od0612

华为OD 上机考试 - 磁盘容量排序 (2025 B卷 100分)_od上机考试(2025年b卷)-CSDN博客

华为OD 机试 - 最长连续子序列 (机考 2025 B卷 100分)-CSDN博客

华为od 上机考试 - 堆栈中的剩余数字 (2025 B卷 200分)-CSDN博客

华为OD 机试 - 篮球游戏 / 队列 (机考 2025 B卷 200分)-CSDN博客

华为OD 机考 - 分苹果 (2025 B卷 100分)-CSDN博客

华为od 机试 - 停车费用统计 (2025 B卷 100分)-CSDN博客

华为OD 机试 - 字符串解密 (2025 B卷 100分)_华为od机考-CSDN博客

华为OD机试 - 构建数列 (2025 B卷 100分)_题目描述构造数列,第一个数为n,后面的数不大于前一个的一半,数列奇偶相间或许全为-CSDN博客

华为OD 机考 - 最少交付时间 (2025 B卷 100分)_华为机考题2025-CSDN博客

华为OD 机考 - AI面板识别 (2025 B卷 100分)-CSDN博客

华为OD 上机考试 - 磁盘容量排序 (2025 B卷 100分)_od上机考试(2025年b卷)-CSDN博客

磁盘容量排序

华为OD机试真题目录: 点击去查看

> 2025 B卷 100分题型

题目描述

磁盘的容量单位常用的有 M，G，T 这三个等级，它们之间的换算关系为：

- 1T = 1024G
- 1G = 1024M

现在给定 n 块磁盘的容量，请对它们按从小到大的顺序进行[稳定排序]。
例如给定5块盘的容量：

> 1T，20M，3G，10G6T，3M12G9M

排序后的结果为：

> 20M，3G，3M12G9M，1T，10G6T

注意单位可以重复出现，上述 3M12G9M 表示的容量即为：3M+12G+9M，和 12M12G 相等。

输入描述

输入第一行包含一个整数 n，表示磁盘的个数

- 2 ⩽ n ⩽ 100

接下的 n 行，每行一个[字符串]（长度大于2，小于30），表示磁盘的容量，由一个或多个格式为mv的子串组成，其中 m 表示容量大小，v 表示容量单位，例如：20M，1T，30G，10G6T，3M12G9M。

- 磁盘容量 m 的范围为 1 到 1024 的正整数
- 容量单位 v 的范围只包含题目中提到的 M，G，T 三种，换算关系如题目描述

输出描述

输出 n 行，表示 n 块磁盘容量排序后的结果。

示例1

输入

```
3
1G
2G
1024M
```

输出

```
1G
1024M
2G
```

说明

> 1G和1024M容量相等，稳定排序要求保留它们原来的相对位置，故1G在1024M之前。

示例2

输入

```
3
2G4M
3M2G
1T
```

输出

```
3M2G
2G4M
1T
```

说明

> 1T的容量大于2G4M，2G4M的容量大于3M2G。

题解

思路： `自定义排序`

1. 将每个输入的磁盘容量字符串统一转换为以 `M` 为单位的int数字，方便排序进行比较。
2. 进行二分排序(也可以使用自带的排序，但一定要选择 稳定排序的API)，书写实现时需要额外注意 `排序稳定` 的要求。(稳定排序的含义：如果数组中两个值相等，排序之后的前后相互关系要和排序前保持一致)
3. 经过2进行稳定排序之后，按顺序输入排序之后的磁盘容量字符串。

