t0727

华为OD机考2025C卷 - 全排列 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 找朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 小明的幸运数 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 报数问题 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025B卷 - 最少面试官 / 招聘 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025B卷 - 上班之路 (C++ & Python & JAVA & JS & GO)_华为od2025b卷-CSDN博客

华为OD机考2025C卷 - 中文分词模拟器 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 连续出牌数量 (C++ & Python & JAVA & JS & GO)-CSDN博客

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

华为OD机考2025C卷 - 全排列 (C++ & Python & JAVA & JS & GO)-CSDN博客

全排列

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

> 华为OD机试2025C卷 100分题型

题目描述

给定一个只包含大写英文字母的字符串S，要求你给出对S重新排列的所有不相同的排列数。
如：S为ABA，则不同的排列有ABA、AAB、BAA三种。

输入描述

输入一个长度不超过10的字符串S，我们确保都是大写的

输出描述

输出S重新排列的**所有不相同的排列数**（包含自己本身）

用例1

输入

```
ABA
```

输出

```
3
```

用例2

输入

```
ABCDEFGHHA
```

输出

```
907200
```

题解

思路: 数学问题,排列组合问题。
有n个元素，不考虑重复的情况有 `n!` 种排列情况。接下来再考虑重复情况，假设一个字母出现了 `m` 次，在其它字符规定情况下，这个字母在整体排列中会占据m个位置，并且相互交换是同种情况，它能形成 `m!` 种不同方案。

> 所以可以得出一个规律，假设某字符出现x次，另一个字符出现y次，则最终不重复方案数有 `n! / x! / y!`

C++

Plain Text

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

int main() {
    string input;
    getline(cin, input);
    int n = input.size();
    // 统计各个字母出现的次数
    map<char, int> charCount;

    // 预计算1-10的阶乘值
    int tmp = 1;
    vector<int> factorial(11, 0);
    for (int i = 1; i <= 10; i++) {
        tmp *= i;
        factorial[i] = tmp;
    }

    // 统计每个字符出现的次数
    for (char c : input) {
        charCount[c]++;
    }

    // 不考虑重复的情况
    int total = factorial[n];

    // 去除重复的
    for (auto &p : charCount) {
        total /= factorial[p.second];
    }
    cout << total;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        int n = input.length();

        // 统计各个字母出现的次数
        Map<Character, Integer> charCount = new HashMap<>();

        // 预计算1-10的阶乘值
        int[] factorial = new int[11];
        int tmp = 1;
        for (int i = 1; i <= 10; i++) {
            tmp *= i;
            factorial[i] = tmp;
        }

        // 统计每个字符出现的次数
        for (char c : input.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        // 不考虑重复的情况
        int total = factorial[n];

        // 去除重复的
        for (int count : charCount.values()) {
            total /= factorial[count];
        }

        System.out.println(total);
    }
}
```

Python

Plain Text

```
import sys
from collections import Counter

# 读取输入字符串
input_str = sys.stdin.readline().strip()
n = len(input_str)

# 统计每个字符出现的次数
char_count = Counter(input_str)

# 预计算 1 到 10 的阶乘
factorial = [1] * 11
for i in range(1, 11):
    factorial[i] = factorial[i - 1] * i

# 总的排列数（不考虑重复）
total = factorial[n]

# 去除重复字符的排列
for cnt in char_count.values():
    total //= factorial[cnt]

print(total)
```

JavaScript

Plain Text

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

// 创建读取接口
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputStr = '';

// 读取整行输入
rl.on('line', (line) => {
    inputStr = line.trim();
    rl.close(); // 只处理一行
});

rl.on('close', () => {
    const n = inputStr.length;

    // 统计每个字符出现的次数
    const charCount = new Map();
    for (const c of inputStr) {
        charCount.set(c, (charCount.get(c) || 0) + 1);
    }

    // 预计算1到10的阶乘
    const factorial = new Array(11).fill(1);
    for (let i = 1; i <= 10; i++) {
        factorial[i] = factorial[i - 1] * i;
    }

    // 不考虑重复字符时的全排列数量
    let total = factorial[n];

    // 去除重复字符带来的重复排列
    for (const cnt of charCount.values()) {
        total /= factorial[cnt];
    }

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

Go

```
Plain Text
```

```
package main

import (
  "bufio"
  "fmt"
  "os"
)

func main() {
  // 读取输入
  reader := bufio.NewReader(os.Stdin)
  input, _ := reader.ReadString('\n')
  input = input[:len(input)-1] // 去除换行符

  n := len(input)

  // 预计算 1 到 10 的阶乘
  factorial := make([]int, 11)
  factorial[0] = 1
  for i := 1; i <= 10; i++ {
    factorial[i] = factorial[i-1] * i
  }

  // 统计字符出现次数
  charCount := make(map[rune]int)
  for _, c := range input {
    charCount[c]++
  }

  // 总的排列数（不考虑重复）
  total := factorial[n]

  // 去除重复方案
  for _, cnt := range charCount {
    total /= factorial[cnt]
  }

  fmt.Println(total)
}
```

全排列

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

> 华为OD机试2025C卷 100分题型

题目描述

给定一个只包含大写英文字母的字符串S，要求你给出对S重新排列的所有不相同的排列数。

如：S为ABA，则不同的排列有ABA、AAB、BAA三种。

输入描述

输入一个长度不超过10的字符串S，我们确保都是大写的

输出描述

输出S重新排列的**所有不相同的排列数**（包含自己本身）

用例1

输入

```Plain Text
ABA
```

输出

```Plain Text
3
```

用例2

输入

```Plain Text
ABCDEFGHHA
```

输出

```Plain Text
907200
```

题解

思路: 数学问题,排列组合问题。
有n个元素，不考虑重复的情况有 `n!` 种排列情况。接下来再考虑重复情况，假设一个字母出现了 `m` 次，在其它字符规定情况下，这个字母在整体排列中会占据m个位置，并且相互交换是同种情况，它能形成 `m!` 种不同方案。

> 所以可以得出一个规律，假设某字符出现x次，另一个字符出现y次，则最终不重复方案数有 `n! / x! / y!`

C++

Plain Text

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

int main() {
    string input;
    getline(cin, input);
    int n = input.size();
    // 统计各个字母出现的次数
    map<char, int> charCount;

    // 预计算1-10的阶乘值
    int tmp = 1;
    vector<int> factorial(11, 0);
    for (int i = 1; i <= 10; i++) {
        tmp *= i;
        factorial[i] = tmp;
    }

    // 统计每个字符出现的次数
    for (char c : input) {
        charCount[c]++;
    }

    // 不考虑重复的情况
    int total = factorial[n];

    // 去除重复的
    for (auto &p : charCount) {
        total /= factorial[p.second];
    }
    cout << total;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        int n = input.length();

        // 统计各个字母出现的次数
        Map<Character, Integer> charCount = new HashMap<>();

        // 预计算1-10的阶乘值
        int[] factorial = new int[11];
        int tmp = 1;
        for (int i = 1; i <= 10; i++) {
            tmp *= i;
            factorial[i] = tmp;
        }

        // 统计每个字符出现的次数
        for (char c : input.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        // 不考虑重复的情况
        int total = factorial[n];

        // 去除重复的
        for (int count : charCount.values()) {
            total /= factorial[count];
        }

        System.out.println(total);
    }
}
```

Python

Plain Text

```
import sys
from collections import Counter

# 读取输入字符串
input_str = sys.stdin.readline().strip()
n = len(input_str)

# 统计每个字符出现的次数
char_count = Counter(input_str)

# 预计算 1 到 10 的阶乘
factorial = [1] * 11
for i in range(1, 11):
    factorial[i] = factorial[i - 1] * i

# 总的排列数（不考虑重复）
total = factorial[n]

# 去除重复字符的排列
for cnt in char_count.values():
    total //= factorial[cnt]

print(total)
```

JavaScript

Plain Text

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

// 创建读取接口
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputStr = '';

// 读取整行输入
rl.on('line', (line) => {
    inputStr = line.trim();
    rl.close(); // 只处理一行
});

rl.on('close', () => {
    const n = inputStr.length;

    // 统计每个字符出现的次数
    const charCount = new Map();
    for (const c of inputStr) {
        charCount.set(c, (charCount.get(c) || 0) + 1);
    }

    // 预计算1到10的阶乘
    const factorial = new Array(11).fill(1);
    for (let i = 1; i <= 10; i++) {
        factorial[i] = factorial[i - 1] * i;
    }

    // 不考虑重复字符时的全排列数量
    let total = factorial[n];

    // 去除重复字符带来的重复排列
    for (const cnt of charCount.values()) {
        total /= factorial[cnt];
    }

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

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
)

func main() {
  // 读取输入
  reader := bufio.NewReader(os.Stdin)
  input, _ := reader.ReadString('\n')
  input = input[:len(input)-1] // 去除换行符

  n := len(input)

  // 预计算 1 到 10 的阶乘
  factorial := make([]int, 11)
  factorial[0] = 1
  for i := 1; i <= 10; i++ {
    factorial[i] = factorial[i-1] * i
  }

  // 统计字符出现次数
  charCount := make(map[rune]int)
  for _, c := range input {
    charCount[c]++
  }

  // 总的排列数（不考虑重复）
  total := factorial[n]

  // 去除重复方案
  for _, cnt := range charCount {
    total /= factorial[cnt]
  }

  fmt.Println(total)
}
```

> 来自: 华为OD机考2025C卷 - 全排列 (C++ & Python & JAVA & JS & GO)-CSDN博客

全排列

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

> 华为OD机试2025C卷 100分题型

题目描述

给定一个只包含大写英文字母的字符串S，要求你给出对S重新排列的所有不相同的排列数。
如：S为ABA，则不同的排列有ABA、AAB、BAA三种。

输入描述

输入一个长度不超过10的字符串S，我们确保都是大写的

输出描述

输出S重新排列的**所有不相同的排列数**（包含自己本身）

用例1

输入

```Plain Text
ABA
```

输出

```Plain Text
3
```

用例2

输入

```Plain Text
ABCDEFGHHA
```

输出

```
907200
```

题解

思路: 数学问题,排列组合问题。

有n个元素，不考虑重复的情况有 `n!` 种排列情况。接下来再考虑重复情况，假设一个字母出现了 `m` 次，在其它字符规定情况下，这个字母在整体排列中会占据m个位置，并且相互交换是同种情况，它能形成 `m!` 种不同方案。

> 所以可以得出一个规律，假设某字符出现x次，另一个字符出现y次，则最终不重复方案数有 `n! / x! / y!`

C++

Plain Text

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

int main() {
    string input;
    getline(cin, input);
    int n = input.size();
    // 统计各个字母出现的次数
    map<char, int> charCount;

    // 预计算1-10的阶乘值
    int tmp = 1;
    vector<int> factorial(11, 0);
    for (int i = 1; i <= 10; i++) {
        tmp *= i;
        factorial[i] = tmp;
    }

    // 统计每个字符出现的次数
    for (char c : input) {
        charCount[c]++;
    }

    // 不考虑重复的情况
    int total = factorial[n];

    // 去除重复的
    for (auto &p : charCount) {
        total /= factorial[p.second];
    }
    cout << total;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        int n = input.length();

        // 统计各个字母出现的次数
        Map<Character, Integer> charCount = new HashMap<>();

        // 预计算1-10的阶乘值
        int[] factorial = new int[11];
        int tmp = 1;
        for (int i = 1; i <= 10; i++) {
            tmp *= i;
            factorial[i] = tmp;
        }

        // 统计每个字符出现的次数
        for (char c : input.toCharArray()) {
            charCount.put(c, charCount.getOrDefault(c, 0) + 1);
        }

        // 不考虑重复的情况
        int total = factorial[n];

        // 去除重复的
        for (int count : charCount.values()) {
            total /= factorial[count];
        }

        System.out.println(total);
    }
}
```

Python

Plain Text

```
import sys
from collections import Counter

# 读取输入字符串
input_str = sys.stdin.readline().strip()
n = len(input_str)

# 统计每个字符出现的次数
char_count = Counter(input_str)

# 预计算 1 到 10 的阶乘
factorial = [1] * 11
for i in range(1, 11):
    factorial[i] = factorial[i - 1] * i

# 总的排列数（不考虑重复）
total = factorial[n]

# 去除重复字符的排列
for cnt in char_count.values():
    total //= factorial[cnt]

print(total)
```

JavaScript

Plain Text

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

// 创建读取接口
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

let inputStr = '';

// 读取整行输入
rl.on('line', (line) => {
    inputStr = line.trim();
    rl.close(); // 只处理一行
});

rl.on('close', () => {
    const n = inputStr.length;

    // 统计每个字符出现的次数
    const charCount = new Map();
    for (const c of inputStr) {
        charCount.set(c, (charCount.get(c) || 0) + 1);
    }

    // 预计算1到10的阶乘
    const factorial = new Array(11).fill(1);
    for (let i = 1; i <= 10; i++) {
        factorial[i] = factorial[i - 1] * i;
    }

    // 不考虑重复字符时的全排列数量
    let total = factorial[n];

    // 去除重复字符带来的重复排列
    for (const cnt of charCount.values()) {
        total /= factorial[cnt];
    }

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

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
)

func main() {
  // 读取输入
  reader := bufio.NewReader(os.Stdin)
  input, _ := reader.ReadString('\n')
  input = input[:len(input)-1] // 去除换行符

  n := len(input)

  // 预计算 1 到 10 的阶乘
  factorial := make([]int, 11)
  factorial[0] = 1
  for i := 1; i <= 10; i++ {
    factorial[i] = factorial[i-1] * i
  }

  // 统计字符出现次数
  charCount := make(map[rune]int)
  for _, c := range input {
    charCount[c]++
  }

  // 总的排列数（不考虑重复）
  total := factorial[n]

  // 去除重复方案
  for _, cnt := range charCount {
    total /= factorial[cnt]
  }

  fmt.Println(total)
}
```

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

> 来自: 华为OD机考2025C卷 – 全排列 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 找朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

找朋友

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

> 华为OD机试2025C卷 100分题型

题目描述

在学校中，N个小朋友站成一队， 第i个小朋友的身高为height[i]，
第i个小朋友可以看到的第一个比自己身高更高的小朋友j，那么j是i的好朋友(要求j > i)。
请重新生成一个列表，对应位置的输出是每个小朋友的好朋友位置，如果没有看到好朋友，请在该位置用0代替。
小朋友人数范围是 [0, 40000]。

输入描述

第一行输入N，N表示有N个小朋友
第二行输入N个小朋友的身高height[i]，都是整数

输出描述

输出N个小朋友的好朋友的位置

用例1

输入

Plain Text

