).fill(0);
    for (let i = 0; i < N; i++) {
        prefix[i + 1] = prefix[i] + nums[i];
    }

    let bestStart = 0;
    let bestLen = L;
    let bestSum = prefix[L] - prefix[0];

    // 判断是否更好
    function isBetter(sum, len, bestSum, bestLen, start, bestStart) {
        // 除法转换为乘法
        const lhs = sum * bestLen;
        const rhs = bestSum * len;
        if (lhs > rhs + 1e-10) return true;
        if (Math.abs(lhs - rhs) <= 1e-10) {
            if (len < bestLen) return true;
            if (len === bestLen && start < bestStart) return true;
        }
        return false;
    }

    // 枚举所有子数组
    for (let start = 0; start < N; start++) {
        for (let end = start + L; end <= N; end++) {
            const len = end - start;
            const sum = prefix[end] - prefix[start];
```

```
            if (isBetter(sum, len, bestSum, bestLen, start, bestStart)) {
                bestStart = start;
                bestLen = len;
                bestSum = sum;
            }
        }
    }

    console.log(`${bestStart} ${bestLen}`);
});
```

Go

Plain Text

```
package main

import (
    "bufio"
    "fmt"
    "math"
    "os"
    "strconv"
)

func isBetter(sum float64, len int, bestSum float64, bestLen int, start int, bestStart int) bool {
    // 除法转换乘法
    lhs := sum * float64(bestLen)
    rhs := bestSum * float64(len)
    // 考虑误差
    if lhs > rhs+1e-10 {
        return true
    }
    if math.Abs(lhs-rhs) <= 1e-10 {
        if len < bestLen {
            return true
        }
        if len == bestLen && start < bestStart {
            return true
        }
    }
    return false
}

func main() {
    reader := bufio.NewScanner(os.Stdin)
    reader.Scan()
    parts := splitInts(reader.Text())
    N, L := parts[0], parts[1]

    nums := make([]float64, N)
    for i := 0; i < N; i++ {
        reader.Scan()
        f, _ := strconv.ParseFloat(reader.Text(), 64)
        nums[i] = math.Log(f) // 转换为对数
    }

    prefix := make([]float64, N+1)
    // 前缀和
```

```
    for i := 0; i < N; i++ {
        prefix[i+1] = prefix[i] + nums[i]
    }

    bestStart := 0
    bestLen := L
    bestSum := prefix[L] - prefix[0]
    // 枚举起点和终点
    for start := 0; start < N; start++ {
        for end := start + L; end <= N; end++ {
            length := end - start
            sum := prefix[end] - prefix[start]
            if isBetter(sum, length, bestSum, bestLen, start, bestStart) {
                bestStart = start
                bestLen = length
                bestSum = sum
            }
        }
    }

    fmt.Printf("%d %d\n", bestStart, bestLen)
}

func splitInts(s string) []int {
    fields := make([]int, 0)
    for _, part := range splitStr(s) {
        v, _ := strconv.Atoi(part)
        fields = append(fields, v)
    }
    return fields
}

func splitStr(s string) []string {
    result := []string{}
    current := ""
    for _, ch := range s {
        if ch == ' ' || ch == '\t' {
            if current != "" {
                result = append(result, current)
                current = ""
            }
        } else {
            current += string(ch)
        }
    }
    if current != "" {
        result = append(result, current)
    }
```

```
        return result
    }
```

> 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

```
news.qq.com
news.sina.com.cn
news.qq.com
news.qq.com
game.163.com
game.163.com
www.huawei.com
www.cctv.com
3
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
3
```

输出

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

示例2

输入

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

输出

```
news.qq.com
www.huawei.com,news.qq.com
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++

Plain Text

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <regex>

using namespace std;

string sortURL(int n, map<string, int>& cache);

int main() {
    map<string, int> cache; // 创建一个哈希表，用于存储每个 URL 出现的次数
    string line;
    while (getline(cin, line)) { // 不断读取输入，直到没有下一行
        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
            cout << sortURL(stoi(line), cache) << endl;
            continue;
        }
        cache[line]++;
    }
    return 0;
}

string sortURL(int n ,map<string, int>& cache) {
    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

    sort(list.begin(), list.end(), [](const pair<string, int>& a, const pair<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
        if (a.second != b.second) {
            return a.second > b.second;
        } else {
            return a.first < b.first;
        }
    });

    string res;
    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
        res += list[i].first + ",";
    }
    if (!res.empty()) { // 如果字符串不为空，则删除最后一个逗号
        res.pop_back();
```

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

Java

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Integer> cache = new HashMap<>(); // 哈希表存储 URL 及其出现次数

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine().trim();
            if (line.matches("^\\d+$")) { // 如果输入是纯数字，处理排序并输出结果
                System.out.println(sortURL(Integer.parseInt(line), cache));
                continue;
            }
            cache.put(line, cache.getOrDefault(line, 0) + 1);
        }
        scanner.close();
    }

    private static String sortURL(int n, Map<String, Integer> cache) {
        List<Map.Entry<String, Integer>> list = new ArrayList<>(cache.entrySet());

        // 按出现次数降序排列，次数相同则按字典序升序排列
        list.sort((a, b) -> {
            if (!a.getValue().equals(b.getValue())) {
                return b.getValue() - a.getValue();
            }
            return a.getKey().compareTo(b.getKey());
        });

        // 取前 n 个 URL 并拼接成字符串
        StringBuilder res = new StringBuilder();
        for (int i = 0; i < Math.min(n, list.size()); i++) {
            res.append(list.get(i).getKey()).append(",");
        }
        if (res.length() > 0) {
            res.setLength(res.length() - 1); // 删除末尾的逗号
        }
        return res.toString();
    }
}
```

Python

```
import sys
import re

def sort_url(n, cache):
    # 按出现次数降序排序，次数相同则按字典序升序
    sorted_urls = sorted(cache.items(), key=lambda x: (-x[1], x[0]))

    # 取前 n 个 URL 并拼接成字符串
    return ",".join(url for url, _ in sorted_urls[:n])

def main():
    cache = {}  # 哈希表存储 URL 及其出现次数

    for line in sys.stdin:
        line = line.strip()
        if re.fullmatch(r"\d+", line):  # 如果输入是纯数字，处理排序并输出结果
            print(sort_url(int(line), cache))
            continue
        cache[line] = cache.get(line, 0) + 1

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let cache = {}; // 哈希表存储 URL 及其出现次数
let lines = [];

rl.on("line", (line) => {
    line = line.trim();
    if (/^\d+$/.test(line)) { // 如果输入是纯数字，处理排序并输出结果
        console.log(sortURL(parseInt(line), cache));
        return;
    }
    cache[line] = (cache[line] || 0) + 1;
});

function sortURL(n, cache) {
    // 转换为数组并排序，按次数降序，次数相同则按字典序升序
    let sortedList = Object.entries(cache).sort((a, b) => {
        if (b[1] !== a[1]) {
            return b[1] - a[1];
        }
        return a[0].localeCompare(b[0]);
    });

    // 取前 n 个 URL 并拼接成字符串
    return sortedList.slice(0, n).map(item => item[0]).join(",");
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strconv"
  "strings"
)

func sortURL(n int, cache map[string]int) string {
  // 将哈希表转换为切片
  type urlCount struct {
    url   string
    count int
  }
  list := make([]urlCount, 0, len(cache))
  for url, count := range cache {
    list = append(list, urlCount{url, count})
  }

  // 按出现次数降序，次数相同则按字典序升序
  sort.Slice(list, func(i, j int) bool {
    if list[i].count != list[j].count {
      return list[i].count > list[j].count
    }
    return list[i].url < list[j].url
  })

  // 取前 n 个 URL 并拼接成字符串
  var res []string
  for i := 0; i < n && i < len(list); i++ {
    res = append(res, list[i].url)
  }
  return strings.Join(res, ",")
}

func main() {
  cache := make(map[string]int) // 哈希表存储 URL 及其出现次数
  scanner := bufio.NewScanner(os.Stdin)

  for scanner.Scan() {
    line := strings.TrimSpace(scanner.Text())