> 这道题主要考察稳定排序，是在不行的话，也可以定义一个多条件排序，优先按照磁盘大小排序，磁盘大小相同情况下按照输入顺序排序。

C++

Plain Text

```
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<algorithm>
using namespace std;
//将字符串的值转换为Longlong类型
long long calVaue(string s) {
    long long res = 0;
    int sum = 0;
    for (int i = 0; i < s.size(); i++) {
        if (s[i] >= '0' && s[i] <= '9') {
            sum = sum * 10 + s[i] - '0';
        } else {
            if (s[i] == 'M') {
                res += sum;
            } else if (s[i] == 'G'){
                res += (1024 * sum);
            } else {
                res += (1024 * sum * 1024);
            }
            sum = 0;
        }
    }
    return res;
}



// 快速排序实现
void quickSort(int left, int right, vector<string> & ans) {
    if (left >= right) {
        return;
    }
    int begin = left;
    int end = right;
    string pos = ans[left];
    while (left < right) {
        // = 条件视为稳定而特别设置
        while (left < right && calVaue(pos) <= calVaue(ans[right])) {
            right--;
        }
        if (left < right) {
            swap(ans[left], ans[right]);
```

```
            left++;
        }
        // 不是>=也是为了排序稳定
        while (left < right && calVaue(pos) > calVaue(ans[left])) {
            left++;
        }
        if (left < right) {
            swap(ans[left], ans[right]);
            right--;
        }
    }
    ans[left] = pos;
    quickSort(begin, left - 1, ans);
    quickSort(left + 1, end ,ans);
}


int main() {
    int n;
    cin >> n;
    vector<string> ans(n);
    for (int i = 0; i < n; i++) {
        cin >> ans[i];
    }
    quickSort(0, ans.size()-1 ,ans);
    for (int i = 0; i < n; i++) {
        cout<<ans[i] <<endl;
    }
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 将字符串的值转换为Longlong类型
    public static long calValue(String s) {
        long res = 0;
        int sum = 0;
        for (int i = 0; i < s.length(); i++) {
            if (Character.isDigit(s.charAt(i))) {
                sum = sum * 10 + s.charAt(i) - '0';
            } else {
                if (s.charAt(i) == 'M') {
                    res += sum;
                } else if (s.charAt(i) == 'G') {
                    res += (1024 * sum);
                } else {
                    res += (1024 * sum * 1024);
                }
                sum = 0;
            }
        }
        return res;
    }

    // 快速排序实现
    public static void quickSort(int left, int right, List<String> ans) {
        if (left >= right) {
            return;
        }
        int begin = left;
        int end = right;
        String pos = ans.get(left);
        while (left < right) {

            while (left < right && calValue(pos) <= calValue(ans.get(righ
t))) {
                right--;
            }
            if (left < right) {
                Collections.swap(ans, left, right);
                left++;
            }
            while (left < right && calValue(pos) > calValue(ans.get(lef
t))) {
                left++;
```

```
            }
            if (left < right) {
                Collections.swap(ans, left, right);
                right--;
            }
        }
        ans.set(left, pos);
        quickSort(begin, left - 1, ans);
        quickSort(left + 1, end, ans);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        scanner.nextLine(); // consume newline
        List<String> ans = new ArrayList<>(n);
        for (int i = 0; i < n; i++) {
            ans.add(scanner.nextLine());
        }
        quickSort(0, ans.size() - 1, ans);
        for (String s : ans) {
            System.out.println(s);
        }
    }
}
```

Python

Plain Text

```
# 统一单位
def cal_value(s):
    res = 0
    sum_val = 0
    for ch in s:
        if ch.isdigit():
            sum_val = sum_val * 10 + int(ch)
        else:
            if ch == 'M':
                res += sum_val
            elif ch == 'G':
                res += 1024 * sum_val
            else:
                res += 1024 * sum_val * 1024
            sum_val = 0
    return res

def quick_sort(left, right, ans):
    if left >= right:
        return
    begin = left
    end = right
    pos = ans[left]
    while left < right:
        # 稳定排序
        while left < right and cal_value(pos) <= cal_value(ans[right]):
            right -= 1
        if left < right:
            ans[left], ans[right] = ans[right], ans[left]
            left += 1
        #稳定排序
        while left < right and cal_value(pos) > cal_value(ans[left]):
            left += 1
        if left < right:
            ans[left], ans[right] = ans[right], ans[left]
            right -= 1
    ans[left] = pos
    quick_sort(begin, left - 1, ans)
    quick_sort(left + 1, end, ans)

if __name__ == "__main__":
    n = int(input())
    ans = [input().strip() for _ in range(n)]
    quick_sort(0, len(ans) - 1, ans)
    for s in ans:
```

```
46            print(s)
```