```
2
100 95
```

输出

Plain Text

```
0 0
```

说明

> 第一个小朋友身高100，站在队尾位置，向队首看，没有比他身高高的小朋友，所以输出第一个值为0。
> 第二个小朋友站在队首，前面也没有比他身高高的小朋友，所以输出第二个值为0。

用例2

输入

```Plain Text
8
123 124 125 121 119 122 126 123
```

输出

```Plain Text
1 2 6 5 5 6 0 0
```

说明

> 123的好朋友是1位置上的124
> 124的好朋友是2位置上的125
> 125的好朋友是6位置上的126
> 以此类推

题解

思路：`单调栈`。

1. 这种题目非常适合使用`单调栈`来解决。题目要求是为每个小朋友找到下一个比自己高的伙伴。可以定义一个单调栈`stk`在栈中维持一个非严格递减序列，栈中存储小朋友的索引。
2. 遍历逻辑如下，当前遍历位置为`i`,根据不同情况处理如下：
   - `stk.empty() || heights[stk.top()] >= heights[i]`： 此时符合非严格递减，直接将i压入栈。
   - `stk.empty() && heights[stk.top()] < heights[i]`：此时将i压入栈会破坏掉非严格递减序列，同时这种情况也代表当前`i`是栈顶元素位置小朋友的第一个比自己大小朋友位置。此时循环迭代弹出栈顶高度比自己矮的小朋友序号，并将对应位置小伙伴的朋友设置为`i`。 最后将`i`压入栈，继续保持非严格递减序列。
3. 最后输出每个小朋友的朋友序号即可。

C++

Plain Text

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

int main() {
    int n;
    cin >> n;
    vector<int> heights(n);
    vector<int> res(n, 0);
    for (int i = 0; i < n; i++) {
        cin >> heights[i];
    }
    // 维持一个递减的栈
    stack<int> stk;
    for (int i = 0; i < n; i++) {
        int height = heights[i];
        // 空,直接压栈
        if (stk.empty() || heights[stk.top()] >= height) {
            stk.push(i);
        } else {
            // 栈中高度小于当前值好朋友全部设置为i
            while (!stk.empty() && heights[stk.top()] < height) {
                int pos = stk.top();
                stk.pop();
                res[pos] = i;
            }
            stk.push(i);
        }
    }
    // 输出结果
    for (int i = 0; i < n; i++) {
        cout << res[i];
        if (i != n -1) {
            cout << " ";
        }
    }
    return 0;
```

```
}
```

JAVA

Plain Text

```java
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int[] heights = new int[n];
        int[] res = new int[n];

        for (int i = 0; i < n; i++) {
            heights[i] = scanner.nextInt();
        }

        // 维持一个递减的栈
        Stack<Integer> stk = new Stack<>();
        for (int i = 0; i < n; i++) {
            int height = heights[i];
            // 空，直接压栈
            if (stk.isEmpty() || heights[stk.peek()] >= height) {
                stk.push(i);
            } else {
                // 栈中高度小于当前值的好朋友全部设置为 i
                while (!stk.isEmpty() && heights[stk.peek()] < height) {
                    int pos = stk.pop();
                    res[pos] = i;
                }
                stk.push(i);
            }
        }

        // 输出结果
        for (int i = 0; i < n; i++) {
            System.out.print(res[i]);
            if (i != n - 1) {
                System.out.print(" ");
            }
        }
    }
}
```

Python

```
import sys

# 读取输入
n = int(sys.stdin.readline())
heights = list(map(int, sys.stdin.readline().split()))
res = [0] * n

stk = []
# 维持一个递减的栈
for i in range(n):
    height = heights[i]
    # 空，直接压栈
    if not stk or heights[stk[-1]] >= height:
        stk.append(i)
    else:
        # 栈中高度小于当前值的好朋友全部设置为 i
        while stk and heights[stk[-1]] < height:
            pos = stk.pop()
            res[pos] = i
        stk.append(i)

# 输出结果
print(" ".join(map(str, res)))
```

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());
    if (inputLines.length === 2) {
        const n = parseInt(inputLines[0]);
        const heights = inputLines[1].split(' ').map(Number);
        const res = new Array(n).fill(0);
        const stk = [];

        // 维持一个递减的栈
        for (let i = 0; i < n; i++) {
            const height = heights[i];
            // 空，直接压栈
            if (stk.length === 0 || heights[stk[stk.length - 1]] >= heigh
t) {
                stk.push(i);
            } else {
                // 栈中高度小于当前值的好朋友全部设置为 i
                while (stk.length > 0 && heights[stk[stk.length - 1]] < he
ight) {
                    const pos = stk.pop();
                    res[pos] = i;
                }
                stk.push(i);
            }
        }

        // 输出结果
        console.log(res.join(' '));
        rl.close();
    }
});
```

Go

Plain Text

```
package main

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

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Split(bufio.ScanWords)

  // 读取 n
  scanner.Scan()
  n, _ := strconv.Atoi(scanner.Text())

  heights := make([]int, n)
  res := make([]int, n)
  stack := make([]int, 0, n)

  // 读取 n 个高度
  for i := 0; i < n; i++ {
    scanner.Scan()
    heights[i], _ = strconv.Atoi(scanner.Text())
  }

  // 单调栈处理
  for i := 0; i < n; i++ {
    height := heights[i]
        // 栈中高度小于当前值的好朋友全部设置为 i
    for len(stack) > 0 && heights[stack[len(stack)-1]] < height {
      pos := stack[len(stack)-1]
      stack = stack[:len(stack)-1]
      res[pos] = i
    }
    stack = append(stack, i)
  }

  // 输出结果
    for i := 0; i < n; i++ {
        fmt.Print(res[i])
        if i != n-1 {
            fmt.Print(" ")
```

```
            }
        }
        fmt.Println()
    }
```

> 来自: 华为OD机试2025C卷 – 找朋友 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025C卷 - 小明的幸运数 (C++ & Python & JAVA & JS & GO)-CSDN博客

小明的幸运数

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

> 华为OD机试2025C卷 100分题型

题目描述

小明在玩一个游戏，游戏规则如下：
在游戏开始前，小明站在坐标轴原点处（坐标值为0）.
给定一组指令和一个幸运数，每个指令都是一个整数，小明按照指令前进指定步数或者后退指定步数。前进代表朝坐标轴的正方向走，后退代表朝坐标轴的负方向走。
幸运数为一个整数，如果某个指令正好和幸运数相等，则小明行进步数+1。
例如：
幸运数为3，指令为[2,3,0,-5]
指令为2，表示前进2步；
指令为3，正好和幸运数相等，前进3+1=4步；
指令为0，表示原地不动，既不前进，也不后退。
指令为-5，表示后退5步。
请你计算小明在整个游戏过程中，小明所处的最大坐标值。

输入描述

第一行输入1个数字，代表指令的总个数 n（1 ≤ n ≤ 100）
第二行输入1个数字，代表幸运数m（-100 ≤ m ≤ 100）
第三行输入n个指令，每个指令的取值范围为：-100 ≤ 指令值 ≤ 100

输出描述

输出在整个游戏过程中，小明所处的最大坐标值。异常情况下输出：12345

示例1

输入

```
2
1
-5 1
```

输出

```
0
```

说明

总共2个指令，幸运数为1，按照指令行进，依次如下游戏开始前，站在坐标轴原点，此时坐标值为0；
指令为-5，后退5步，此时坐标值为-5；
指令为1，正好等于幸运数，前进1+1=2步，此时坐标值为-3；
整个游戏过程中，小明所处的坐标值依次为[0，-5，-3]，最大坐标值为0。

示例2

输入

```
5
-5
-5 1 6 0 -7
```

输出

```
1
```

说明

总共5个指令，幸运数为-5，依照指令行进，依次如下：
游戏开始前，站在坐标轴原点，此时坐标值为0，
指令为-5，正好等于幸运数，后退5+1=6步，此时坐标值为-6；
指令为1，前进1步，此时坐标值为-5；
指令为6，前进6步，此时坐标值为1；

> 指令为0，既不前进，也不后退，此时坐标值为1；
> 指令为−7，后退7步，此时坐标值为−6。
> 整个游戏过程中，小明所处的坐标值依次为：
> [0, −6, −5, 1, 1, −6]，最大坐标值为1。

题解

思路： `模拟题`

简单模拟题，主要注意一下几点

1. 注意判断输入数据范围，是否超出指定题目描述要求。
2. 当指令==幸运数时
    - 幸运数 < 0：是向后多移动一步
    - 幸运数 > 0: 是向前移动一步
    - 幸运数 = 0: 不应该发生移动

> 明白两点之后，这道题就非常简单了。具体逻辑可参照下面代码。

C++

Plain Text

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

int main() {
    int n,m;
    cin >> n;
    cin >> m;
    // 异常
    if (n < 1 || n > 100 || m < -100 || m > 100) {
        cout << "12345";
        return 0;
    }
    int res = 0;
    int pos = 0;
    for (int i = 0; i < n; i++) {
        int command;
        cin >> command;
        // 异常
        if (command < -100 || command > 100) {
            cout << "12345";
            return 0;
        }

        // 等于幸运数字 幸运数字等于0的情况应该还是不移动的
        if (command == m) {
            if (command < 0) {
                command -= 1;
            } else if (command > 0) {
                command += 1;
            }
        }
        pos += command;
        // 尝试更新最大坐标数
        res = max(pos, res);
    }
    cout << res;
    return 0;
```

```
45    }
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(); // 命令数量
        int m = sc.nextInt(); // 幸运数字

        // 异常判断
        if (n < 1 || n > 100 || m < -100 || m > 100) {
            System.out.println("12345");
            return;
        }

        int res = 0;
        int pos = 0;

        for (int i = 0; i < n; i++) {
            int command = sc.nextInt();
            // 异常判断
            if (command < -100 || command > 100) {
                System.out.println("12345");
                return;
            }

            // 等于幸运数字 幸运数==0是不发生移动
            if (command == m) {
                if (command < 0) {
                    command -= 1;
                } else if (command > 0) {
                    command += 1;
                }

            }

            pos += command;
            res = Math.max(res, pos);
        }

        System.out.println(res);
    }
}
```

Python

Plain Text

```
# 读取输入
n = int(input())
m = int(input())
commands = list(map(int, input().split()))

# 异常检查
if n < 1 or n > 100 or m < -100 or m > 100 or len(commands) != n:
    print("12345")
    exit()

res = 0
pos = 0
for command in commands:
    if command < -100 or command > 100:
        print("12345")
        exit()
    # 幸运数字处理
    if command == m:
        if command > 0:
            command += 1
        elif command < 0:
            command -= 1
    pos += command
    res = max(res, pos)

print(res)
```

JavaScript

Plain Text

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

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

let lines = [];

rl.on('line', (line) => {
    lines.push(line.trim());
    if (lines.length === 3) {
        let n = parseInt(lines[0]);
        let m = parseInt(lines[1]);
        let commands = lines[2].split(' ').map(Number);
        // 输入异常
        if (n < 1 || n > 100 || m < -100 || m > 100 || commands.length !== n) {
            console.log("12345");
            return;
        }

        let pos = 0;
        let res = 0;

        for (let command of commands) {
            // 超过范围
            if (command < -100 || command > 100) {
                console.log("12345");
                return;
            }
            // 处理等于幸运数的情况
            if (command === m) {
                if (command > 0) {
                    command += 1;
                } else if (command < 0) {
                    command -= 1;
                }
            }
            pos += command;
            res = Math.max(res, pos);
        }

        console.log(res);
```

```
44            rl.close();
45        }
46    });
```

Go

Plain Text

```
package main

import (
    "fmt"
)

func main() {
    var n, m int
    fmt.Scan(&n)
    fmt.Scan(&m)
    // 超过范围
    if n < 1 || n > 100 || m < -100 || m > 100 {
        fmt.Println("12345")
        return
    }

    commands := make([]int, n)
    for i := 0; i < n; i++ {
        fmt.Scan(&commands[i])
        // 超过范围
        if commands[i] < -100 || commands[i] > 100 {
            fmt.Println("12345")
            return
        }
    }

    pos, res := 0, 0
    for _, command := range commands {
        // 处理等于幸运数的情况
        if command == m {
            if command > 0 {
                command++
            } else if command < 0 {
                command--
            }
        }
        pos += command
        if pos > res {
            res = pos
        }
    }

    fmt.Println(res)
}
```

小明的幸运数

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

> 华为OD机试2025C卷 100分题型

题目描述

小明在玩一个游戏，游戏规则如下：
在游戏开始前，小明站在坐标轴原点处（坐标值为0）.
给定一组指令和一个幸运数，每个指令都是一个整数，小明按照指令前进指定步数或者后退指定步数。前进代表朝坐标轴的正方向走，后退代表朝坐标轴的负方向走。
幸运数为一个整数，如果某个指令正好和幸运数相等，则小明行进步数+1。
例如：
幸运数为3，指令为[2,3,0,-5]
指令为2，表示前进2步；
指令为3，正好和幸运数相等，前进3+1=4步；
指令为0，表示原地不动，既不前进，也不后退。
指令为-5，表示后退5步。
请你计算小明在整个游戏过程中，小明所处的最大坐标值。

输入描述

第一行输入1个数字，代表指令的总个数 n（1 ≤ n ≤ 100）
第二行输入1个数字，代表幸运数m（-100 ≤ m ≤ 100）
第三行输入n个指令，每个指令的取值范围为：-100 ≤ 指令值 ≤ 100

输出描述

输出在整个游戏过程中，小明所处的最大坐标值。异常情况下输出：12345

示例1

输入

Plain Text

```
2
1
-5 1
```

输出

```
0
```

说明

总共2个指令，幸运数为1，按照指令行进，依次如下游戏开始前，站在坐标轴原点，此时坐标值为0；
指令为-5，后退5步，此时坐标值为-5；
指令为1，正好等于幸运数，前进1+1=2步，此时坐标值为-3；
整个游戏过程中，小明所处的坐标值依次为[0, -5, -3]，最大坐标值为0。

示例2

输入