```

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

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

```
news.qq.com
news.sina.com.cn
news.qq.com
news.qq.com
game.163.com
game.163.com
www.huawei.com
www.cctv.com
3
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
3
```

输出

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

示例2

输入

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

输出

```
news.qq.com
www.huawei.com,news.qq.com
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++

Plain Text

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <regex>

using namespace std;

string sortURL(int n, map<string, int>& cache);

int main() {
    map<string, int> cache; // 创建一个哈希表，用于存储每个 URL 出现的次数
    string line;
    while (getline(cin, line)) { // 不断读取输入，直到没有下一行
        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
            cout << sortURL(stoi(line), cache) << endl;
            continue;
        }
        cache[line]++;
    }
    return 0;
}

string sortURL(int n ,map<string, int>& cache) {
    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

    sort(list.begin(), list.end(), [](const pair<string, int>& a, const pair<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
        if (a.second != b.second) {
            return a.second > b.second;
        } else {
            return a.first < b.first;
        }
    });

    string res;
    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
        res += list[i].first + ",";
    }
    if (!res.empty()) { // 如果字符串不为空，则删除最后一个逗号
        res.pop_back();
```

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

Java

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Integer> cache = new HashMap<>(); // 哈希表存储 URL 及其出现次数

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine().trim();
            if (line.matches("^\\d+$")) { // 如果输入是纯数字，处理排序并输出结果
                System.out.println(sortURL(Integer.parseInt(line), cache));
                continue;
            }
            cache.put(line, cache.getOrDefault(line, 0) + 1);
        }
        scanner.close();
    }

    private static String sortURL(int n, Map<String, Integer> cache) {
        List<Map.Entry<String, Integer>> list = new ArrayList<>(cache.entrySet());

        // 按出现次数降序排列，次数相同则按字典序升序排列
        list.sort((a, b) -> {
            if (!a.getValue().equals(b.getValue())) {
                return b.getValue() - a.getValue();
            }
            return a.getKey().compareTo(b.getKey());
        });

        // 取前 n 个 URL 并拼接成字符串
        StringBuilder res = new StringBuilder();
        for (int i = 0; i < Math.min(n, list.size()); i++) {
            res.append(list.get(i).getKey()).append(",");
        }
        if (res.length() > 0) {
            res.setLength(res.length() - 1); // 删除末尾的逗号
        }
        return res.toString();
    }
}
```

Python

```
import sys
import re

def sort_url(n, cache):
    # 按出现次数降序排序，次数相同则按字典序升序
    sorted_urls = sorted(cache.items(), key=lambda x: (-x[1], x[0]))

    # 取前 n 个 URL 并拼接成字符串
    return ",".join(url for url, _ in sorted_urls[:n])

def main():
    cache = {}  # 哈希表存储 URL 及其出现次数

    for line in sys.stdin:
        line = line.strip()
        if re.fullmatch(r"\d+", line):  # 如果输入是纯数字，处理排序并输出结果
            print(sort_url(int(line), cache))
            continue
        cache[line] = cache.get(line, 0) + 1

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let cache = {}; // 哈希表存储 URL 及其出现次数
let lines = [];

rl.on("line", (line) => {
    line = line.trim();
    if (/^\d+$/.test(line)) { // 如果输入是纯数字，处理排序并输出结果
        console.log(sortURL(parseInt(line), cache));
        return;
    }
    cache[line] = (cache[line] || 0) + 1;
});

function sortURL(n, cache) {
    // 转换为数组并排序，按次数降序，次数相同则按字典序升序
    let sortedList = Object.entries(cache).sort((a, b) => {
        if (b[1] !== a[1]) {
            return b[1] - a[1];
        }
        return a[0].localeCompare(b[0]);
    });

    // 取前 n 个 URL 并拼接成字符串
    return sortedList.slice(0, n).map(item => item[0]).join(",");
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strconv"
  "strings"
)

func sortURL(n int, cache map[string]int) string {
  // 将哈希表转换为切片
  type urlCount struct {
    url   string
    count int
  }
  list := make([]urlCount, 0, len(cache))
  for url, count := range cache {
    list = append(list, urlCount{url, count})
  }

  // 按出现次数降序，次数相同则按字典序升序
  sort.Slice(list, func(i, j int) bool {
    if list[i].count != list[j].count {
      return list[i].count > list[j].count
    }
    return list[i].url < list[j].url
  })

  // 取前 n 个 URL 并拼接成字符串
  var res []string
  for i := 0; i < n && i < len(list); i++ {
    res = append(res, list[i].url)
  }
  return strings.Join(res, ",")
}

func main() {
  cache := make(map[string]int) // 哈希表存储 URL 及其出现次数
  scanner := bufio.NewScanner(os.Stdin)

  for scanner.Scan() {
    line := strings.TrimSpace(scanner.Text())

```

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

> 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

```
news.qq.com
news.sina.com.cn
news.qq.com
news.qq.com
game.163.com
game.163.com
www.huawei.com
www.cctv.com
3
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
www.cctv.com
www.huawei.com
3
```

输出

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

示例2

输入

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

输出

```
news.qq.com
www.huawei.com,news.qq.com
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++