JavaScript

Plain Text

```
// 统一单位
function calValue(s) {
    let res = 0;
    let sum = 0;
    for (let i = 0; i < s.length; i++) {
        if (s[i] >= '0' && s[i] <= '9') {
            sum = sum * 10 + (s.charCodeAt(i) - '0'.charCodeAt(0));
        } else {
            if (s[i] === 'M') {
                res += sum;
            } else if (s[i] === 'G') {
                res += (1024 * sum);
            } else {
                res += (1024 * sum * 1024);
            }
            sum = 0;
        }
    }
    return res;
}

function quickSort(left, right, ans) {
    if (left >= right) {
        return;
    }
    let begin = left;
    let end = right;
    let pos = ans[left];
    while (left < right) {
        // = 为了排序稳定
        while (left < right && calValue(pos) <= calValue(ans[right])) {
            right--;
        }
        if (left < right) {
            [ans[left], ans[right]] = [ans[right], ans[left]];
            left++;
        }
        // = 为了排序稳定
        while (left < right && calValue(pos) > calValue(ans[left])) {
            left++;
        }
        if (left < right) {
            [ans[left], ans[right]] = [ans[right], ans[left]];
            right--;
        }
```

```
    }
    ans[left] = pos;
    quickSort(begin, left - 1, ans);
    quickSort(left + 1, end, ans);
}

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 === 1 + parseInt(inputLines[0])) {
        let n = parseInt(inputLines[0]);
        let ans = inputLines.slice(1);
        quickSort(0, ans.length - 1, ans);
        ans.forEach(s => console.log(s));
        rl.close();
    }
});
```

Go

Plain Text

```
package main

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

)

// 将字符串的值转换为Longlong类型
func calValue(s string) int64 {
  var res int64 = 0
  sum := 0
  for i := 0; i < len(s); i++ {
    if s[i] >= '0' && s[i] <= '9' {
      sum = sum*10 + int(s[i]-'0')
    } else {
      if s[i] == 'M' {
        res += int64(sum)
      } else if s[i] == 'G' {
        res += int64(1024 * sum)
      } else {
        res += int64(1024 * sum * 1024)
      }
      sum = 0
    }
  }
  return res
}

// 快速排序实现
func quickSort(left, right int, ans []string) {
  if left >= right {
    return
  }
  begin := left
  end := right
  pos := ans[left]
  for left < right {
    for left < right && calValue(pos) <= calValue(ans[right]) {
      right--
    }
    if left < right {
      ans[left], ans[right] = ans[right], ans[left]
```

```
      left++
    }
    for left < right && calValue(pos) > calValue(ans[left]) {
      left++
    }
    if left < right {
      ans[left], ans[right] = ans[right], ans[left]
      right--
    }
  }
  ans[left] = pos
  quickSort(begin, left-1, ans)
  quickSort(left+1, end, ans)
}

func main() {
  scanner := bufio.NewScanner(os.Stdin)
  scanner.Scan()
  n, _ := strconv.Atoi(scanner.Text())
  ans := make([]string, n)
  for i := 0; i < n; i++ {
    scanner.Scan()
    ans[i] = scanner.Text()
  }
  quickSort(0, len(ans)-1, ans)
  for _, s := range ans {
    fmt.Println(s)
  }
}
```