```
5
-5
-5 1 6 0 -7
```

输出

```
1
```

说明

总共5个指令，幸运数为-5，依照指令行进，依次如下：
游戏开始前，站在坐标轴原点，此时坐标值为0，
指令为-5，正好等于幸运数，后退5+1=6步，此时坐标值为-6；
指令为1，前进1步，此时坐标值为-5；
指令为6，前进6步，此时坐标值为1；
指令为0，既不前进，也不后退，此时坐标值为1；
指令为-7，后退7步，此时坐标值为-6。
整个游戏过程中，小明所处的坐标值依次为：
[0, -6, -5, 1, 1, -6]，最大坐标值为1。

题解

思路： 模拟题

简单模拟题，主要注意一下几点

1. 注意判断输入数据范围，是否超出指定题目描述要求。
2. 当指令==幸运数时
    - 幸运数 < 0：是向后多移动一步
    - 幸运数 > 0: 是向前移动一步
    - 幸运数 = 0: 不应该发生移动

> 明白两点之后，这道题就非常简单了。具体逻辑可参照下面代码。

C++

Plain Text

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

int main() {
    int n,m;
    cin >> n;
    cin >> m;
    // 异常
    if (n < 1 || n > 100 || m < -100 || m > 100) {
        cout << "12345";
        return 0;
    }
    int res = 0;
    int pos = 0;
    for (int i = 0; i < n; i++) {
        int command;
        cin >> command;
        // 异常
        if (command < -100 || command > 100) {
            cout << "12345";
            return 0;
        }

        // 等于幸运数字 幸运数字等于0的情况应该还是不移动的
        if (command == m) {
            if (command < 0) {
                command -= 1;
            } else if (command > 0) {
                command += 1;
            }
        }
        pos += command;
        // 尝试更新最大坐标数
        res = max(pos, res);
    }
    cout << res;
    return 0;
```

```
45    }
```

Java

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(); // 命令数量
        int m = sc.nextInt(); // 幸运数字

        // 异常判断
        if (n < 1 || n > 100 || m < -100 || m > 100) {
            System.out.println("12345");
            return;
        }

        int res = 0;
        int pos = 0;

        for (int i = 0; i < n; i++) {
            int command = sc.nextInt();
            // 异常判断
            if (command < -100 || command > 100) {
                System.out.println("12345");
                return;
            }

            // 等于幸运数字 幸运数==0是不发生移动
            if (command == m) {
                if (command < 0) {
                    command -= 1;
                } else if (command > 0) {
                    command += 1;
                }

            }

            pos += command;
            res = Math.max(res, pos);
        }

        System.out.println(res);
    }
}
```

Python

```
# 读取输入
n = int(input())
m = int(input())
commands = list(map(int, input().split()))

# 异常检查
if n < 1 or n > 100 or m < -100 or m > 100 or len(commands) != n:
    print("12345")
    exit()

res = 0
pos = 0
for command in commands:
    if command < -100 or command > 100:
        print("12345")
        exit()
    # 幸运数字处理
    if command == m:
        if command > 0:
            command += 1
        elif command < 0:
            command -= 1
    pos += command
    res = max(res, pos)

print(res)
```

JavaScript

Plain Text

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

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

let lines = [];

rl.on('line', (line) => {
    lines.push(line.trim());
    if (lines.length === 3) {
        let n = parseInt(lines[0]);
        let m = parseInt(lines[1]);
        let commands = lines[2].split(' ').map(Number);
        // 输入异常
        if (n < 1 || n > 100 || m < -100 || m > 100 || commands.length !== n) {
            console.log("12345");
            return;
        }

        let pos = 0;
        let res = 0;

        for (let command of commands) {
            // 超过范围
            if (command < -100 || command > 100) {
                console.log("12345");
                return;
            }
            // 处理等于幸运数的情况
            if (command === m) {
                if (command > 0) {
                    command += 1;
                } else if (command < 0) {
                    command -= 1;
                }
            }
            pos += command;
            res = Math.max(res, pos);
        }

        console.log(res);
```

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

Go

Plain Text

```
package main

import (
    "fmt"
)

func main() {
    var n, m int
    fmt.Scan(&n)
    fmt.Scan(&m)
    // 超过范围
    if n < 1 || n > 100 || m < -100 || m > 100 {
        fmt.Println("12345")
        return
    }

    commands := make([]int, n)
    for i := 0; i < n; i++ {
        fmt.Scan(&commands[i])
        // 超过范围
        if commands[i] < -100 || commands[i] > 100 {
            fmt.Println("12345")
            return
        }
    }

    pos, res := 0, 0
    for _, command := range commands {
        // 处理等于幸运数的情况
        if command == m {
            if command > 0 {
                command++
            } else if command < 0 {
                command--
            }
        }
        pos += command
        if pos > res {
            res = pos
        }
    }

    fmt.Println(res)
}
```

> 来自: 华为OD机试2025C卷 – 小明的幸运数 (C++ & Python & JAVA & JS & GO)–CSDN博客

> 来自: 华为OD机试2025C卷 – 小明的幸运数 (C++ & Python & JAVA & JS & GO)–CSDN博客

华为OD机考2025C卷 - 报数问题 (C++ & Python & JAVA & JS & GO)-CSDN博客

报数问题

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

> 华为OD机试2025C卷 100分题型

题目描述

有n个人围成一圈，顺序排号为1-n。
从第一个人开始报数(从1到3报数)，凡报到3的人退出圈子，问最后留下的是原来第几号的那位。

输入描述

输入人数n（n < 1000）

输出描述

输出最后留下的是原来第几号

用例1

输入

```
Plain Text
1 2
```

输出

```
Plain Text
1 2
```

说明

> 报数序号为1的人最终报3，因此序号1的人退出圈子，最后剩下序号为2的那位

题解一

思路: `模拟题` 。使用链表数据结构进行模拟即可。考虑到本题较小，其实使用数组模拟也不会超时。

C++

Plain Text

```
#include<iostream>
#include <list>
using namespace std;

int main() {
    int m ;
    cin >> m;
    list<int> lst;
    for (int i = 1; i <= m; ++i) {
        lst.push_back(i);
    }
    // 当前轮报数为1的下标
    int pos = 0;
    while (lst.size() > 1) {
        pos = (pos + 3 - 1) % lst.size();
        // 移除指定位置的元素
        // 移除一个元素之后pos是不需要动的。[1,2,3,4], pos =2,移除3,[1,2,4],pos会自动指向4了
        std::list<int>::iterator it = lst.begin();
        std::advance(it, pos);
        lst.erase(it);
    }
    // 遍历列表
    cout << * (lst.begin());
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        LinkedList<Integer> lst = new LinkedList<>();
        for (int i = 1; i <= m; ++i) {
            lst.add(i);
        }

        // 当前轮报数为1的下标
        int pos = 0;
        while (lst.size() > 1) {
            pos = (pos + 3 - 1) % lst.size();  // 模拟每次报数3
            lst.remove(pos); // 移除第 pos 个元素
            // 移除后，pos 已经指向下一位了，不需要手动更新
        }
        // 输出剩下的最后一个元素
        System.out.println(lst.get(0));
    }
}
```

Python

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

def josephus(n, k):
    # 构建循环链表
    head = Node(1)
    tail = head
    for i in range(2, n + 1):
        tail.next = Node(i)
        tail = tail.next
    tail.next = head  # 成环

    # 模拟每数到第k个就出圈
    curr = head
    prev = tail
    count = 0

    while curr != curr.next:
        count += 1
        if count == k:
            # 删除当前节点
            prev.next = curr.next
            count = 0
        else:
            prev = curr
        curr = prev.next

    return curr.val

if __name__ == "__main__":
    m = int(input())
    print(josephus(m, 3))
```

JavaScript

Plain Text

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

rl.on('line', function (line) {
    const m = parseInt(line.trim());

    // 构造循环链表节点
    class Node {
        constructor(val) {
            this.val = val;
            this.next = null;
        }
    }

    let head = new Node(1), tail = head;
    for (let i = 2; i <= m; ++i) {
        tail.next = new Node(i);
        tail = tail.next;
    }
    tail.next = head; // 成环

    // 模拟报数
    let prev = tail, curr = head;
    let count = 0;
    // 成环情况 next == 本身 说明只有一个元素
    while (curr !== curr.next) {
        count++;
        if (count === 3) {
            // 删除 curr 节点
            prev.next = curr.next;
            count = 0;
        } else {
            prev = curr;
        }
        curr = prev.next;
    }

    console.log(curr.val);
    rl.close();
});
```

Go

Plain Text

```
package main

import (
    "fmt"
)

type Node struct {
    val  int
    next *Node
}

func main() {
    var m int
    fmt.Scan(&m)

    // 构造循环链表
    head := &Node{val: 1}
    tail := head
    for i := 2; i <= m; i++ {
        node := &Node{val: i}
        tail.next = node
        tail = node
    }
    tail.next = head // 成环

    // 模拟报数
    curr := head
    var prev *Node = tail
    count := 0
    // 成环情况 next == 本身 说明只有一个元素
    for curr != curr.next {
        count++
        if count == 3 {
            // 删除 curr
            prev.next = curr.next
            count = 0
        } else {
            prev = curr
        }
        curr = prev.next
    }
    fmt.Println(curr.val)
}
```

题解二

思路: `数学原理`

约瑟夫环问题其实存在递推公式 `f(n,m) = (f(n-1,m)+m) %n`

- f(N,M)表示，N个人报数，每报到M时杀掉那个人，最终胜利者的编号
- f(N−1,M)表示，N−1个人报数，每报到M时杀掉那个人，最终胜利者的编号

C++

Plain Text

```
#include<iostream>
#include <list>
using namespace std;

int main() {
    int n;
    cin >> n;
    // 从0开始
    int p = 0;
    // 递推公式
    for (int i = 2; i <= n; i++) {
        p = (p + 3) %i;
    }
    // 这是数组下标所以要加一
    cout << p +1;
}
```

JAVA

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int p = 0; // 从0开始编号

        // 根据递推公式计算最后留下的编号
        for (int i = 2; i <= n; i++) {
            p = (p + 3) % i;
        }

        // 最终编号是从0开始，所以输出要加1
        System.out.println(p + 1);
    }
}
```

Python

```
n = int(input())
p = 0  # 从0开始编号

# 使用递推公式求出最后留下的编号
for i in range(2, n + 1):
    p = (p + 3) % i

# 输出结果（从1开始编号）
print(p + 1)
```

JavaScript

Plain Text

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

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

let input = [];

rl.on('line', (line) => {
    input.push(line.trim());
    if (input.length === 1) {
        const n = parseInt(input[0]);
        let p = 0;

        // 递推公式计算约瑟夫环最后位置
        for (let i = 2; i <= n; i++) {
            p = (p + 3) % i;
        }

        // 输出结果（从1开始）
        console.log(p + 1);
        rl.close();
    }
});
```

Go

```plain text
package main

import (
  "fmt"
)

func main() {
  var n int
  fmt.Scan(&n)
  p := 0 // 从0开始编号

  // 使用递推公式计算
  for i := 2; i <= n; i++ {
    p = (p + 3) % i
  }

  // 输出结果（从1开始编号）
  fmt.Println(p + 1)
}
```

报数问题

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

> 华为OD机试2025C卷 100分题型

题目描述

有n个人围成一圈，顺序排号为1-n。

从第一个人开始报数(从1到3报数)，凡报到3的人退出圈子，问最后留下的是原来第几号的那位。

输入描述

输入人数n（n < 1000）

输出描述

输出最后留下的是原来第几号

用例1

输入

```
Plain Text
1 2
```

输出

```
Plain Text
1 2
```

说明

> 报数序号为1的人最终报3，因此序号1的人退出圈子，最后剩下序号为2的那位

题解一

思路：`模拟题`。使用链表数据结构进行模拟即可。考虑到本题较小，其实使用数组模拟也不会超时。

C++

Plain Text

```
#include<iostream>
#include <list>
using namespace std;

int main() {
    int m ;
    cin >> m;
    list<int> lst;
    for (int i = 1; i <= m; ++i) {
        lst.push_back(i);
    }
    // 当前轮报数为1的下标
    int pos = 0;
    while (lst.size() > 1) {
        pos = (pos + 3 - 1) % lst.size();
        // 移除指定位置的元素
        // 移除一个元素之后pos是不需要动的。[1,2,3,4], pos =2,移除3,[1,2,4],pos会自动指向4了
        std::list<int>::iterator it = lst.begin();
        std::advance(it, pos);
        lst.erase(it);
    }
    // 遍历列表
    cout << * (lst.begin());
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        LinkedList<Integer> lst = new LinkedList<>();
        for (int i = 1; i <= m; ++i) {
            lst.add(i);
        }

        // 当前轮报数为1的下标
        int pos = 0;
        while (lst.size() > 1) {
            pos = (pos + 3 - 1) % lst.size();  // 模拟每次报数3
            lst.remove(pos); // 移除第 pos 个元素
            // 移除后，pos 已经指向下一位了，不需要手动更新
        }
        // 输出剩下的最后一个元素
        System.out.println(lst.get(0));
    }
}
```

Python

Plain Text

```
class Node:
    def __init__(self, val):
        self.val = val
        self.next = None

def josephus(n, k):
    # 构建循环链表
    head = Node(1)
    tail = head
    for i in range(2, n + 1):
        tail.next = Node(i)
        tail = tail.next
    tail.next = head  # 成环

    # 模拟每数到第k个就出圈
    curr = head
    prev = tail
    count = 0

    while curr != curr.next:
        count += 1
        if count == k:
            # 删除当前节点
            prev.next = curr.next
            count = 0
        else:
            prev = curr
        curr = prev.next

    return curr.val

if __name__ == "__main__":
    m = int(input())
    print(josephus(m, 3))
```

JavaScript

Plain Text

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

rl.on('line', function (line) {
    const m = parseInt(line.trim());

    // 构造循环链表节点
    class Node {
        constructor(val) {
            this.val = val;
            this.next = null;
        }
    }

    let head = new Node(1), tail = head;
    for (let i = 2; i <= m; ++i) {
        tail.next = new Node(i);
        tail = tail.next;
    }
    tail.next = head; // 成环