Plain Text

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <regex>

using namespace std;

string sortURL(int n, map<string, int>& cache);

int main() {
    map<string, int> cache; // 创建一个哈希表，用于存储每个 URL 出现的次数
    string line;
    while (getline(cin, line)) { // 不断读取输入，直到没有下一行
        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
            cout << sortURL(stoi(line), cache) << endl;
            continue;
        }
        cache[line]++;
    }
    return 0;
}

string sortURL(int n ,map<string, int>& cache) {
    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

    sort(list.begin(), list.end(), [](const pair<string, int>& a, const pair<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
        if (a.second != b.second) {
            return a.second > b.second;
        } else {
            return a.first < b.first;
        }
    });

    string res;
    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
        res += list[i].first + ",";
    }
    if (!res.empty()) { // 如果字符串不为空，则删除最后一个逗号
        res.pop_back();
```

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

Java

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Map<String, Integer> cache = new HashMap<>(); // 哈希表存储 URL 及其出现次数

        while (scanner.hasNextLine()) {
            String line = scanner.nextLine().trim();
            if (line.matches("^\\d+$")) { // 如果输入是纯数字，处理排序并输出结果
                System.out.println(sortURL(Integer.parseInt(line), cache));
                continue;
            }
            cache.put(line, cache.getOrDefault(line, 0) + 1);
        }
        scanner.close();
    }

    private static String sortURL(int n, Map<String, Integer> cache) {
        List<Map.Entry<String, Integer>> list = new ArrayList<>(cache.entrySet());

        // 按出现次数降序排列，次数相同则按字典序升序排列
        list.sort((a, b) -> {
            if (!a.getValue().equals(b.getValue())) {
                return b.getValue() - a.getValue();
            }
            return a.getKey().compareTo(b.getKey());
        });

        // 取前 n 个 URL 并拼接成字符串
        StringBuilder res = new StringBuilder();
        for (int i = 0; i < Math.min(n, list.size()); i++) {
            res.append(list.get(i).getKey()).append(",");
        }
        if (res.length() > 0) {
            res.setLength(res.length() - 1); // 删除末尾的逗号
        }
        return res.toString();
    }
}
```

Python

```
import sys
import re

def sort_url(n, cache):
    # 按出现次数降序排序，次数相同则按字典序升序
    sorted_urls = sorted(cache.items(), key=lambda x: (-x[1], x[0]))

    # 取前 n 个 URL 并拼接成字符串
    return ",".join(url for url, _ in sorted_urls[:n])

def main():
    cache = {}  # 哈希表存储 URL 及其出现次数

    for line in sys.stdin:
        line = line.strip()
        if re.fullmatch(r"\d+", line):  # 如果输入是纯数字，处理排序并输出结果
            print(sort_url(int(line), cache))
            continue
        cache[line] = cache.get(line, 0) + 1

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

JavaScript

Plain Text

```
const readline = require("readline");

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let cache = {}; // 哈希表存储 URL 及其出现次数
let lines = [];

rl.on("line", (line) => {
    line = line.trim();
    if (/^\d+$/.test(line)) { // 如果输入是纯数字，处理排序并输出结果
        console.log(sortURL(parseInt(line), cache));
        return;
    }
    cache[line] = (cache[line] || 0) + 1;
});

function sortURL(n, cache) {
    // 转换为数组并排序，按次数降序，次数相同则按字典序升序
    let sortedList = Object.entries(cache).sort((a, b) => {
        if (b[1] !== a[1]) {
            return b[1] - a[1];
        }
        return a[0].localeCompare(b[0]);
    });

    // 取前 n 个 URL 并拼接成字符串
    return sortedList.slice(0, n).map(item => item[0]).join(",");
}
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strconv"
  "strings"
)

func sortURL(n int, cache map[string]int) string {
  // 将哈希表转换为切片
  type urlCount struct {
    url   string
    count int
  }
  list := make([]urlCount, 0, len(cache))
  for url, count := range cache {
    list = append(list, urlCount{url, count})
  }

  // 按出现次数降序，次数相同则按字典序升序
  sort.Slice(list, func(i, j int) bool {
    if list[i].count != list[j].count {
      return list[i].count > list[j].count
    }
    return list[i].url < list[j].url
  })

  // 取前 n 个 URL 并拼接成字符串
  var res []string
  for i := 0; i < n && i < len(list); i++ {
    res = append(res, list[i].url)
  }
  return strings.Join(res, ",")
}

func main() {
  cache := make(map[string]int) // 哈希表存储 URL 及其出现次数
  scanner := bufio.NewScanner(os.Stdin)

  for scanner.Scan() {
    line := strings.TrimSpace(scanner.Text())

```

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

> 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

```
4 1 3 5 2
```

输出

```
4 1 5 2 3
```

示例2

输入

```
1 1 1 1 1
```

输出

```
1 1 1 1 1
```

示例3

输入

```
xxx
```

输出

```
[ ]
```

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应把前面的人排好，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++

Plain Text

```
#include <cctype>
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
using namespace std;

// 解析字符串到int
int parseStringToInt(string s) {
    int res = 0;
    for (int i = 0; i < s.size();i++) {
        if ( '0' <=s[i] && s[i] <= '9') {
            res = res * 10 + s[i] -'0';
        } else {
            return -1;
        }
    }
    return res;
}


int main() {
    vector<int> ans;
    string s;
    while (cin >> s) {
        if (s == "") {
            break;
        }
        int tmp = parseStringToInt(s);
        // 出现不合法数据
        if (tmp == -1) {
            cout << "[ ]";
            return 0;
        }
        ans.push_back(tmp);
    }

    // 贪心算法
    for (int i = 1;  i < ans.size(); i++) {
        if (ans[i] > ans[i-1] && (i % 2)) {
            swap(ans[i], ans[i-1]);
        }

```

```
            if (ans[i] < ans[i-1] && !(i % 2)) {
                swap(ans[i], ans[i-1]);
            }
        }
        for (int i =0 ;i <ans.size(); i++) {
            cout << ans[i] << " ";
        }
        return 0;
    }
```

Java

Plain Text

```
import java.util.*;

public class Main {
    // 解析字符串到 int
    public static int parseStringToInt(String s) {
        int res = 0;
        for (char c : s.toCharArray()) {
            if (Character.isDigit(c)) {
                res = res * 10 + (c - '0');
            } else {
                return -1; // 非数字字符，返回 -1
            }
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        List<Integer> ans = new ArrayList<>();

        while (scanner.hasNext()) {
            String s = scanner.next();
            int tmp = parseStringToInt(s);
            if (tmp == -1) {
                System.out.println("[ ]");
                scanner.close();
                return;
            }
            ans.add(tmp);
        }
        scanner.close();

        // 贪心算法
        for (int i = 1; i < ans.size(); i++) {
            if (ans.get(i) > ans.get(i - 1) && (i % 2 == 1)) {
                Collections.swap(ans, i, i - 1);
            }
            if (ans.get(i) < ans.get(i - 1) && (i % 2 == 0)) {
                Collections.swap(ans, i, i - 1);
            }
        }

        for (int i = 0; i < ans.size(); i++) {
            if (i > 0) System.out.print(" ");
            System.out.print(ans.get(i));
```

```
        }
    }
}
```

Python

Plain Text

```
import sys

# 解析字符串到 int
def parse_string_to_int(s):
    if s.isdigit():
        return int(s)
    return -1  # 非数字字符返回 -1

def main():
    # 读取整行输入
    line = sys.stdin.readline().strip()

    # 处理空输入
    if not line:
        print("[ ]")
        return

    # 分割输入并解析
    ans = []
    for s in line.split():
        tmp = parse_string_to_int(s)
        if tmp == -1:
            print("[ ]")
            return
        ans.append(tmp)

    # 贪心算法
    for i in range(1, len(ans)):
        if ans[i] > ans[i - 1] and (i % 2 == 1):
            ans[i], ans[i - 1] = ans[i - 1], ans[i]
        if ans[i] < ans[i - 1] and (i % 2 == 0):
            ans[i], ans[i - 1] = ans[i - 1], ans[i]

    print(" ".join(map(str, ans)))