    // 模拟报数
    let prev = tail, curr = head;
    let count = 0;
    // 成环情况 next == 本身 说明只有一个元素
    while (curr !== curr.next) {
        count++;
        if (count === 3) {
            // 删除 curr 节点
            prev.next = curr.next;
            count = 0;
        } else {
            prev = curr;
        }
        curr = prev.next;
    }

    console.log(curr.val);
    rl.close();
});
```

Go

Plain Text

```
package main

import (
    "fmt"
)

type Node struct {
    val  int
    next *Node
}

func main() {
    var m int
    fmt.Scan(&m)

    // 构造循环链表
    head := &Node{val: 1}
    tail := head
    for i := 2; i <= m; i++ {
        node := &Node{val: i}
        tail.next = node
        tail = node
    }
    tail.next = head // 成环

    // 模拟报数
    curr := head
    var prev *Node = tail
    count := 0
    // 成环情况 next == 本身 说明只有一个元素
    for curr != curr.next {
        count++
        if count == 3 {
            // 删除 curr
            prev.next = curr.next
            count = 0
        } else {
            prev = curr
        }
        curr = prev.next
    }
    fmt.Println(curr.val)
}
```

题解二

思路: `数学原理`

约瑟夫环问题其实存在递推公式 `f(n,m) = (f(n-1,m)+m) %n`

- f(N,M)表示，N个人报数，每报到M时杀掉那个人，最终胜利者的编号
- f(N−1,M)表示，N−1个人报数，每报到M时杀掉那个人，最终胜利者的编号

C++

Plain Text

```
#include<iostream>
#include <list>
using namespace std;

int main() {
    int n;
    cin >> n;
    // 从0开始
    int p = 0;
    // 递推公式
    for (int i = 2; i <= n; i++) {
        p = (p + 3) %i;
    }
    // 这是数组下标所以要加一
    cout << p +1;
}
```

JAVA

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int p = 0; // 从0开始编号

        // 根据递推公式计算最后留下的编号
        for (int i = 2; i <= n; i++) {
            p = (p + 3) % i;
        }

        // 最终编号是从0开始，所以输出要加1
        System.out.println(p + 1);
    }
}
```

Python

```
n = int(input())
p = 0  # 从0开始编号

# 使用递推公式求出最后留下的编号
for i in range(2, n + 1):
    p = (p + 3) % i

# 输出结果（从1开始编号）
print(p + 1)
```

JavaScript

Plain Text

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

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

let input = [];

rl.on('line', (line) => {
    input.push(line.trim());
    if (input.length === 1) {
        const n = parseInt(input[0]);
        let p = 0;

        // 递推公式计算约瑟夫环最后位置
        for (let i = 2; i <= n; i++) {
            p = (p + 3) % i;
        }

        // 输出结果（从1开始）
        console.log(p + 1);
        rl.close();
    }
});
```

Go

Plain Text

```
package main

import (
  "fmt"
)

func main() {
  var n int
  fmt.Scan(&n)
  p := 0 // 从0开始编号

  // 使用递推公式计算
  for i := 2; i <= n; i++ {
    p = (p + 3) % i
  }

  // 输出结果（从1开始编号）
  fmt.Println(p + 1)
}
```

> 来自: 华为OD机考2025C卷 – 报数问题 (C++ & Python & JAVA & JS & GO)-CSDN博客

> 来自: 华为OD机考2025C卷 – 报数问题 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试2025B卷 - 最少面试官 / 招聘 (C++ & Python & JAVA & JS & GO)-CSDN博客

最少面试官 / 招聘

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> 华为OD机试2025B卷 200分题型

题目描述

某公司组织一场公开招聘活动，假设由于人数和场地的限制，每人每次面试的时长不等，并已经安排给定，用(S1,E1)、(S2,E2)、(Sj,Ej)...(Si < Ei，均为非负整数)表示每场面试的开始和结束时间。
面试采用一对一的方式，即一名面试官同时只能面试一名应试者，一名面试官完成一次面试后可以立即进行下一场面试，且每个面试官的面试人次不超过 m。
为了支撑招聘活动高效顺利进行，请你计算至少需要多少名面试官。

输入描述

输入的第一行为面试官的最多面试人次 m，第二行为当天总的面试场次 n，
接下来的 n 行为每场面试的起始时间和结束时间，起始时间和结束时间用空格分隔。
其中， 1 <= n, m <= 500

输出描述

输出一个整数，表示至少需要的面试官数量。

用例1

输入

Plain Text

```
2
5
1 2
2 3
3 4
4 5
5 6
```

输出

```
3
```

说明

> 总共有 5 场面试，且面试时间都不重叠，但每个面试官最多只能面试 2 人次，所以需要 3 名面试官。

用例2

输入

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

输出

```
1
```

说明

> 总共有3场面试，面试时间不重叠，每个面试官最多能面试3人次，所以只需要一名面试官

用例3

输入

```
3
3
8 35
5 10
1 3
```

输出

```
2
```

说明

```
总共有3场面试，[5,10]和[8,35]有重叠，所以至少需要2名面试官
```

题解

思路： `二分 + 优先队列`

1. 将输入面试按照 `开始时间升序，开始时间相等情况下降序` 排序。
2. 二分枚举面试官数，通过收缩边界确定最小面试官时，初始定义 `left = (n+m-1)/m, right = n` 。left是根据每个面试官最多处理面试数量取的(向上取整)。每次枚举 `mid = (left + right) /2` 作为尝试面试官数，二分检测逻辑大致如下：
   a. 定义优先队列 `pq` ，pq中存储 `{endTime（上一次面试结束时间）, meetingCount(已进行面试次数)}` ,小顶堆，上一次面试结束时间小的位于栈顶。
   b. 接下来就是遍历排序后的面试，判断优先队列栈顶面试官能否处理这个面试，不能则返回false。能则更新，栈顶元素的面试结束时间和执行面试次数。如果已进行面试次数 < m则重新入队。
   c. 如果枚举的mid面试官能处理完所有面试则进行右边界收缩 `right = mid` ,否则进行左边界收缩 `left = mid + 1`
   d. 重复以上逻辑 `left == right` 结束二分，此时left就是结果。

C++

Plain Text

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

// 按照结束时间升序排序
bool cmp(pair<int, int>& p1, pair<int, int>& p2) {
    if (p1.first == p2.first) {
        return p1.second < p2.second;
    }
    return p1.first < p2.first;
}

bool check(vector<pair<int,int>>& times, int count, int m) {
   // 优先队列中存储{上次面试结束时间, 已面试次数} 面试结束小的属于
   priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
   for (int i = 0 ; i < count; i++) {
        pq.push({0,0});
    }

    int n = times.size();
    for (int i = 0; i < n; i++) {
        // 没有面试官可用
        if (pq.empty()) {
            return false;
        }
        pair<int,int> top = pq.top();
        pq.pop();
        // 栈顶的结束时间最小的面试官 还是无法满足当前面试 则分配失败
        if (top.first > times[i].first) {
            return false;
        }
        top.first = times[i].second;
        top.second += 1;
        // 该面试官完成m个面试不考虑了
        if (top.second < m) {
            pq.push(top);
```

```
            }
        }

        return true;
    }


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

        vector<pair<int, int>> times(n);
        for (int i = 0; i < n; i++) {
            int start, end;
            cin >> start >> end;
            times[i] = {start, end};
        }
        // 自定义排序
        sort(times.begin(), times.end(), cmp);

        int left = ceil(1.0 * n / m);
        int right = n;
        // 二分枚举
        while (left < right) {
            int mid = (left + right) >> 1;
            if(check(times, mid, m)) {
                right = mid;
            } else {
                left = mid  + 1;
            }
        }

        cout << left;
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static boolean check(List<int[]> times, int count, int m) {
        // 小顶堆，按结束时间排序 {endTime, meetingCount}
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> a[0] - b
[0]);
        for (int i = 0; i < count; i++) {
            pq.offer(new int[]{0, 0});
        }

        for (int[] time : times) {
            if (pq.isEmpty()) return false;

            int[] top = pq.poll();
            // 当前最早结束的面试官都无法安排
            if (top[0] > time[0]) return false;

            top[0] = time[1];
            top[1]++;
            if (top[1] < m) {
                pq.offer(top);
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        int n = sc.nextInt();

        List<int[]> times = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            int start = sc.nextInt();
            int end = sc.nextInt();
            times.add(new int[]{start, end});
        }

        // 起始时间升序，若相同结束时间升序
        times.sort((a, b) -> a[0] != b[0] ? a[0] - b[0] : a[1] - b[1]);

        int left = (int) Math.ceil((double) n / m);
```

```
44          int right = n;
45
46          while (left < right) {
47              int mid = (left + right) / 2;
48              if (check(times, mid, m)) {
49                  right = mid;
50              } else {
51                  left = mid + 1;
52              }
53          }
54
55          System.out.println(left);
56      }
57  }
```

Python

```
import heapq
import math

def check(times, count, m):
    # 小顶堆：存放 (结束时间, 面试次数)
    pq = [(0, 0) for _ in range(count)]
    heapq.heapify(pq)

    for start, end in times:
        if not pq:
            return False
        top_end, cnt = heapq.heappop(pq)
        # 该面试无法分配
        if top_end > start:
            return False
        heapq.heappush(pq, (end, cnt + 1) if cnt + 1 < m else (float('in
f'), cnt + 1))
    return True

def main():
    m = int(input())
    n = int(input())
    times = [tuple(map(int, input().split())) for _ in range(n)]

    # 起始时间升序，若相同则结束时间升序
    times.sort(key=lambda x: (x[0], x[1]))

    left = (n + m -1) // m
    right = n

    while left < right:
        mid = (left + right) // 2
        if check(times, mid, m):
            right = mid
        else:
            left = mid + 1
    print(left)

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

JavaScript

Plain Text

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

// 自定义小根堆（优先队列）实现
class PriorityQueue {
    constructor(comparator = (a, b) => a[0] - b[0]) {
        this._heap = [];
        this._comparator = comparator;
    }

    size() {
        return this._heap.length;
    }

    isEmpty() {
        return this.size() === 0;
    }

    peek() {
        return this._heap[0];
    }

    push(value) {
        this._heap.push(value);
        this._siftUp();
    }

    pop() {
        const top = this.peek();
        const bottom = this._heap.pop();
        if (this.size()) {
            this._heap[0] = bottom;
            this._siftDown();
        }
        return top;
    }

    _siftUp() {
        let nodeIdx = this.size() - 1;
        while (nodeIdx > 0) {
            const parentIdx = (nodeIdx - 1) >> 1;
            if (this._comparator(this._heap[nodeIdx], this._heap[parentId
x]) >= 0) break;
            [this._heap[nodeIdx], this._heap[parentIdx]] = [this._heap[pa
rentIdx], this._heap[nodeIdx]];
```

```
            nodeIdx = parentIdx;
        }
    }

    _siftDown() {
        let nodeIdx = 0;
        const length = this.size();
        while (true) {
            const leftIdx = (nodeIdx << 1) + 1;
            const rightIdx = leftIdx + 1;
            let smallestIdx = nodeIdx;

            if (leftIdx < length && this._comparator(this._heap[leftId
x], this._heap[smallestIdx]) < 0) {
                smallestIdx = leftIdx;
            }
            if (rightIdx < length && this._comparator(this._heap[rightId
x], this._heap[smallestIdx]) < 0) {
                smallestIdx = rightIdx;
            }
            if (smallestIdx === nodeIdx) break;
            [this._heap[nodeIdx], this._heap[smallestIdx]] = [this._heap
[smallestIdx], this._heap[nodeIdx]];
            nodeIdx = smallestIdx;
        }
    }
}

// 面试安排检测函数
function check(times, count, m) {
    const pq = new PriorityQueue((a, b) => a[0] - b[0]); // 小根堆，按结束
时间排序

    // 初始化 count 个面试官
    for (let i = 0; i < count; i++) {
        pq.push([0, 0]); // [面试结束时间, 已面试次数]
    }

    for (const [start, end] of times) {
        if (pq.isEmpty()) return false;

        const [prevEnd, cnt] = pq.pop();
        // 没有面试官空闲
        if (prevEnd > start) return false;

        const next = [end, cnt + 1];
        if (next[1] < m) {
```

```
            pq.push(next);
        }
    }

    return true;
}

// 主逻辑
function main(m, n, times) {
    // 排序规则：先按开始时间升序，开始时间相同的，结束时间升序
    times.sort((a, b) => a[0] - b[0] || a[1] - b[1]);

    let left = Math.ceil(n / m);
    let right = n;

    while (left < right) {
        const mid = (left + right) >> 1;
        if (check(times, mid, m)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    console.log(left);
}

// 使用 readline 接收输入（ACM 模式）
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const inputLines = [];
rl.on('line', (line) => {
    inputLines.push(line.trim());
}).on('close', () => {
    const m = parseInt(inputLines[0]);
    const n = parseInt(inputLines[1]);
    const times = inputLines.slice(2, 2 + n).map(line => line.split(' ').map(Number));
    main(m, n, times);
});
```

Go

Plain Text

```
package main

import (
  "container/heap"
  "fmt"
  "math"
  "sort"
)

type Pair struct {
  endTime int
  count   int
}

type MinHeap []Pair

func (h MinHeap) Len() int { return len(h) }
func (h MinHeap) Less(i, j int) bool {
  return h[i].endTime < h[j].endTime
}
func (h MinHeap) Swap(i, j int) {
  h[i], h[j] = h[j], h[i]
}
func (h *MinHeap) Push(x interface{}) {
  *h = append(*h, x.(Pair))
}
func (h *MinHeap) Pop() interface{} {
  old := *h
  n := len(old)
  top := old[n-1]
  *h = old[0 : n-1]
  return top
}

func check(times [][2]int, count int, m int) bool {
    // 小顶堆：存放 (结束时间, 面试次数)
  h := &MinHeap{}
  for i := 0; i < count; i++ {
    heap.Push(h, Pair{0, 0})
  }

  for _, t := range times {
    if h.Len() == 0 {
      return false
```

```
      }
      top := heap.Pop(h).(Pair)
          // 该面试无法分配
      if top.endTime > t[0] {
        return false
      }
      top.endTime = t[1]
      top.count++
          // 执行面试 >= m的不入队
      if top.count < m {
        heap.Push(h, top)
      }
    }
    return true
  }

  func main() {
    var m, n int
    fmt.Scan(&m, &n)
    times := make([][2]int, n)
    for i := 0; i < n; i++ {
      fmt.Scan(&times[i][0], &times[i][1])
    }
      // 起始时间升序，若相同则结束时间升序
    sort.Slice(times, func(i, j int) bool {
      if times[i][0] == times[j][0] {
        return times[i][1] < times[j][1]
      }
      return times[i][0] < times[j][0]
    })

    left := int(math.Ceil(float64(n) / float64(m)))
    right := n

    for left < right {
      mid := (left + right) / 2
      if check(times, mid, m) {
        right = mid
      } else {
        left = mid + 1
      }
    }
    fmt.Println(left)
  }
```