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

JavaScript

Plain Text

```
const readline = require('readline');

// 解析字符串到 int
function parseStringToInt(s) {
    if (/^\d+$/.test(s)) {
        return parseInt(s, 10);
    }
    return -1; // 非数字字符返回 -1
}

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

rl.on('line', (line) => {
    let tokens = line.trim().split(/\s+/);

    let ans = [];
    for (let s of tokens) {
        let num = parseStringToInt(s);
        if (num === -1) {
            console.log("[ ]");
            rl.close();
            return;
        }
        ans.push(num);
    }

    // 贪心算法
    for (let i = 1; i < ans.length; i++) {
        if (ans[i] > ans[i - 1] && (i % 2 === 1)) {
            [ans[i], ans[i - 1]] = [ans[i - 1], ans[i]];
        }
        if (ans[i] < ans[i - 1] && (i % 2 === 0)) {
            [ans[i], ans[i - 1]] = [ans[i - 1], ans[i]];
        }
    }

    console.log(ans.join(" "));
    rl.close();
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

// 解析字符串到 int
func parseStringToInt(s string) (int, bool) {
  num, err := strconv.Atoi(s)
  if err != nil {
    return -1, false // 解析失败返回 -1，并标记为非法输入
  }
  return num, true
}

func main() {
  // 读取整行输入
  scanner := bufio.NewScanner(os.Stdin)
  if !scanner.Scan() {
    fmt.Println("[ ]")
    return
  }
  line := strings.TrimSpace(scanner.Text())

  // 处理空输入
  if line == "" {
    fmt.Println("[ ]")
    return
  }

  // 解析输入
  tokens := strings.Fields(line)
  ans := make([]int, 0, len(tokens))

  for _, s := range tokens {
    num, valid := parseStringToInt(s)
    if !valid {
      fmt.Println("[ ]")
      return
    }
    ans = append(ans, num)
```

```
    }

    // 贪心算法
    for i := 1; i < len(ans); i++ {
      if ans[i] > ans[i-1] && (i%2 == 1) {
        ans[i], ans[i-1] = ans[i-1], ans[i]
      }
      if ans[i] < ans[i-1] && (i%2 == 0) {
        ans[i], ans[i-1] = ans[i-1], ans[i]
      }
    }

    // 输出结果
    fmt.Println(strings.Trim(fmt.Sprint(ans), "[]"))
}
```

> 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（X1,X2,X3,...,Xn），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
```

输出

```
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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<set>
using namespace std;

// 通用 split 函数
vector<int> split(const string& str, const string& delimiter) {
    vector<int> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(stoi(str.substr(start, end - start)));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(stoi(str.substr(start)));
    return result;
}

// 找到组中的代表人
int find(int a, vector<int> &ans) {
    if (ans[a] != a) {
        ans[a] = find(ans[a], ans);
    }
    return ans[a];
}

// 合并两个组
void merge(int a, int b, vector<int> &ans) {
    int rootA = find(a, ans);
    int rootB = find(b, ans);
    int newRoot = min(rootA, rootB);
    ans[rootA] = newRoot;
    ans[rootB] = newRoot;
}


```

```
int main() {
    int n;
    cin >> n;

    string confirm;
    // 忽略换行符
    cin.ignore();
    getline(cin, confirm);
    if (confirm.empty()) {
        cout << 0;
        return 0;
    }
    vector<int> confirmId = split(confirm, ",");
    vector<vector<int>> grid(n);
    for (int i = 0; i < n; i++) {
        string tmp;
        getline(cin, tmp);
        grid[i] = split(tmp, ",");
    }

    vector<int> ans(n);
    for (int i = 0; i < n; i++) {
        ans[i] = i;
    }

    // 对称的，只需要遍历一半除对角线
    for (int i = 0; i < n; i++) {
        for (int j =0; j < i; j++) {
            // 合并两个组
            if (grid[i][j] == 1) {
                merge(i, j, ans);
            }
        }
    }

    int res = 0;
    set<int> s;
    // 使用集合存储所有组长
    for (int i = 0; i < confirmId.size(); i++) {
        s.insert(find(confirmId[i], ans));
    }

    for (int i = 0; i < n; i++) {
        // 处于密切接触组
        if (s.find(find(i, ans)) != s.end()) {
            res++;
        }
    }
```

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

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 查找代表人
    public static int find(int a, int[] ans) {
        if (ans[a] != a) {
            ans[a] = find(ans[a], ans);
        }
        return ans[a];
    }

    // 合并两个组
    public static void merge(int a, int b, int[] ans) {
        int rootA = find(a, ans);
        int rootB = find(b, ans);
        int newRoot = Math.min(rootA, rootB);
        ans[rootA] = newRoot;
        ans[rootB] = newRoot;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine());
        String confirm = sc.nextLine().trim();
        // 没有病例的情况
        if (confirm.isEmpty()) {
            System.out.println(0);
            return;
        }

        String[] confirmParts = confirm.split(",");
        List<Integer> confirmIds = new ArrayList<>();
        for (String s : confirmParts) {
            confirmIds.add(Integer.parseInt(s));
        }

        int[][] grid = new int[n][n];
        for (int i = 0; i < n; i++) {
            String[] parts = sc.nextLine().split(",");
            for (int j = 0; j < n; j++) {
                grid[i][j] = Integer.parseInt(parts[j]);
            }
        }

        int[] ans = new int[n];
```

```
        // 初始化
        for (int i = 0; i < n; i++) ans[i] = i;

        // 合并组
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (grid[i][j] == 1) {
                    merge(i, j, ans);
                }
            }
        }

        Set<Integer> infectedGroups = new HashSet<>();
        // 使用集合存储病例所在组
        for (int id : confirmIds) {
            infectedGroups.add(find(id, ans));
        }

        int res = 0;
        for (int i = 0; i < n; i++) {
            if (infectedGroups.contains(find(i, ans))) {
                res++;
            }
        }

        // 减去已知病例人数
        System.out.println(res - confirmIds.size());
    }
}
```

Python

Plain Text

```
# 并查集查找代表人
def find(a, parent):
    if parent[a] != a:
        parent[a] = find(parent[a], parent)
    return parent[a]

# 合并两个组
def merge(a, b, parent):
    rootA = find(a, parent)
    rootB = find(b, parent)
    new_root = min(rootA, rootB)
    parent[rootA] = new_root
    parent[rootB] = new_root

if __name__ == "__main__":
    n = int(input())
    confirm = input().strip()
    # 不存在病例的情况
    if not confirm:
        print(0)
        exit()

    confirm_ids = list(map(int, confirm.split(",")))
    grid = []
    for _ in range(n):
        row = list(map(int, input().strip().split(",")))
        grid.append(row)

    parent = list(range(n))

    # 只遍历一半（对称矩阵）
    for i in range(n):
        for j in range(i):
            # 合并组
            if grid[i][j] == 1:
                merge(i, j, parent)
    # 存储病例所在组
    infected_groups = set(find(id, parent) for id in confirm_ids)

    res = 0
    for i in range(n):
        if find(i, parent) in infected_groups:
            res += 1

    # 减去已知病例人数
```