最少面试官 / 招聘

华为OD机试真题目录点击查看: 华为OD机试2025B卷真题题库目录｜机考题库 + 算法考点详解

> 华为OD机试2025B卷 200分题型

题目描述

某公司组织一场公开招聘活动，假设由于人数和场地的限制，每人每次面试的时长不等，并已经安排给定，用(S1,E1)、(S2,E2)、(Sj,Ej)...(Si < Ei，均为非负整数)表示每场面试的开始和结束时间。
面试采用一对一的方式，即一名面试官同时只能面试一名应试者，一名面试官完成一次面试后可以立即进行下一场面试，且每个面试官的面试人次不超过 m。
为了支撑招聘活动高效顺利进行，请你计算至少需要多少名面试官。

输入描述

输入的第一行为面试官的最多面试人次 m，第二行为当天总的面试场次 n，
接下来的 n 行为每场面试的起始时间和结束时间，起始时间和结束时间用空格分隔。
其中， 1 <= n, m <= 500

输出描述

输出一个整数，表示至少需要的面试官数量。

用例1

输入

Plain Text

```
2
5
1 2
2 3
3 4
4 5
5 6
```

输出

Plain Text

```
3
```

说明

> 总共有 5 场面试，且面试时间都不重叠，但每个面试官最多只能面试 2 人次，所以需要 3 名面试官。

用例2

输入

```Plain Text
3
3
1 2
2 3
3 4
```

输出

```Plain Text
1
```

说明

> 总共有3场面试，面试时间不重叠，每个面试官最多能面试3人次，所以只需要一名面试官

用例3

输入

```Plain Text
3
3
8 35
5 10
1 3
```

输出

```Plain Text
2
```

说明

```
1  总共有3场面试，[5,10]和[8,35]有重叠，所以至少需要2名面试官
```

题解

思路：`二分 + 优先队列`

1. 将输入面试按照`开始时间升序，开始时间相等情况下降序`排序。
2. 二分枚举面试官数，通过收缩边界确定最小面试官时，初始定义`left = (n+m-1)/m, right = n`。left是根据每个面试官最多处理面试数量取的(向上取整)。每次枚举`mid = (left + right) /2`作为尝试面试官数，二分检测逻辑大致如下：
   a. 定义优先队列`pq`，pq中存储`{endTime（上一次面试结束时间）, meetingCount(已进行面试次数)}`,小顶堆，上一次面试结束时间小的位于栈顶。
   b. 接下来就是遍历排序后的面试，判断优先队列栈顶面试官能否处理这个面试，不能则返回false。能则更新，栈顶元素的面试结束时间和执行面试次数。如果已进行面试次数 < m则重新入队。
   c. 如果枚举的mid面试官能处理完所有面试则进行右边界收缩`right = mid`,否则进行左边界收缩`left = mid + 1`
   d. 重复以上逻辑`left == right`结束二分，此时left就是结果。

C++

Plain Text

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

// 按照结束时间升序排序
bool cmp(pair<int, int>& p1, pair<int, int>& p2) {
    if (p1.first == p2.first) {
        return p1.second < p2.second;
    }
    return p1.first < p2.first;
}

bool check(vector<pair<int,int>>& times, int count, int m) {
   // 优先队列中存储{上次面试结束时间, 已面试次数} 面试结束小的属于
   priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;
   for (int i = 0 ; i < count; i++) {
        pq.push({0,0});
    }

    int n = times.size();
    for (int i = 0; i < n; i++) {
        // 没有面试官可用
        if (pq.empty()) {
            return false;
        }
        pair<int,int> top = pq.top();
        pq.pop();
        // 栈顶的结束时间最小的面试官 还是无法满足当前面试 则分配失败
        if (top.first > times[i].first) {
            return false;
        }
        top.first = times[i].second;
        top.second += 1;
        // 该面试官完成m个面试不考虑了
        if (top.second < m) {
            pq.push(top);
```

```
            }
        }

        return true;
    }


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

        vector<pair<int, int>> times(n);
        for (int i = 0; i < n; i++) {
            int start, end;
            cin >> start >> end;
            times[i] = {start, end};
        }
        // 自定义排序
        sort(times.begin(), times.end(), cmp);

        int left = ceil(1.0 * n / m);
        int right = n;
        // 二分枚举
        while (left < right) {
            int mid = (left + right) >> 1;
            if(check(times, mid, m)) {
                right = mid;
            } else {
                left = mid  + 1;
            }
        }

        cout << left;
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static boolean check(List<int[]> times, int count, int m) {
        // 小顶堆，按结束时间排序 {endTime, meetingCount}
        PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> a[0] - b
[0]);
        for (int i = 0; i < count; i++) {
            pq.offer(new int[]{0, 0});
        }

        for (int[] time : times) {
            if (pq.isEmpty()) return false;

            int[] top = pq.poll();
            // 当前最早结束的面试官都无法安排
            if (top[0] > time[0]) return false;

            top[0] = time[1];
            top[1]++;
            if (top[1] < m) {
                pq.offer(top);
            }
        }

        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int m = sc.nextInt();
        int n = sc.nextInt();

        List<int[]> times = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            int start = sc.nextInt();
            int end = sc.nextInt();
            times.add(new int[]{start, end});
        }

        // 起始时间升序，若相同结束时间升序
        times.sort((a, b) -> a[0] != b[0] ? a[0] - b[0] : a[1] - b[1]);

        int left = (int) Math.ceil((double) n / m);
```

```
44          int right = n;
45
46          while (left < right) {
47              int mid = (left + right) / 2;
48              if (check(times, mid, m)) {
49                  right = mid;
50              } else {
51                  left = mid + 1;
52              }
53          }
54
55          System.out.println(left);
56      }
57  }
```

Python

```
import heapq
import math

def check(times, count, m):
    # 小顶堆：存放 (结束时间, 面试次数)
    pq = [(0, 0) for _ in range(count)]
    heapq.heapify(pq)

    for start, end in times:
        if not pq:
            return False
        top_end, cnt = heapq.heappop(pq)
        # 该面试无法分配
        if top_end > start:
            return False
        heapq.heappush(pq, (end, cnt + 1) if cnt + 1 < m else (float('in
f'), cnt + 1))
    return True

def main():
    m = int(input())
    n = int(input())
    times = [tuple(map(int, input().split())) for _ in range(n)]

    # 起始时间升序，若相同则结束时间升序
    times.sort(key=lambda x: (x[0], x[1]))

    left = (n + m -1) // m
    right = n

    while left < right:
        mid = (left + right) // 2
        if check(times, mid, m):
            right = mid
        else:
            left = mid + 1
    print(left)

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

JavaScript

Plain Text

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

// 自定义小根堆（优先队列）实现
class PriorityQueue {
    constructor(comparator = (a, b) => a[0] - b[0]) {
        this._heap = [];
        this._comparator = comparator;
    }

    size() {
        return this._heap.length;
    }

    isEmpty() {
        return this.size() === 0;
    }

    peek() {
        return this._heap[0];
    }

    push(value) {
        this._heap.push(value);
        this._siftUp();
    }

    pop() {
        const top = this.peek();
        const bottom = this._heap.pop();
        if (this.size()) {
            this._heap[0] = bottom;
            this._siftDown();
        }
        return top;
    }

    _siftUp() {
        let nodeIdx = this.size() - 1;
        while (nodeIdx > 0) {
            const parentIdx = (nodeIdx - 1) >> 1;
            if (this._comparator(this._heap[nodeIdx], this._heap[parentId
x]) >= 0) break;
            [this._heap[nodeIdx], this._heap[parentIdx]] = [this._heap[pa
rentIdx], this._heap[nodeIdx]];
```

```
            nodeIdx = parentIdx;
        }
    }

    _siftDown() {
        let nodeIdx = 0;
        const length = this.size();
        while (true) {
            const leftIdx = (nodeIdx << 1) + 1;
            const rightIdx = leftIdx + 1;
            let smallestIdx = nodeIdx;

            if (leftIdx < length && this._comparator(this._heap[leftId
x], this._heap[smallestIdx]) < 0) {
                smallestIdx = leftIdx;
            }
            if (rightIdx < length && this._comparator(this._heap[rightId
x], this._heap[smallestIdx]) < 0) {
                smallestIdx = rightIdx;
            }
            if (smallestIdx === nodeIdx) break;
            [this._heap[nodeIdx], this._heap[smallestIdx]] = [this._heap
[smallestIdx], this._heap[nodeIdx]];
            nodeIdx = smallestIdx;
        }
    }
}

// 面试安排检测函数
function check(times, count, m) {
    const pq = new PriorityQueue((a, b) => a[0] - b[0]); // 小根堆，按结束
时间排序

    // 初始化 count 个面试官
    for (let i = 0; i < count; i++) {
        pq.push([0, 0]); // [面试结束时间, 已面试次数]
    }

    for (const [start, end] of times) {
        if (pq.isEmpty()) return false;

        const [prevEnd, cnt] = pq.pop();
        // 没有面试官空闲
        if (prevEnd > start) return false;

        const next = [end, cnt + 1];
        if (next[1] < m) {
```

```
            pq.push(next);
        }
    }

    return true;
}

// 主逻辑
function main(m, n, times) {
    // 排序规则：先按开始时间升序，开始时间相同的，结束时间升序
    times.sort((a, b) => a[0] - b[0] || a[1] - b[1]);

    let left = Math.ceil(n / m);
    let right = n;

    while (left < right) {
        const mid = (left + right) >> 1;
        if (check(times, mid, m)) {
            right = mid;
        } else {
            left = mid + 1;
        }
    }

    console.log(left);
}

// 使用 readline 接收输入（ACM 模式）
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const inputLines = [];
rl.on('line', (line) => {
    inputLines.push(line.trim());
}).on('close', () => {
    const m = parseInt(inputLines[0]);
    const n = parseInt(inputLines[1]);
    const times = inputLines.slice(2, 2 + n).map(line => line.split(' ').
map(Number));
    main(m, n, times);
});
```

Go

Plain Text

```
package main

import (
  "container/heap"
  "fmt"
  "math"
  "sort"
)

type Pair struct {
  endTime int
  count   int
}

type MinHeap []Pair

func (h MinHeap) Len() int { return len(h) }
func (h MinHeap) Less(i, j int) bool {
  return h[i].endTime < h[j].endTime
}
func (h MinHeap) Swap(i, j int) {
  h[i], h[j] = h[j], h[i]
}
func (h *MinHeap) Push(x interface{}) {
  *h = append(*h, x.(Pair))
}
func (h *MinHeap) Pop() interface{} {
  old := *h
  n := len(old)
  top := old[n-1]
  *h = old[0 : n-1]
  return top
}

func check(times [][2]int, count int, m int) bool {
    // 小顶堆：存放 (结束时间, 面试次数)
  h := &MinHeap{}
  for i := 0; i < count; i++ {
    heap.Push(h, Pair{0, 0})
  }

  for _, t := range times {
    if h.Len() == 0 {
      return false
```

```
    }
    top := heap.Pop(h).(Pair)
        // 该面试无法分配
    if top.endTime > t[0] {
      return false
    }
    top.endTime = t[1]
    top.count++
        // 执行面试 >= m的不入队
    if top.count < m {
      heap.Push(h, top)
    }
  }
  return true
}

func main() {
  var m, n int
  fmt.Scan(&m, &n)
  times := make([][2]int, n)
  for i := 0; i < n; i++ {
    fmt.Scan(&times[i][0], &times[i][1])
  }
    // 起始时间升序，若相同则结束时间升序
  sort.Slice(times, func(i, j int) bool {
    if times[i][0] == times[j][0] {
      return times[i][1] < times[j][1]
    }
    return times[i][0] < times[j][0]
  })

  left := int(math.Ceil(float64(n) / float64(m)))
  right := n

  for left < right {
    mid := (left + right) / 2
    if check(times, mid, m) {
      right = mid
    } else {
      left = mid + 1
    }
  }
  fmt.Println(left)
}
```

> 来自: 华为OD机试2025B卷 – 最少面试官 / 招聘 (C++ & Python & JAVA & JS & GO)-CSDN博客

> 来自: 华为OD机试2025B卷 – 最少面试官 / 招聘 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试 2025B卷 - 上班之路 (C++ & Python & JAVA & JS & GO)_华为od2025b卷-CSDN博客

上班之路

华为OD机试真题目录: 点击去查看

> 华为od2025B卷 200分题型

题目描述

Jungle 生活在美丽的蓝鲸城，大马路都是方方正正，但是每天马路的封闭情况都不一样。
地图由以下元素组成：

1. ”.” — 空地，可以达到;
2. ” * ” — 路障，不可达到;
3. ”S” — Jungle的家;
4. ”T” — 公司.

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

输入描述

输入的第一行为两个整数t,c（0 ⩽ t,c ⩽ 100）,t代表可以拐弯的次数，c代表可以清除的路障个数。
输入的第二行为两个整数n,m（1 ⩽ n,m ⩽ 100）,代表地图的大小。
接下来是n行包含m个字符的地图。n和m可能不一样大。
我们保证地图里有S和T。

输出描述

输出是否可以从家里出发到达公司，是则输出YES，不能则输出NO。

用例1

输入

```
2 0
5 5
..S..
****.
T....
****.
.....
```

输出

```
YES
```

用例2

输入

```
1 2
5 5
.*S*.
*****
..*..
*****
T....
```

输出

```
NO
```

说明

> 该用例中，至少需要拐弯1次，清除3个路障，所以无法到达

题解

思路：`记忆化搜索 + BFS` 求解

1. 这道题使用常规的 `递归回溯` 方法在大数据集时，会出现爆内存以及超时的情况。需要采用 `记忆化搜索 + BFS` 方式进行求解。
2. 记忆化搜索：定义 `visited[pos][dir][2]` 记录放到此处时的最优状态，初始数组所有值设置为大值，其中 `visited[pos][dir][0] = 该位置该方向下最小拐弯数, visited[pos][dir][1] = 该位置该方向下最小破墙数` .用于后续BFS遍历过程的剪枝。这个数组定义已经对位置进行了压缩 二维转换成一维 , `pos = x * m + y`
3. 每一次移动都具备 `方向性、已拐弯、已破墙` 的状态，所以在进行 `BFS` 遍历时候需要维持这些状态。并通过 `visited` 进行剪枝。具体逻辑如下：
    a. 对于方向判断，我们可以定义把方向转换为 `上0，下1，左2，右3` ，如果当前探寻方向和上一次方向不一致则拐弯+1
    b. 可以使用队列进行模拟BFS过程，队列中存储 `int x, y, dir, usedT, usedC` 的类似结构可以定义类或者使用数组保存。分别表示 `当前x坐标，当前y坐标，行走的方向，已经使用的拐弯数，已经使用的破墙数` 。初始往队列中插入起点的四个方向状态(起点可以以任意方向开始)
    c. 循环迭代进行队列中元素四方向探寻，尝试方向 `currentDir != lastDir` 时usedT + 1, 当前遇到路障时 usedC+1. 并进行剪枝优化
        i. 探寻方向坐标越界剪枝。
        ii. 到达探寻方向需要拐弯的次数大于t或者需要清除路障数大于c。剪枝
        iii. 利用 `visisted` 数组剪枝逻辑，你已这个方向到达这个位置，拐弯和碰撞你肯定要有一个优于之前才能进行继续探寻。
    d. 终止条件：迭代到终点(结果为true) 或者队列为空(结果为false)

C++

Plain Text

```
#include <iostream>
#include <vector>
#include <queue>
#include <tuple>
#include <cstring>
using namespace std;

int t, c; // 最大拐弯次数t，最大破墙数c
int n, m; // 地图大小
vector<vector<char>> grid;

// 方向定义：上0，下1，左2，右3
int dx[4] = {-1, 1, 0, 0};
int dy[4] = {0, 0, -1, 1};

// 状态压缩 二维 压成 一维
// visited[pos][dir][0] = 该位置该方向下最小拐弯数
// visited[pos][dir][1] = 该位置该方向下最小破墙数
int visited[10000][4][2];

struct State {
    int x, y, dir, usedT, usedC;
};

bool bfs(int startX, int startY) {
    memset(visited, 0x3f, sizeof(visited)); // 初始化为大数

    queue<State> q;

    // 起点四个方向均可，拐弯数0，破墙数0
    for (int dir = 0; dir < 4; dir++) {
        visited[startX * m + startY][dir][0] = 0;
        visited[startX * m + startY][dir][1] = 0;
        q.push({startX, startY, dir, 0, 0});
    }

    while (!q.empty()) {
        auto [x, y, dir, usedT, usedC] = q.front();
        q.pop();

        if (grid[x][y] == 'T') return true;

        for (int ndir = 0; ndir < 4; ndir++) {
            int nx = x + dx[ndir];
```

```
45              int ny = y + dy[ndir];
46              // 超过边界
47              if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
48
49              int nUsedT = usedT;
50              if (ndir != dir) nUsedT++; // 拐弯
51              // 超过最大拐弯
52              if (nUsedT > t) continue;
53
54              int nUsedC = usedC;
55              if (grid[nx][ny] == '*') nUsedC++;
56              // 超过最大破墙
57              if (nUsedC > c) continue;
58
59              int pos = nx * m + ny;
60
61              // 剪枝逻辑就是你已这个方向到达这个位置，拐弯和碰撞你肯定要有一个优于之前才能进行下去
62              // 剪枝：新状态如果不优于旧状态，则跳过
63              if (nUsedT > visited[pos][ndir][0] && nUsedC >= visited[pos][ndir][1]) continue;
64              if (nUsedT >= visited[pos][ndir][0] && nUsedC > visited[pos][ndir][1]) continue;
65
66              if (nUsedT == visited[pos][ndir][0] && nUsedC == visited[pos][ndir][1]) continue;
67
68              //上述剪枝可以写成这样的哈，为了把情况给各位分清楚所以分步了
69              // if (nUsedT >= visited[pos][ndir][0] && nUsedC >= visited[pos][ndir][1]) continue;
70
71
72              // 更新状态
73              if (nUsedT < visited[pos][ndir][0]) visited[pos][ndir][0] = nUsedT;
74              if (nUsedC < visited[pos][ndir][1]) visited[pos][ndir][1] = nUsedC;
75
76              q.push({nx, ny, ndir, nUsedT, nUsedC});
77          }
78      }
79      return false;
80  }
81
82  int main() {
83      ios::sync_with_stdio(false);
84      cin.tie(nullptr);
```

```
85
86      cin >> t >> c;
87      cin >> n >> m;
88      grid.resize(n, vector<char>(m));
89      int startX = -1, startY = -1;
90      for (int i = 0; i < n; i++)
91          for (int j = 0; j < m; j++) {
92              cin >> grid[i][j];
93              if (grid[i][j] == 'S') {
94                  startX = i;
95                  startY = j;
96              }
97          }
98
99      bool ans = bfs(startX, startY);
100     cout << (ans ? "YES" : "NO") << "\n";
101     return 0;
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static int t, c, n, m; // 最大拐弯次数t，最大破墙数c，地图大小n行m列
    static char[][] grid;
    static int[][][] visited; // visited[pos][dir][0] = 最小拐弯数, visited[pos][dir][1] = 最小破墙数

    static int[] dx = {-1, 1, 0, 0}; // 上下左右
    static int[] dy = {0, 0, -1, 1};

    static class State {
        int x, y, dir, usedT, usedC;
        State(int x, int y, int dir, int usedT, int usedC) {
            this.x = x; this.y = y; this.dir = dir; this.usedT = usedT; this.usedC = usedC;
        }
    }

    static boolean bfs(int startX, int startY) {
        visited = new int[n * m][4][2];
        for (int[][] v : visited)
            for (int[] d : v)
                Arrays.fill(d, Integer.MAX_VALUE); // 初始化为大数

        Queue<State> q = new LinkedList<>();

        // 起点四个方向均可，拐弯数0，破墙数0
        for (int dir = 0; dir < 4; dir++) {
            visited[startX * m + startY][dir][0] = 0;
            visited[startX * m + startY][dir][1] = 0;
            q.add(new State(startX, startY, dir, 0, 0));
        }

        while (!q.isEmpty()) {
            State s = q.poll();
            int x = s.x, y = s.y, dir = s.dir, usedT = s.usedT, usedC = s.usedC;

            if (grid[x][y] == 'T') return true;

            for (int ndir = 0; ndir < 4; ndir++) {
                int nx = x + dx[ndir];
                int ny = y + dy[ndir];
```

```
42                  if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
43
44                  int nUsedT = usedT;
45                  if (ndir != dir) nUsedT++; // 拐弯
46                  if (nUsedT > t) continue;
47
48                  int nUsedC = usedC;
49                  if (grid[nx][ny] == '*') nUsedC++;
50                  if (nUsedC > c) continue;
51
52                  int pos = nx * m + ny;
53
54                  // 剪枝逻辑就是你已这个方向到达这个位置，拐弯和碰撞你肯定要有一个优于之前才能进行下去
55                  // 剪枝：新状态如果不优于旧状态，则跳过
56                  if (nUsedT > visited[pos][ndir][0] && nUsedC >= visited[pos][ndir][1]) continue;
57                  if (nUsedT >= visited[pos][ndir][0] && nUsedC > visited[pos][ndir][1]) continue;
58                  if (nUsedT == visited[pos][ndir][0] && nUsedC == visited[pos][ndir][1]) continue;
59
60                  //上述剪枝可以写成这样的哈，为了把情况给各位分清楚所以分步了
61                  // if (nUsedT >= visited[pos][ndir][0] && nUsedC >= visited[pos][ndir][1]) continue;
62
63                  if (nUsedT < visited[pos][ndir][0]) visited[pos][ndir][0] = nUsedT;
64                  if (nUsedC < visited[pos][ndir][1]) visited[pos][ndir][1] = nUsedC;
65
66                  q.add(new State(nx, ny, ndir, nUsedT, nUsedC));
67              }
68          }
69          return false;
70      }
71
72      public static void main(String[] args) {
73          Scanner sc = new Scanner(System.in);
74          t = sc.nextInt();
75          c = sc.nextInt();
76          n = sc.nextInt();
77          m = sc.nextInt();
78          grid = new char[n][m];
79          int startX = -1, startY = -1;
80
81          for (int i = 0; i < n; i++) {
```

```
            String line = sc.next();
            for (int j = 0; j < m; j++) {
                grid[i][j] = line.charAt(j);
                if (grid[i][j] == 'S') {
                    startX = i;
                    startY = j;
                }
            }
        }

        boolean ans = bfs(startX, startY);
        System.out.println(ans ? "YES" : "NO");
    }
```

Python

Plain Text

```
from collections import deque

# 方向定义：上0，下1，左2，右3
dx = [-1, 1, 0, 0]
dy = [0, 0, -1, 1]

def bfs(grid, t, c, n, m, start_x, start_y):
    visited = [[[ [float('inf')] * 2 for _ in range(4)] for _ in range(m)] for _ in range(n)]
    q = deque()

    # 起点四个方向均可，拐弯数0，破墙数0
    for dir in range(4):
        visited[start_x][start_y][dir][0] = 0
        visited[start_x][start_y][dir][1] = 0
        q.append((start_x, start_y, dir, 0, 0))

    while q:
        x, y, dir, used_t, used_c = q.popleft()
        if grid[x][y] == 'T':
            return True

        for ndir in range(4):
            nx = x + dx[ndir]
            ny = y + dy[ndir]
            if not (0 <= nx < n and 0 <= ny < m):
                continue

            nused_t = used_t + (ndir != dir)  # 拐弯
            if nused_t > t:
                continue

            nused_c = used_c + (grid[nx][ny] == '*')  # 破墙
            if nused_c > c:
                continue

             #剪枝逻辑就是你已这个方向到达这个位置，拐弯和碰撞你肯定要有一个优于之前才能进行下去
             #剪枝：新状态如果不优于旧状态，则跳过
            if nused_t > visited[nx][ny][ndir][0] and nused_c >= visited[nx][ny][ndir][1]:
                continue
            if nused_t >= visited[nx][ny][ndir][0] and nused_c > visited[nx][ny][ndir][1]:
```

```
                continue
            if nused_t == visited[nx][ny][ndir][0] and nused_c == visited
[nx][ny][ndir][1]:
                continue
            #上述剪枝可以写成这样的哈，为了把情况给各位分清楚所以分步了
            # if nused_t >= visited[nx][ny][ndir][0] and nused_c >= visite
d[nx][ny][ndir][1]:
            #     continue

            if nused_t < visited[nx][ny][ndir][0]:
                visited[nx][ny][ndir][0] = nused_t
            if nused_c < visited[nx][ny][ndir][1]:
                visited[nx][ny][ndir][1] = nused_c

            q.append((nx, ny, ndir, nused_t, nused_c))
    return False

# 输入处理
t, c = map(int, input().split())
n, m = map(int, input().split())
grid = []
start_x = start_y = -1
for i in range(n):
    row = input()
    grid.append(row)
    for j in range(m):
        if row[j] == 'S':
            start_x, start_y = i, j

print("YES" if bfs(grid, t, c, n, m, start_x, start_y) else "NO")
```

JavaScript

Plain Text

```
const readline = require('readline');
const rl = readline.createInterface({ input: process.stdin });
let input = [];

rl.on('line', line => input.push(line.trim()));
rl.on('close', () => {
    let [t, c] = input[0].split(' ').map(Number);
    let [n, m] = input[1].split(' ').map(Number);
    let grid = input.slice(2, 2 + n).map(line => line.split(''));

    let startX = -1, startY = -1;
    for (let i = 0; i < n; i++) {
        for (let j = 0; j < m; j++) {
            if (grid[i][j] === 'S') {
                startX = i;
                startY = j;
            }
        }
    }

    // 方向定义：上0，下1，左2，右3
    const dx = [-1, 1, 0, 0];
    const dy = [0, 0, -1, 1];

    const visited = Array.from({ length: n * m }, () =>
        Array.from({ length: 4 }, () => [Infinity, Infinity])
    );

    const queue = [];

    for (let dir = 0; dir < 4; dir++) {
        visited[startX * m + startY][dir][0] = 0;
        visited[startX * m + startY][dir][1] = 0;
        queue.push([startX, startY, dir, 0, 0]);
    }

    let found = false;
    while (queue.length) {
        let [x, y, dir, usedT, usedC] = queue.shift();
        if (grid[x][y] === 'T') {
            found = true;
            break;
        }

```

```
45          for (let ndir = 0; ndir < 4; ndir++) {
46              let nx = x + dx[ndir];
47              let ny = y + dy[ndir];
48              if (nx < 0 || nx >= n || ny < 0 || ny >= m) continue;
49
50              let nUsedT = usedT + (ndir !== dir ? 1 : 0); // 拐弯
51              if (nUsedT > t) continue;
52
53              let nUsedC = usedC + (grid[nx][ny] === '*' ? 1 : 0); // 破墙
54              if (nUsedC > c) continue;
55
56              let pos = nx * m + ny;
57              // 剪枝逻辑就是你已这个方向到达这个位置，拐弯和碰撞你肯定要有一个优于之前
    才能进行下去
58              // 剪枝：新状态如果不优于旧状态，则跳过
59              if (nUsedT > visited[pos][ndir][0] && nUsedC >= visited[pos][n
    dir][1]) continue;
60              if (nUsedT >= visited[pos][ndir][0] && nUsedC > visited[pos][n
    dir][1]) continue;
61              if (nUsedT == visited[pos][ndir][0] && nUsedC == visited[pos]
    [ndir][1]) continue;
62              //上述剪枝可以写成这样的哈，为了把情况给各位分清楚所以分步了
63              // if (nUsedT >= visited[pos][ndir][0] && nUsedC >= visited[po
    s][ndir][1]) continue;
64
65              if (nUsedT < visited[pos][ndir][0]) visited[pos][ndir][0] = nU
    sedT;
66              if (nUsedC < visited[pos][ndir][1]) visited[pos][ndir][1] = nU
    sedC;
67
68              queue.push([nx, ny, ndir, nUsedT, nUsedC]);
69          }
70      }
71
72      console.log(found ? 'YES' : 'NO');
73  });
```

Go

Plain Text