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

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on('line', (line) => {
    inputLines.push(line.trim());
}).on('close', () => {
    let n = parseInt(inputLines[0]);  // 读取矩阵大小

    // 若无确诊病例
    if (inputLines.length < 2 || inputLines[1] === '') {
        console.log(0);
        return;
    }

    // 读取确诊病例编号
    let confirmId = inputLines[1].split(',').map(Number);

    // 构造邻接矩阵
    let grid = [];
    for (let i = 0; i < n; i++) {
        grid.push(inputLines[2 + i].split(',').map(Number));
    }

    // 并查集初始化，每人是自己的代表人
    let parent = Array.from({ length: n }, (_, i) => i);

    // 查找组的代表人
    function find(a) {
        if (parent[a] !== a) {
            parent[a] = find(parent[a]);
        }
        return parent[a];
    }

    // 合并两个组
    function merge(a, b) {
        let rootA = find(a);
        let rootB = find(b);
        let newRoot = Math.min(rootA, rootB);
        parent[rootA] = newRoot;
```

```
            parent[rootB] = newRoot;
        }

        // 遍历下三角区域（不含对角线）合并接触者
        for (let i = 0; i < n; i++) {
            for (let j = 0; j < i; j++) {
                if (grid[i][j] === 1) {
                    merge(i, j);
                }
            }
        }

        // 收集确诊病例所在的组
        let infectedGroups = new Set();
        for (let id of confirmId) {
            infectedGroups.add(find(id));
        }

        // 统计密接人数（不含确诊者）
        let result = 0;
        for (let i = 0; i < n; i++) {
            if (infectedGroups.has(find(i)) && !confirmId.includes(i)) {
                result++;
            }
        }

        console.log(result);
    });
```

Go

Plain Text

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

// 查找代表人
func find(a int, parent []int) int {
    if parent[a] != a {
        parent[a] = find(parent[a], parent)
    }
    return parent[a]
}

// 合并两个组
func merge(a, b int, parent []int) {
    rootA := find(a, parent)
    rootB := find(b, parent)
    newRoot := min(rootA, rootB)
    parent[rootA] = newRoot
    parent[rootB] = newRoot
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}

func main() {
    reader := bufio.NewReader(os.Stdin)

    line, _ := reader.ReadString('\n')
    n, _ := strconv.Atoi(strings.TrimSpace(line))

    confirmLine, _ := reader.ReadString('\n')
    confirmLine = strings.TrimSpace(confirmLine)
    // 没有病例
    if confirmLine == "" {
        fmt.Println(0)
```

```
        return
    }

    confirmStrs := strings.Split(confirmLine, ",")
    confirmIds := make([]int, len(confirmStrs))
    for i, s := range confirmStrs {
        confirmIds[i], _ = strconv.Atoi(s)
    }

    grid := make([][]int, n)
    for i := 0; i < n; i++ {
        line, _ := reader.ReadString('\n')
        parts := strings.Split(strings.TrimSpace(line), ",")
        grid[i] = make([]int, n)
        for j := range parts {
            grid[i][j], _ = strconv.Atoi(parts[j])
        }
    }

    parent := make([]int, n)
    // 初始化
    for i := range parent {
        parent[i] = i
    }
    // 对称矩阵 遍历一半
    for i := 0; i < n; i++ {
        for j := 0; j < i; j++ {
            // 合并组
            if grid[i][j] == 1 {
                merge(i, j, parent)
            }
        }
    }
    // 病例所在组
    infected := make(map[int]bool)
    for _, id := range confirmIds {
        infected[find(id, parent)] = true
    }

    res := 0
    // 统计和病例处于同一组的人数(包含病例本身)
    for i := 0; i < n; i++ {
        if infected[find(i, parent)] {
            res++
        }
    }
    // 减去病例数量
    fmt.Println(res - len(confirmIds))
```

```
94    }
```

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（X1,X2,X3,...,Xn），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

Plain Text

```
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
```

输出

```
Plain Text
```

```
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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<list>
#include<queue>
#include<map>
#include<set>
using namespace std;

// 通用 split 函数
vector<int> split(const string& str, const string& delimiter) {
    vector<int> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(stoi(str.substr(start, end - start)));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(stoi(str.substr(start)));
    return result;
}

// 找到组中的代表人
int find(int a, vector<int> &ans) {
    if (ans[a] != a) {
        ans[a] = find(ans[a], ans);
    }
    return ans[a];
}

// 合并两个组
void merge(int a, int b, vector<int> &ans) {
    int rootA = find(a, ans);
    int rootB = find(b, ans);
    int newRoot = min(rootA, rootB);
    ans[rootA] = newRoot;
    ans[rootB] = newRoot;
}


```

```
int main() {
    int n;
    cin >> n;

    string confirm;
    // 忽略换行符
    cin.ignore();
    getline(cin, confirm);
    if (confirm.empty()) {
        cout << 0;
        return 0;
    }
    vector<int> confirmId = split(confirm, ",");
    vector<vector<int>> grid(n);
    for (int i = 0; i < n; i++) {
        string tmp;
        getline(cin, tmp);
        grid[i] = split(tmp, ",");
    }

    vector<int> ans(n);
    for (int i = 0; i < n; i++) {
        ans[i] = i;
    }

    // 对称的，只需要遍历一半除对角线
    for (int i = 0; i < n; i++) {
        for (int j =0; j < i; j++) {
            // 合并两个组
            if (grid[i][j] == 1) {
                merge(i, j, ans);
            }
        }
    }

    int res = 0;
    set<int> s;
    // 使用集合存储所有组长
    for (int i = 0; i < confirmId.size(); i++) {
        s.insert(find(confirmId[i], ans));
    }

    for (int i = 0; i < n; i++) {
        // 处于密切接触组
        if (s.find(find(i, ans)) != s.end()) {
            res++;
        }
    }
```

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

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 查找代表人
    public static int find(int a, int[] ans) {
        if (ans[a] != a) {
            ans[a] = find(ans[a], ans);
        }
        return ans[a];
    }

    // 合并两个组
    public static void merge(int a, int b, int[] ans) {
        int rootA = find(a, ans);
        int rootB = find(b, ans);
        int newRoot = Math.min(rootA, rootB);
        ans[rootA] = newRoot;
        ans[rootB] = newRoot;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = Integer.parseInt(sc.nextLine());
        String confirm = sc.nextLine().trim();
        // 没有病例的情况
        if (confirm.isEmpty()) {
            System.out.println(0);
            return;
        }

        String[] confirmParts = confirm.split(",");
        List<Integer> confirmIds = new ArrayList<>();
        for (String s : confirmParts) {
            confirmIds.add(Integer.parseInt(s));
        }

        int[][] grid = new int[n][n];
        for (int i = 0; i < n; i++) {
            String[] parts = sc.nextLine().split(",");
            for (int j = 0; j < n; j++) {
                grid[i][j] = Integer.parseInt(parts[j]);
            }
        }

        int[] ans = new int[n];
```

```
        // 初始化
        for (int i = 0; i < n; i++) ans[i] = i;