```
package main

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

type State struct {
  x, y, dir, usedT, usedC int
}

var dx = []int{-1, 1, 0, 0}
var dy = []int{0, 0, -1, 1}

func bfs(grid [][]byte, t, c, n, m, startX, startY int) bool {
  visited := make([][][][2]int, n)
  for i := range visited {
    visited[i] = make([][][2]int, m)
    for j := range visited[i] {
      visited[i][j] = make([][2]int, 4)
      for d := 0; d < 4; d++ {
        visited[i][j][d] = [2]int{1e9, 1e9} // 初始化为大数
      }
    }
  }

  q := []State{}
  for dir := 0; dir < 4; dir++ {
    visited[startX][startY][dir] = [2]int{0, 0}
    q = append(q, State{startX, startY, dir, 0, 0})
  }

  for len(q) > 0 {
    s := q[0]
    q = q[1:]

    if grid[s.x][s.y] == 'T' {
      return true
    }

    for ndir := 0; ndir < 4; ndir++ {
```

```
45          nx, ny := s.x+dx[ndir], s.y+dy[ndir]
46          if nx < 0 || nx >= n || ny < 0 || ny >= m {
47            continue
48          }
49
50          nUsedT := s.usedT
51          if ndir != s.dir {
52            nUsedT++
53          }
54          if nUsedT > t {
55            continue
56          }
57
58          nUsedC := s.usedC
59          if grid[nx][ny] == '*' {
60            nUsedC++
61          }
62          if nUsedC > c {
63            continue
64          }
65
66                // 剪枝逻辑就是你已这个方向到达这个位置，拐弯和碰撞你肯定要有一个优于之前
    才能进行下去
67                // 剪枝：新状态如果不优于旧状态，则跳过
68          old := visited[nx][ny][ndir]
69          if nUsedT > old[0] && nUsedC >= old[1] {
70            continue
71          }
72          if nUsedT >= old[0] && nUsedC > old[1] {
73            continue
74          }
75          if nUsedT == old[0] && nUsedC == old[1] {
76            continue
77          }
78
79          // 上述剪枝可以写成这样的哈，为了把情况给各位分清楚所以分步了
80          // if nUsedT >= old[0] && nUsedC >= old[1] {
81          //  continue
82          // }
83
84
85          if nUsedT < old[0] {
86            visited[nx][ny][ndir][0] = nUsedT
87          }
88          if nUsedC < old[1] {
89            visited[nx][ny][ndir][1] = nUsedC
90          }
```

```
        q = append(q, State{nx, ny, ndir, nUsedT, nUsedC})
      }
    }
    return false
  }

  func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    parts := strings.Split(scanner.Text(), " ")
    t, _ := strconv.Atoi(parts[0])
    c, _ := strconv.Atoi(parts[1])

    scanner.Scan()
    parts = strings.Split(scanner.Text(), " ")
    n, _ := strconv.Atoi(parts[0])
    m, _ := strconv.Atoi(parts[1])

    grid := make([][]byte, n)
    startX, startY := -1, -1
    for i := 0; i < n; i++ {
      scanner.Scan()
      line := scanner.Text()
      grid[i] = []byte(line)
      for j := 0; j < m; j++ {
        if grid[i][j] == 'S' {
          startX, startY = i, j
        }
      }
    }

    if bfs(grid, t, c, n, m, startX, startY) {
      fmt.Println("YES")
    } else {
      fmt.Println("NO")
    }
  }
```

来自: 华为OD机试 2025B卷 – 上班之路 (C++ & Python & JAVA & JS & GO)_华为od2025b卷-CSDN博客

华为OD机考2025C卷 - 中文分词模拟器 (C++ & Python & JAVA & JS & GO)-CSDN博客

中文分词模拟器

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

> 华为OD机试2025C卷 200分题型

题目描述

给定一个连续不包含空格的字符串，该字符串仅包含英文小写字母及英文标点符号（逗号、分号、句号），同时给定词库，对该字符串进行精确分词。

说明：

1. 精确分词：字符串分词后，不会出现重叠。即"ilovechina"，不同词库可分割为"i,love,china"，“ilove,china”，不能分割出现重叠的"i,ilove,china"，i 出现重叠
2. 标点符号不成词，仅用于断句
3. 词库：根据外部知识库统计出来的常用词汇例：dictionary = [“i”, “love”, “china”, “lovechina”, “ilove”]
4. 分词原则：采用分词顺序优先且最长匹配原则
   - “ilovechina”，假设分词结果 [i,ilove,lo,love,ch,china,lovechina]，则输出 [ilove,china]
   - 错误输出：[i,lovechina]，原因：“ilove” > 优先于“lovechina”成词
   - 错误输出：[i,love,china]，原因：“ilove” > "i"遵循最长匹配原则

输入描述

第一行输入待分词语句“ilovechina”字符串长度限制：0 < length < 256

第二行输入中文词库“i,love,china,ch,na,ve,lo,this,is,this,word”词库长度限制：1 < length < 100000

输出描述

按顺序输出分词结果“i,love,china”

示例1

输入

Plain Text

```
ilovechina
i,love,china,ch,na,ve,lo,this,is,the,word
```

输出

```
i,love,china
```

示例2

输入

```
iat
i,love,china,ch,na,ve,lo,this,is,the,word,beauti,tiful,ful
```

输出

```
i,a,t
```

说明

> 单个字母，不在词库中且不成词则输出单个字母

示例3

输入

```
ilovechina,thewordisbeautiful
i,love,china,ch,na,ve,lo,this,is,the,word,beauti,tiful,ful
```

输出

```
i,love,china the,word,is,beauti,ful
```

说明

> 标点符号为英文标点符号

题解

思路：本题解决需要用到 `前缀树/字典树` 的数据结构来处理词典。

1. 接收词库中的单词，使用 `词库` 单词预构建 `字典树` ，用于处理字符串分词。 `字典树` 也是数据结构中一个比较常见的数据结构，推荐自己去学习学习。
2. 后续利用 `前缀树` 模拟分词，遍历待分词字符串，确定起点之后 `贪心` 尽可能在词典中向后寻找最长匹配的单词进行分词。具体匹配的逻辑可以参照下面的代码逻辑，本质上也是 `字典树` 数据结构的运用。匹配不上的情况则原样输出当前字符即可。
3. 遍历完待分词字符串之后按题目要求输出对应答案即可。

Plain Text

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <algorithm>

using namespace std;

// Trie 结构定义
struct TrieNode {
    bool isWord;
    TrieNode* children[26];

    TrieNode() : isWord(false) {
        fill(begin(children), end(children), nullptr);
    }
};

// 根节点
TrieNode* root = new TrieNode();

// 插入单词
void insertWord(const string& word) {
    TrieNode* currentNode = root;
    for (char c : word) {
        int index = c - 'a';
        if (!currentNode->children[index]) {
            currentNode->children[index] = new TrieNode();
        }
        currentNode = currentNode->children[index];
    }
    currentNode->isWord = true;
}

int main() {
    // 读取输入并转换为小写
    string sentence, dictionary;
    getline(cin, sentence);
    transform(sentence.begin(), sentence.end(), sentence.begin(), ::tolowe
r);
    getline(cin, dictionary);

    // 插入字典中的单词
    stringstream ss(dictionary);
```

```
    string word;
    while (getline(ss, word, ',')) {
        insertWord(word);
    }

    vector<string> result;
    int startIndex = 0;
    int sentenceSize = sentence.size();

    while (startIndex < sentenceSize) {
        // 如果当前字符不是字母，直接加入结果
        if (!isalpha(sentence[startIndex])) {
            result.push_back(string(1, sentence[startIndex++]));
            continue;
        }

        int endIndex = sentenceSize;
        TrieNode* currentNode = root;
        int lastValidIndex = -1;

        // 从 startIndex 开始，寻找最长匹配的单词
        for (int i = startIndex; i < sentenceSize; i++) {
            char c = sentence[i];
            if (!isalpha(c) || !currentNode->children[c - 'a']) {
                break;
            }
            currentNode = currentNode->children[c - 'a'];
            if (currentNode->isWord) {
                lastValidIndex = i;
            }
        }

        // 如果找到了匹配的单词，则使用最长匹配的单词，否则取当前字符
        if (lastValidIndex != -1) {
            result.push_back(sentence.substr(startIndex, lastValidIndex -
startIndex + 1));
            startIndex = lastValidIndex + 1;
        } else {
            result.push_back(string(1, sentence[startIndex++]));
        }
    }

    // 结果输出
    cout << result[0];
    for (size_t i = 1; i < result.size(); i++) {
        cout << "," << result[i];
    }
```

```
90        cout << endl;
91
92        return 0;
```

Java

Plain Text

```
import java.util.*;

public class Main {
    // Trie 结构定义
    static class TrieNode {
        boolean isWord;
        TrieNode[] children = new TrieNode[26];

        TrieNode() {
            Arrays.fill(children, null);
        }
    }

    static TrieNode root = new TrieNode();

    // 插入单词
    static void insertWord(String word) {
        TrieNode currentNode = root;
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            if (currentNode.children[index] == null) {
                currentNode.children[index] = new TrieNode();
            }
            currentNode = currentNode.children[index];
        }
        currentNode.isWord = true;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 读取输入，并转换为小写
        String sentence = scanner.nextLine().toLowerCase();
        String dictionary = scanner.nextLine();
        scanner.close();

        // 插入字典中的单词
        for (String word : dictionary.split(",")) {
            insertWord(word);
        }

        List<String> result = new ArrayList<>();
        int startIndex = 0, sentenceSize = sentence.length();

```

```
        while (startIndex < sentenceSize) {
            // 如果当前字符不是字母，直接加入结果
            if (!Character.isLetter(sentence.charAt(startIndex))) {
                result.add(String.valueOf(sentence.charAt(startIndex++)));
                continue;
            }

            int lastValidIndex = -1;
            TrieNode currentNode = root;

            // 从 startIndex 开始，寻找最长匹配的单词
            for (int i = startIndex; i < sentenceSize; i++) {
                char c = sentence.charAt(i);
                if (!Character.isLetter(c) || currentNode.children[c - 'a'] == null) {
                    break;
                }
                currentNode = currentNode.children[c - 'a'];
                //记录当前最大分割位置
                if (currentNode.isWord) {
                    lastValidIndex = i;
                }
            }
            // 切割
            if (lastValidIndex != -1) {
                result.add(sentence.substring(startIndex, lastValidIndex + 1));
                startIndex = lastValidIndex + 1;
            } else {
                result.add(String.valueOf(sentence.charAt(startIndex++)));
            }
        }

        // 结果输出
        System.out.println(String.join(",", result));
    }
}
```

Python

Plain Text

```
import sys

# Trie 结构定义
class TrieNode:
    def __init__(self):
        self.is_word = False
        self.children = {}

# 根节点
root = TrieNode()

# 插入单词
def insert_word(word):
    current_node = root
    for c in word:
        if c not in current_node.children:
            current_node.children[c] = TrieNode()
        current_node = current_node.children[c]
    current_node.is_word = True

# 读取输入并转换为小写
sentence = sys.stdin.readline().strip().lower()
dictionary = sys.stdin.readline().strip()

# 插入字典中的单词
for word in dictionary.split(','):
    insert_word(word)

result = []
start_index = 0
sentence_size = len(sentence)

while start_index < sentence_size:
    # 如果当前字符不是字母，直接加入结果
    if not sentence[start_index].isalpha():
        result.append(sentence[start_index])
        start_index += 1
        continue

    last_valid_index = -1
    current_node = root

    # 从 start_index 开始，寻找最长匹配的单词
    for i in range(start_index, sentence_size):
```

```
            c = sentence[i]
            if c not in current_node.children:
                break
            current_node = current_node.children[c]
            if current_node.is_word:
                last_valid_index = i

        if last_valid_index != -1:
            result.append(sentence[start_index:last_valid_index + 1])
            start_index = last_valid_index + 1
        else:
            result.append(sentence[start_index])
            start_index += 1

    # 结果输出
    print(",".join(result))
```

JavaScript

Plain Text

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

// Trie 结构定义
class TrieNode {
    constructor() {
        this.isWord = false;
        this.children = {};
    }
}

// 根节点
const root = new TrieNode();

// 插入单词
function insertWord(word) {
    let currentNode = root;
    for (const c of word) {
        if (!(c in currentNode.children)) {
            currentNode.children[c] = new TrieNode();
        }
        currentNode = currentNode.children[c];
    }
    currentNode.isWord = true;
}

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

let inputLines = [];
rl.on('line', (line) => {
    inputLines.push(line);
    if (inputLines.length === 2) {
        rl.close();
    }
});

rl.on('close', () => {
    let sentence = inputLines[0].toLowerCase();
    let dictionary = inputLines[1];

    // 插入字典中的单词
```

```
    dictionary.split(',').forEach(insertWord);

    let result = [];
    let startIndex = 0;
    let sentenceSize = sentence.length;

    while (startIndex < sentenceSize) {
        if (!/[a-z]/.test(sentence[startIndex])) {
            result.push(sentence[startIndex]);
            startIndex++;
            continue;
        }

        let lastValidIndex = -1;
        let currentNode = root;

        for (let i = startIndex; i < sentenceSize; i++) {
            let c = sentence[i];
            if (!(c in currentNode.children)) break;
            currentNode = currentNode.children[c];
            // 记录能够到达的最大切割位置
            if (currentNode.isWord) lastValidIndex = i;
        }

        if (lastValidIndex !== -1) {
            result.push(sentence.slice(startIndex, lastValidIndex + 1));
            startIndex = lastValidIndex + 1;
        } else {
            result.push(sentence[startIndex]);
            startIndex++;
        }
    }

    console.log(result.join(","));
});
```

Go

Plain Text

```
package main

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

// Trie 结构定义
type TrieNode struct {
  isWord    bool
  children  map[rune]*TrieNode
}

// 初始化根节点
var root = &TrieNode{children: make(map[rune]*TrieNode)}

// 插入单词
func insertWord(word string) {
  currentNode := root
  for _, c := range word {
    if _, exists := currentNode.children[c]; !exists {
      currentNode.children[c] = &TrieNode{children: make(map[rune]*TrieNod
e)}
    }
    currentNode = currentNode.children[c]
  }
  currentNode.isWord = true
}

func main() {
  reader := bufio.NewReader(os.Stdin)
  sentence, _ := reader.ReadString('\n')
  sentence = strings.ToLower(strings.TrimSpace(sentence))

  dictionary, _ := reader.ReadString('\n')
  dictionary = strings.TrimSpace(dictionary)

  // 插入字典中的单词
  for _, word := range strings.Split(dictionary, ",") {
    insertWord(word)
  }

```

```
    var result []string
    startIndex := 0
    sentenceSize := len(sentence)

    for startIndex < sentenceSize {
      if sentence[startIndex] < 'a' || sentence[startIndex] > 'z' {
        result = append(result, string(sentence[startIndex]))
        startIndex++
        continue
      }

      lastValidIndex := -1
      currentNode := root

      for i := startIndex; i < sentenceSize; i++ {
        c := rune(sentence[i])
        if _, exists := currentNode.children[c]; !exists {
          break
        }
        currentNode = currentNode.children[c]
              // 当前能够到达的最大位置
        if currentNode.isWord {
          lastValidIndex = i
        }
      }
          // 分词
      if lastValidIndex != -1 {
        result = append(result, sentence[startIndex:lastValidIndex+1])
        startIndex = lastValidIndex + 1
      } else {
        result = append(result, string(sentence[startIndex]))
        startIndex++
      }
    }

    fmt.Println(strings.Join(result, ","))
}
```

> 来自: 华为OD机考2025C卷 – 中文分词模拟器 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机考2025C卷 - 连续出牌数量 (C++ & Python & JAVA & JS & GO)-CSDN博客

连续出牌数量

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

> 华为OD机试2025C卷 200分题型

题目描述

有这么一款单人卡牌游戏，牌面由颜色和数字组成，颜色为红、黄、蓝、绿中的一种，数字为0-9中的一个。游戏开始时玩家从手牌中选取一张卡牌打出，接下来如果玩家手中有和他上一次打出的手牌颜色或者数字相同的手牌，他可以继续将该手牌打出，直至手牌打光或者没有符合条件可以继续打出的手牌。现给定一副手牌，请找到最优的出牌策略，使打出的手牌最多。

输入描述

输入为两行

- 第一行是每张手牌的数字，数字由空格分隔，
- 第二行为对应的每张手牌的颜色，用r y b g这4个字母分别代表4种颜色，字母也由空格分隔。

手牌数量不超过10。

输出描述

输出一个数字，即最多能打出的手牌的数量。

用例1

输入

Plain Text

```
1 4 3 4 5
r y b b r
```

输出

Plain Text

```
3
```

说明

> 如果打（1, r）-> (5, r)，那么能打两张。
> 如果打（4，y）-> (4, b) -> (3, b)，那么能打三张。

用例2

输入

Plain Text

```
1 1 2 3 4
2 r y b l
```

输出

Plain Text

```
1 1
```

说明

> 没有能够连续出牌的组合，只能在开始时打出一张手牌，故输出1

题解

思路：题目可以数据量非常小，直接使用 `递归回溯` 计算不同出牌顺序 可以连续出牌数量的最大值。

1. 定义 `visited` 代表某张牌是否出过，防止重复访问。
2. 在递归回溯中判断一张牌可出有以下情况：
    a. 首次出牌，任意牌都可出。
    b. 该牌与上一次出牌花色相同。
    c. 该牌与上一次出牌牌面值相同。
3. 明白上面两个注意点之后，接下来就是常规的递归回溯处理套路，具体逻辑可参照下面的代码。

C++

Plain Text

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


int res = 0;

// 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;
}

void DFS(vector<string>& card, vector<string>& color, string lastCardValue, string lastCardColor, int count, vector<bool>& visited) {
    res = max(res, count);
    int n = card.size();
    for (int i = 0; i < n; i++) {
        // 重复访问
        if (visited[i]) {
            continue;
        }
        // 第一次出牌 或者 该牌和上次出牌颜色相同 或者 该牌和上次出牌牌面值相同
        if ((lastCardValue == "" && lastCardColor == "") || (color[i] == lastCardColor) || (card[i] == lastCardValue)) {
            visited[i] = true;
            DFS(card, color, card[i], color[i], count + 1, visited);
            visited[i] = false;
        }
```

```
    }
}

int main() {
    string input;
    getline(cin, input);
    vector<string> card = split(input, " ");
    getline(cin, input);
    vector<string> color = split(input, " ");
    int n = card.size();
    vector<bool> visited(n, false);

    DFS(card, color, "", "", 0, visited);
    cout << res;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static int res = 0;

    // 深度优先搜索
    static void DFS(String[] card, String[] color, String lastCardValue, String lastCardColor, int count, boolean[] visited) {
        res = Math.max(res, count);
        int n = card.length;
        for (int i = 0; i < n; i++) {
            if (visited[i]) continue;
            // 第一次出牌 或者 与上张牌颜色或点数相同
            if ((lastCardValue.equals("") && lastCardColor.equals("")) ||
                color[i].equals(lastCardColor) || card[i].equals(lastCardValue)) {
                visited[i] = true;
                DFS(card, color, card[i], color[i], count + 1, visited);
                visited[i] = false;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] card = sc.nextLine().split(" ");
        String[] color = sc.nextLine().split(" ");
        int n = card.length;
        boolean[] visited = new boolean[n];
        DFS(card, color, "", "", 0, visited);
        System.out.println(res);
    }
}
```

Python

```
res = 0

# 深度优先搜索
def dfs(card, color, last_value, last_color, count, visited):
    global res
    res = max(res, count)
    n = len(card)
    for i in range(n):
        if visited[i]:
            continue
        # 第一次出牌 或者 与上一张牌颜色或点数相同
        if (last_value == "" and last_color == "") or color[i] == last_col
or or card[i] == last_value:
            visited[i] = True
            dfs(card, color, card[i], color[i], count + 1, visited)
            visited[i] = False

card = input().split()
color = input().split()
visited = [False] * len(card)
dfs(card, color, "", "", 0, visited)
print(res)
```

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);
    if (inputLines.length === 2) {
        const card = inputLines[0].split(" ");
        const color = inputLines[1].split(" ");
        const visited = Array(card.length).fill(false);
        let res = 0;

        // 深度优先搜索
        function dfs(lastValue, lastColor, count) {
            res = Math.max(res, count);
            for (let i = 0; i < card.length; i++) {
                if (visited[i]) continue;
                // 第一次出牌 或者 与上张牌颜色或点数相同
                if ((lastValue === "" && lastColor === "") || card[i] === lastValue || color[i] === lastColor) {
                    visited[i] = true;
                    dfs(card[i], color[i], count + 1);
                    visited[i] = false;
                }
            }
        }

        dfs("", "", 0);
        console.log(res);
        rl.close();
    }
});
```

Go

Plain Text

```
package main

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

var res int

// 深度优先搜索
func dfs(card, color []string, lastValue, lastColor string, count int, visited []bool) {
  if count > res {
    res = count
  }
  n := len(card)
  for i := 0; i < n; i++ {
    if visited[i] {
      continue
    }
        // 第一次出牌 或者 与上张牌颜色或点数相同
    if (lastValue == "" && lastColor == "") || card[i] == lastValue || color[i] == lastColor {
      visited[i] = true
      dfs(card, color, card[i], color[i], count+1, visited)
      visited[i] = false
    }
  }
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  card := strings.Split(scanner.Text(), " ")
  scanner.Scan()
  color := strings.Split(scanner.Text(), " ")
  visited := make([]bool, len(card))
  dfs(card, color, "", "", 0, visited)
  fmt.Println(res)
}
```

> 来自: 华为OD机考2025C卷 – 连续出牌数量 (C++ & Python & JAVA & JS & GO)-CSDN博客

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

字母组合

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

> 华为OD机试2025C卷 200分题型

题目描述

每个数字关联多个字母，关联关系如下：

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

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

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

给定一个数字字符串和一个屏蔽字符串，输出所有可能的字符组合；

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

输入描述

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

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

输出描述

输出可能的字符串组合

注：字符串之间使用逗号隔开，最后一个字符串后携带逗号

用例1

输入

```Plain Text
78
ux
```

输出

```Plain Text
uw,vw,vx,
```

说明

> ux完全包含屏蔽字符串ux，因此剔除

用例2

输入

```Plain Text
78
x
```

输出

```Plain Text
uw,vw,
```

题解

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

C++

Plain Text

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

vector<string> map = {"abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "wx", "yz"};
unordered_set<char> filterSet;

bool isContainsAllFilterChars(unordered_set<char>& used) {
    for (auto x : filterSet) {
        if (used.find(x) == used.end()) {
            return true;
        }
    }
    return false;
}


void dfs(vector<string>& letters, int index, string path, string& res, string& filter, unordered_set<char>& used) {
    if (index == letters.size()) {
        if (isContainsAllFilterChars(used)) {
            res += path + ",";
        }
        return;
    }

    // 对于每个数字，遍历其对应的字母字符串
    for (int i = 0; i < letters[index].length(); i++) {
        char c = letters[index][i];
        // 递归回溯
        path += c;
        used.insert(c);
        dfs(letters, index + 1, path, res, filter, used);
        path.pop_back();
        used.erase(c);
    }
}

int main() {
    string digits;
    cin >> digits;
    string filter;
```

```
    cin >> filter;

    for (int i = 0; i < filter.size(); i++) {
        filterSet.insert(filter[i]);
    }

    // 根据数字字符串得到每个数字对应的字母字符串
    vector<string> letters(digits.length());
    for (int i = 0; i < digits.length(); i++) {
        letters[i] = map[digits[i] - '0'];
    }

    // 用于存储结果的字符串
    string res = "";
    // 开始进行深度优先搜索
    unordered_set<char> used;
    dfs(letters, 0, "", res, filter, used);

    // 输出结果
    cout << res << endl;

    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    static List<String> map = Arrays.asList("abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "wx", "yz");
    static Set<Character> filterSet = new HashSet<>();

    // 判断是否包含所有过滤字符
    static boolean isContainsAllFilterChars(Set<Character> used) {
        for (char x : filterSet) {
            if (!used.contains(x)) {
                return true;
            }
        }
        return false;
    }

    // 递归回溯
    static void dfs(List<String> letters, int index, StringBuilder path, StringBuilder res, Set<Character> used) {
        if (index == letters.size()) {
            if (isContainsAllFilterChars(used)) {
                res.append(path).append(",");
            }
            return;
        }

        // 遍历当前数字对应的字母
        for (char c : letters.get(index).toCharArray()) {
            path.append(c);
            used.add(c);
            dfs(letters, index + 1, path, res, used);
            path.deleteCharAt(path.length() - 1);
            used.remove(c);
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String digits = scanner.next();
        String filter = scanner.next();
        scanner.close();

        // 记录需要筛选的字符
```

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

Python

```
# 定义映射关系
char_map = ["abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "wx", "yz"]
filter_set = set()

# 判断是否包含所有过滤字符
def is_contains_all_filter_chars(used):
    return any(x not in used for x in filter_set)

# 递归回溯
def dfs(letters, index, path, res, used):
    if index == len(letters):
        if is_contains_all_filter_chars(used):
            res.append("".join(path) + ",")
        return

    # 遍历当前数字对应的字母
    for c in letters[index]:
        path.append(c)
        used.add(c)
        dfs(letters, index + 1, path, res, used)
        path.pop()
        used.remove(c)

# 处理输入
digits = input().strip()
filter_chars = input().strip()

filter_set = set(filter_chars)
letters = [char_map[int(d)] for d in digits]

# 开始递归
res = []
dfs(letters, 0, [], res, set())

# 输出结果
print("".join(res))
```

JavaScript

Plain Text

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

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

const charMap = ["abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "wx", "yz"];
let filterSet = new Set();

// 判断是否包含全部单词
function isContainsAllFilterChars(used) {
    for (let x of filterSet) {
        if (!used.has(x)) {
            return true;
        }
    }
    return false;
}

function dfs(letters, index, path, res, used) {
    if (index === letters.length) {
        if (isContainsAllFilterChars(used)) {
            res.push(path.join("") + ",");
        }
        return;
    }

    // 遍历当前数字对应的字母
    for (let c of letters[index]) {
        path.push(c);
        used.add(c);
        dfs(letters, index + 1, path, res, used);
        path.pop();
        used.delete(c);
    }
}

rl.question("", (digits) => {
    rl.question("", (filter) => {
        filterSet = new Set(filter);
        let letters = digits.split("").map(d => charMap[parseInt(d)]);

```

```
        let res = [];
        dfs(letters, 0, [], res, new Set());

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

Go

Plain Text

```
package main

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

var charMap = []string{"abc", "def", "ghi", "jkl", "mno", "pqr", "st", "uv", "wx", "yz"}
var filterSet map[rune]struct{}

// 判断是否包含所有过滤字符
func isContainsAllFilterChars(used map[rune]struct{}) bool {
  for x := range filterSet {
    if _, exists := used[x]; !exists {
      return true
    }
  }
  return false
}

// 递归回溯
func dfs(letters []string, index int, path []rune, res *[]string, used map[rune]struct{}) {
  if index == len(letters) {
    if isContainsAllFilterChars(used) {
      *res = append(*res, string(path)+",")
    }
    return
  }

  // 遍历当前数字对应的字母
  for _, c := range letters[index] {
    path = append(path, c)
    used[c] = struct{}{}
    dfs(letters, index+1, path, res, used)
    path = path[:len(path)-1]
    delete(used, c)
  }
}

func main() {
```

```
    scanner := bufio.NewScanner(os.Stdin)

    // 读取数字字符串
    scanner.Scan()
    digits := scanner.Text()

    // 读取过滤字符
    scanner.Scan()
    filter := scanner.Text()

    // 记录需要筛选的字符
    filterSet = make(map[rune]struct{})
    for _, c := range filter {
      filterSet[c] = struct{}{}
    }

    // 获取对应的字母映射
    letters := make([]string, len(digits))
    for i, c := range digits {
      letters[i] = charMap[c-'0']
    }

    // 结果存储
    var res []string
    dfs(letters, 0, []rune{}, &res, make(map[rune]struct{}))

    // 输出结果
    fmt.Println(strings.Join(res, ""))
}
```

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