        // 合并组
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < i; j++) {
                if (grid[i][j] == 1) {
                    merge(i, j, ans);
                }
            }
        }

        Set<Integer> infectedGroups = new HashSet<>();
        // 使用集合存储病例所在组
        for (int id : confirmIds) {
            infectedGroups.add(find(id, ans));
        }

        int res = 0;
        for (int i = 0; i < n; i++) {
            if (infectedGroups.contains(find(i, ans))) {
                res++;
            }
        }

        // 减去已知病例人数
        System.out.println(res - confirmIds.size());
    }
}
```

Python

Plain Text

```
# 并查集查找代表人
def find(a, parent):
    if parent[a] != a:
        parent[a] = find(parent[a], parent)
    return parent[a]

# 合并两个组
def merge(a, b, parent):
    rootA = find(a, parent)
    rootB = find(b, parent)
    new_root = min(rootA, rootB)
    parent[rootA] = new_root
    parent[rootB] = new_root

if __name__ == "__main__":
    n = int(input())
    confirm = input().strip()
    # 不存在病例的情况
    if not confirm:
        print(0)
        exit()

    confirm_ids = list(map(int, confirm.split(",")))
    grid = []
    for _ in range(n):
        row = list(map(int, input().strip().split(",")))
        grid.append(row)

    parent = list(range(n))

    # 只遍历一半（对称矩阵）
    for i in range(n):
        for j in range(i):
            # 合并组
            if grid[i][j] == 1:
                merge(i, j, parent)
    # 存储病例所在组
    infected_groups = set(find(id, parent) for id in confirm_ids)

    res = 0
    for i in range(n):
        if find(i, parent) in infected_groups:
            res += 1

    # 减去已知病例人数
```

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

JavaScript

Plain Text

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputLines = [];
rl.on('line', (line) => {
    inputLines.push(line.trim());
}).on('close', () => {
    let n = parseInt(inputLines[0]);  // 读取矩阵大小

    // 若无确诊病例
    if (inputLines.length < 2 || inputLines[1] === '') {
        console.log(0);
        return;
    }

    // 读取确诊病例编号
    let confirmId = inputLines[1].split(',').map(Number);

    // 构造邻接矩阵
    let grid = [];
    for (let i = 0; i < n; i++) {
        grid.push(inputLines[2 + i].split(',').map(Number));
    }

    // 并查集初始化，每人是自己的代表人
    let parent = Array.from({ length: n }, (_, i) => i);

    // 查找组的代表人
    function find(a) {
        if (parent[a] !== a) {
            parent[a] = find(parent[a]);
        }
        return parent[a];
    }

    // 合并两个组
    function merge(a, b) {
        let rootA = find(a);
        let rootB = find(b);
        let newRoot = Math.min(rootA, rootB);
        parent[rootA] = newRoot;
```

```
            parent[rootB] = newRoot;
        }

        // 遍历下三角区域（不含对角线）合并接触者
        for (let i = 0; i < n; i++) {
            for (let j = 0; j < i; j++) {
                if (grid[i][j] === 1) {
                    merge(i, j);
                }
            }
        }

        // 收集确诊病例所在的组
        let infectedGroups = new Set();
        for (let id of confirmId) {
            infectedGroups.add(find(id));
        }

        // 统计密接人数（不含确诊者）
        let result = 0;
        for (let i = 0; i < n; i++) {
            if (infectedGroups.has(find(i)) && !confirmId.includes(i)) {
                result++;
            }
        }

        console.log(result);
    });
```

Go

Plain Text

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "strconv"
    "strings"
)

// 查找代表人
func find(a int, parent []int) int {
    if parent[a] != a {
        parent[a] = find(parent[a], parent)
    }
    return parent[a]
}

// 合并两个组
func merge(a, b int, parent []int) {
    rootA := find(a, parent)
    rootB := find(b, parent)
    newRoot := min(rootA, rootB)
    parent[rootA] = newRoot
    parent[rootB] = newRoot
}

func min(a, b int) int {
    if a < b {
        return a
    }
    return b
}

func main() {
    reader := bufio.NewReader(os.Stdin)

    line, _ := reader.ReadString('\n')
    n, _ := strconv.Atoi(strings.TrimSpace(line))

    confirmLine, _ := reader.ReadString('\n')
    confirmLine = strings.TrimSpace(confirmLine)
    // 没有病例
    if confirmLine == "" {
        fmt.Println(0)
```

```
        return
    }

    confirmStrs := strings.Split(confirmLine, ",")
    confirmIds := make([]int, len(confirmStrs))
    for i, s := range confirmStrs {
        confirmIds[i], _ = strconv.Atoi(s)
    }

    grid := make([][]int, n)
    for i := 0; i < n; i++ {
        line, _ := reader.ReadString('\n')
        parts := strings.Split(strings.TrimSpace(line), ",")
        grid[i] = make([]int, n)
        for j := range parts {
            grid[i][j], _ = strconv.Atoi(parts[j])
        }
    }

    parent := make([]int, n)
    // 初始化
    for i := range parent {
        parent[i] = i
    }
    // 对称矩阵 遍历一半
    for i := 0; i < n; i++ {
        for j := 0; j < i; j++ {
            // 合并组
            if grid[i][j] == 1 {
                merge(i, j, parent)
            }
        }
    }
    // 病例所在组
    infected := make(map[int]bool)
    for _, id := range confirmIds {
        infected[find(id, parent)] = true
    }

    res := 0
    // 统计和病例处于同一组的人数(包含病例本身)
    for i := 0; i < n; i++ {
        if infected[find(i, parent)] {
            res++
        }
    }
    // 减去病例数量
    fmt.Println(res - len(confirmIds))
```

```
94    }
```

> 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

```
1234567,1234567*
123456789012345
```

输出

```
1234567*
```

说明

> *可以匹配0或多个任意字符，故输出1234567 *

用例2

输入

```
123??????????345,123????*????345
123456789012345
```

输出

```
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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<cmath>
#include<map>
using namespace std;

// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

// 匹配函数
bool judge(string& pattern, string& str) {
    // 直接匹配成功
    if (pattern == "*") {
        return true;
    }
    int i = 0;
    int n = str.size();
    int m = pattern.size();
    for (; i < str.size() && i < pattern.size(); i++) {
        if (pattern[i] == str[i]) {
            continue;
        } else if (pattern[i] == '?') {
            // 偶数位置不能匹配 又不相同
            if (i % 2 == 0) {
                return false;
            }
        } else if (pattern[i] == '*') {
            // 如果是pattern最后一个肯定可以匹配成功
            if (i == m - 1) {
```

```
                return true;
            }
            string suffix = pattern.substr(i+1);
            int pos = suffix.size() - 1;
            // 后缀进行匹配
            for (int j = n - 1; j >= i && pos >= 0; j--,pos--) {
                if (suffix[pos] == str[j]) {
                    continue;
                } else if (suffix[pos] == '?') {
                    if (j % 2 == 0) {
                        break;
                    }
                }
            }
            // 判断后缀是否匹配成功
            return pos < 0;
        } else {
            return false;
        }
    }
    // 从这里返回说明不存在* 所以必需两者长度相同
    return n == m;
}

int main() {
    string input;
    getline(cin, input);
    vector<string> settings = split(input, ",");
    string number;
    getline(cin, number);

    vector<string> res;
    int n = settings.size();
    for (int i = 0; i < n; i++) {
        string pattern = settings[i];
        bool flag = judge(pattern, number);
        if (flag) {
            res.push_back(pattern);
        }
    }
    if (res.empty()) {
        cout << "null";
        return 0;
    }
    // 字典序升序
    sort(res.begin(), res.end());
    int m = res.size();
    for (int i = 0; i < m; i++) {
```

```
            cout << res[i];
            if (i != m - 1) {
                cout << ",";
            }
        }
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 匹配函数
    public static boolean judge(String pattern, String str) {
        if (pattern.equals("*")) return true;
        int n = str.length(), m = pattern.length();
        int i = 0;
        for (; i < n && i < m; i++) {
            char pc = pattern.charAt(i), sc = str.charAt(i);
            if (pc == sc) continue;
            else if (pc == '?') {
                // 偶数位置不能匹配又不相同
                if (i % 2 == 0) return false;
            } else if (pc == '*') {
                // pattern的最后一位*肯定可以匹配完成
                if (i == m - 1) return true;
                // pattern的后缀必须完整匹配
                String suffix = pattern.substring(i + 1);
                int pos = suffix.length() - 1;
                for (int j = n - 1; j >= i && pos >= 0; j--, pos--) {
                    char sp = suffix.charAt(pos), sj = str.charAt(j);
                    if (sp == sj) continue;
                    else if (sp == '?') {
                        if (j % 2 == 0) break;
                    } else break;
                }
                return pos < 0;
            } else return false;
        }
        // 从这里返回说明不存在 *，必须长度相同
        return n == m;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] settings = sc.nextLine().split(",");
        String number = sc.nextLine();
        List<String> res = new ArrayList<>();
        for (String pattern : settings) {
            if (judge(pattern, number)) res.add(pattern);
        }
        if (res.isEmpty()) {
            System.out.println("null");
        } else {
```

```
            Collections.sort(res);
            System.out.println(String.join(",", res));
        }
    }
}
```

Python

Plain Text

```
# 匹配函数
def judge(pattern, s):
    if pattern == '*':
        return True
    n, m = len(s), len(pattern)
    i = 0
    while i < n and i < m:
        if pattern[i] == s[i]:
            pass
        elif pattern[i] == '?':
            # 偶数位置不能匹配又不相同
            if i % 2 == 0:
                return False
        elif pattern[i] == '*':
            # pattern末尾为*
            if i == m - 1:
                return True
            # 后缀匹配
            suffix = pattern[i + 1:]
            pos = len(suffix) - 1
            j = n - 1
            while j >= i and pos >= 0:
                if suffix[pos] == s[j]:
                    pass
                elif suffix[pos] == '?':
                    if j % 2 == 0:
                        break
                else:
                    break
                j -= 1
                pos -= 1
            return pos < 0
        else:
            return False
        i += 1
    # 从这里返回说明不存在 *，必须长度相同
    return m == n

settings = input().split(',')
number = input()
res = []

for pattern in settings:
    if judge(pattern, number):
        res.append(pattern)
```

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

JavaScript

Plain Text

```
const rl = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});

let lines = [];
rl.on('line', line => {
    lines.push(line);
    if (lines.length === 2) {
        const settings = lines[0].split(',');
        const number = lines[1];
        const res = [];

        // 匹配函数
        function judge(pattern, str) {
            if (pattern === '*') return true;
            let n = str.length, m = pattern.length;
            let i = 0;
            for (; i < n && i < m; i++) {
                if (pattern[i] === str[i]) continue;
                else if (pattern[i] === '?') {
                    // 偶数位置不能匹配又不相同
                    if (i % 2 === 0) return false;
                } else if (pattern[i] === '*') {
                    // 达到pattern的末尾并且为*肯定可以匹配
                    if (i === m - 1) return true;
                    const suffix = pattern.slice(i + 1);
                    // 后缀匹配
                    let pos = suffix.length - 1;
                    for (let j = n - 1; j >= i && pos >= 0; j--, pos--) {
                        if (suffix[pos] === str[j]) continue;
                        else if (suffix[pos] === '?') {
                            if (j % 2 === 0) break;
                        } else break;
                    }
                    return pos < 0;
                } else return false;
            }
            return m === n;
        }

        for (const pattern of settings) {
            if (judge(pattern, number)) {
                res.push(pattern);
            }
```

```
        }

        if (res.length === 0) console.log("null");
        else console.log(res.sort().join(','));

        rl.close();
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strings"
)

// 匹配函数
func judge(pattern, str string) bool {

  if pattern == "*" {
    return true
  }
  n, m := len(str), len(pattern)
  i := 0
  for i < n && i < m {
    pc, sc := pattern[i], str[i]
    if pc == sc {
      continue
    } else if pc == '?' {
      // 偶数位置不能匹配又不相同
      if i%2 == 0 {
        return false
      }
    } else if pc == '*' {
      // 到pattern的末尾并且为*肯定可以匹配
      if i == m-1 {
        return true
      }
      // 后缀匹配
      suffix := pattern[i+1:]
      pos := len(suffix) - 1
      for j := n - 1; j >= i && pos >= 0; j, pos = j-1, pos-1 {
        sp := suffix[pos]
        sj := str[j]
        if sp == sj {
          continue
        } else if sp == '?' {
          if j%2 == 0 {
            break
          }
        } else {
```

```
            break
          }
        }
        return pos < 0
      } else {
        return false
      }
      i++
    }
    return m == n
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    settings := strings.Split(scanner.Text(), ",")
    scanner.Scan()
    number := scanner.Text()

    var res []string
    for _, pattern := range settings {
      if judge(pattern, number) {
        res = append(res, pattern)
      }
    }

    if len(res) == 0 {
      fmt.Println("null")
    } else {
      sort.Strings(res)
      fmt.Println(strings.Join(res, ","))
    }
}
```

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

```
1234567,1234567*
123456789012345
```

输出

Plain Text

```
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

```
123??????????345,123????*????345
123456789012345
```

输出

```
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++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
#include<cmath>
#include<map>
using namespace std;

// 通用 split 函数
vector<string> split(const string& str, const string& delimiter) {
    vector<string> result;
    size_t start = 0;
    size_t end = str.find(delimiter);
    while (end != string::npos) {
        result.push_back(str.substr(start, end - start));
        start = end + delimiter.length();
        end = str.find(delimiter, start);
    }
    // 添加最后一个部分
    result.push_back(str.substr(start));
    return result;
}

// 匹配函数
bool judge(string& pattern, string& str) {
    // 直接匹配成功
    if (pattern == "*") {
        return true;
    }
    int i = 0;
    int n = str.size();
    int m = pattern.size();
    for (; i < str.size() && i < pattern.size(); i++) {
        if (pattern[i] == str[i]) {
            continue;
        } else if (pattern[i] == '?') {
            // 偶数位置不能匹配 又不相同
            if (i % 2 == 0) {
                return false;
            }
        } else if (pattern[i] == '*') {
            // 如果是pattern最后一个肯定可以匹配成功
            if (i == m - 1) {
```

```
                return true;
            }
            string suffix = pattern.substr(i+1);
            int pos = suffix.size() - 1;
            // 后缀进行匹配
            for (int j = n - 1; j >= i && pos >= 0; j--,pos--) {
                if (suffix[pos] == str[j]) {
                    continue;
                } else if (suffix[pos] == '?') {
                    if (j % 2 == 0) {
                        break;
                    }
                }
            }
            // 判断后缀是否匹配成功
            return pos < 0;
        } else {
            return false;
        }
    }
    // 从这里返回说明不存在* 所以必需两者长度相同
    return n == m;
}

int main() {
    string input;
    getline(cin, input);
    vector<string> settings = split(input, ",");
    string number;
    getline(cin, number);

    vector<string> res;
    int n = settings.size();
    for (int i = 0; i < n; i++) {
        string pattern = settings[i];
        bool flag = judge(pattern, number);
        if (flag) {
            res.push_back(pattern);
        }
    }
    if (res.empty()) {
        cout << "null";
        return 0;
    }
    // 字典序升序
    sort(res.begin(), res.end());
    int m = res.size();
    for (int i = 0; i < m; i++) {
```

```
            cout << res[i];
            if (i != m - 1) {
                cout << ",";
            }
        }
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 匹配函数
    public static boolean judge(String pattern, String str) {
        if (pattern.equals("*")) return true;
        int n = str.length(), m = pattern.length();
        int i = 0;
        for (; i < n && i < m; i++) {
            char pc = pattern.charAt(i), sc = str.charAt(i);
            if (pc == sc) continue;
            else if (pc == '?') {
                // 偶数位置不能匹配又不相同
                if (i % 2 == 0) return false;
            } else if (pc == '*') {
                // pattern的最后一位*肯定可以匹配完成
                if (i == m - 1) return true;
                // pattern的后缀必须完整匹配
                String suffix = pattern.substring(i + 1);
                int pos = suffix.length() - 1;
                for (int j = n - 1; j >= i && pos >= 0; j--, pos--) {
                    char sp = suffix.charAt(pos), sj = str.charAt(j);
                    if (sp == sj) continue;
                    else if (sp == '?') {
                        if (j % 2 == 0) break;
                    } else break;
                }
                return pos < 0;
            } else return false;
        }
        // 从这里返回说明不存在 *，必须长度相同
        return n == m;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] settings = sc.nextLine().split(",");
        String number = sc.nextLine();
        List<String> res = new ArrayList<>();
        for (String pattern : settings) {
            if (judge(pattern, number)) res.add(pattern);
        }
        if (res.isEmpty()) {
            System.out.println("null");
        } else {
```

```
            Collections.sort(res);
            System.out.println(String.join(",", res));
        }
    }
}
```

Python

Plain Text

```
# 匹配函数
def judge(pattern, s):
    if pattern == '*':
        return True
    n, m = len(s), len(pattern)
    i = 0
    while i < n and i < m:
        if pattern[i] == s[i]:
            pass
        elif pattern[i] == '?':
            # 偶数位置不能匹配又不相同
            if i % 2 == 0:
                return False
        elif pattern[i] == '*':
            # pattern末尾为*
            if i == m - 1:
                return True
            # 后缀匹配
            suffix = pattern[i + 1:]
            pos = len(suffix) - 1
            j = n - 1
            while j >= i and pos >= 0:
                if suffix[pos] == s[j]:
                    pass
                elif suffix[pos] == '?':
                    if j % 2 == 0:
                        break
                else:
                    break
                j -= 1
                pos -= 1
            return pos < 0
        else:
            return False
        i += 1
    # 从这里返回说明不存在 *，必须长度相同
    return m == n

settings = input().split(',')
number = input()
res = []

for pattern in settings:
    if judge(pattern, number):
        res.append(pattern)
```

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

JavaScript

Plain Text

```
const rl = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
});

let lines = [];
rl.on('line', line => {
    lines.push(line);
    if (lines.length === 2) {
        const settings = lines[0].split(',');
        const number = lines[1];
        const res = [];

        // 匹配函数
        function judge(pattern, str) {
            if (pattern === '*') return true;
            let n = str.length, m = pattern.length;
            let i = 0;
            for (; i < n && i < m; i++) {
                if (pattern[i] === str[i]) continue;
                else if (pattern[i] === '?') {
                    // 偶数位置不能匹配又不相同
                    if (i % 2 === 0) return false;
                } else if (pattern[i] === '*') {
                    // 达到pattern的末尾并且为*肯定可以匹配
                    if (i === m - 1) return true;
                    const suffix = pattern.slice(i + 1);
                    // 后缀匹配
                    let pos = suffix.length - 1;
                    for (let j = n - 1; j >= i && pos >= 0; j--, pos--) {
                        if (suffix[pos] === str[j]) continue;
                        else if (suffix[pos] === '?') {
                            if (j % 2 === 0) break;
                        } else break;
                    }
                    return pos < 0;
                } else return false;
            }
            return m === n;
        }

        for (const pattern of settings) {
            if (judge(pattern, number)) {
                res.push(pattern);
            }
```

```
        }

        if (res.length === 0) console.log("null");
        else console.log(res.sort().join(','));

        rl.close();
    }
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "sort"
  "strings"
)

// 匹配函数
func judge(pattern, str string) bool {

  if pattern == "*" {
    return true
  }
  n, m := len(str), len(pattern)
  i := 0
  for i < n && i < m {
    pc, sc := pattern[i], str[i]
    if pc == sc {
      continue
    } else if pc == '?' {
      // 偶数位置不能匹配又不相同
      if i%2 == 0 {
        return false
      }
    } else if pc == '*' {
      // 到pattern的末尾并且为*肯定可以匹配
      if i == m-1 {
        return true
      }
      // 后缀匹配
      suffix := pattern[i+1:]
      pos := len(suffix) - 1
      for j := n - 1; j >= i && pos >= 0; j, pos = j-1, pos-1 {
        sp := suffix[pos]
        sj := str[j]
        if sp == sj {
          continue
        } else if sp == '?' {
          if j%2 == 0 {
            break
          }
        } else {
```

```
            break
          }
        }
        return pos < 0
      } else {
        return false
      }
      i++
    }
    return m == n
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    settings := strings.Split(scanner.Text(), ",")
    scanner.Scan()
    number := scanner.Text()

    var res []string
    for _, pattern := range settings {
      if judge(pattern, number) {
        res = append(res, pattern)
      }
    }

    if len(res) == 0 {
      fmt.Println("null")
    } else {
      sort.Strings(res)
      fmt.Println(strings.Join(res, ","))
    }
}
```

> 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