> 来自: 华为OD 上机考试 – 磁盘容量排序 (2025 B卷 100分)_od上机考试(2025年b卷)-CSDN博客

华为OD 机试 - 最长连续子序列 (机考 2025 B卷 100分)-CSDN博客

最长连续子序列

华为OD机试真题目录: 点击去查看

> `2025 B卷 100分题型`

题目描述

有N个正整数组成的一个序列。给定整数sum，求长度最长的连续子序列，使他们的和等于sum，返回此子序列的长度，
如果没有满足要求的序列，返回-1。

输入描述

第一行输入是：N个正整数组成的一个序列
第二行输入是：给定整数sum

输出描述

最长的连续子序列的长度
备注

- 输入序列仅由数字和英文逗号构成，数字之间采用英文逗号分隔
- 序列长度：1 <= N <= 200
- 输入序列不考虑异常情况

示例1

输入

Plain Text

```
1,2,3,4,2
6
```

输出

```
3
```

说明

> 1,2,3和4,2两个序列均能满足要求，所以最长的连续序列为1,2,3，因此结果为3。

示例2

输入

```
1,2,3,4,2
20
```

输出

```
-1
```

说明

> 没有满足要求的子序列，返回-1

题解

思路：`双指针求解`，定义`left 和 right两个指针`，定义`res` = -1存储`sum(ans[left : right]) == sum`的最大长度。双指针移动逻辑如下：

- 当sum(ans[left:right]) < sum时，right右移
- 当sum(ans[left:right]) > sum时，left右移
- 当sum(ans[left:right]) = sum时，更新结果`res = max(res, right-left+1)`

> 双指针移动终止条件为`right >= len(nums)`，此时直接输出`res`就是为最终结果。

C++

Plain Text

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

int main() {
    string line1, line2;
    getline(cin, line1);  // 读取序列输入
    getline(cin, line2);  // 读取目标值输入

    // 解析输入的序列
    vector<int> nums;
    stringstream ss(line1);
    string token;
    while (getline(ss, token, ',')) {
        nums.push_back(stoi(token));
    }

    int target = stoi(line2);
    int n = nums.size();
    int left = 0, right = 0, sum = 0, maxLen = -1;

    while (right < n) {
        // 不断扩大窗口，增加右边界
        sum += nums[right];
        right++;

        // 如果当前窗口内的和大于目标值，收缩左边界
        while (sum > target && left < right) {
            sum -= nums[left];
            left++;
        }

        // 检查是否等于目标值，并更新最大长度
        if (sum == target) {
            maxLen = max(maxLen, right - left);
        }
    }

    // 输出结果
    cout << maxLen << endl;
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

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

        // 读取序列输入
        String line1 = sc.nextLine();
        // 读取目标值输入
        String line2 = sc.nextLine();

        // 解析输入的序列
        String[] tokens = line1.split(",");
        int[] nums = new int[tokens.length];
        for (int i = 0; i < tokens.length; i++) {
            nums[i] = Integer.parseInt(tokens[i]);
        }

        int target = Integer.parseInt(line2);
        int n = nums.length;
        int left = 0, right = 0, sum = 0, maxLen = -1;

        while (right < n) {
            // 不断扩大窗口，增加右边界
            sum += nums[right];
            right++;

            // 如果当前窗口内的和大于目标值，收缩左边界
            while (sum > target && left < right) {
                sum -= nums[left];
                left++;
            }

            // 检查是否等于目标值，并更新最大长度
            if (sum == target) {
                maxLen = Math.max(maxLen, right - left);
            }
        }

        // 输出结果
        System.out.println(maxLen);
        sc.close();
    }
}
```

Python

```
def main():
    # 读取序列输入
    line1 = input().strip()
    # 读取目标值输入
    line2 = input().strip()

    # 解析输入的序列
    nums = list(map(int, line1.split(',')))
    target = int(line2)
    n = len(nums)
    left = 0
    right = 0
    sum = 0
    max_len = -1

    while right < n:
        # 不断扩大窗口，增加右边界
        sum += nums[right]
        right += 1

        # 如果当前窗口内的和大于目标值，收缩左边界
        while sum > target and left < right:
            sum -= nums[left]
            left += 1

        # 检查是否等于目标值，并更新最大长度
        if sum == target:
            max_len = max(max_len, right - left)

    # 输出结果
    print(max_len)

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

JavaScript

Plain Text

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

rl.question('', (line1) => {
  rl.question('', (line2) => {
    // 解析输入的序列
    const nums = line1.split(',').map(Number);
    const target = parseInt(line2);
    let n = nums.length;
    let left = 0;
    let right = 0;
    let sum = 0;
    let maxLen = -1;

    while (right < n) {
      // 不断扩大窗口，增加右边界
      sum += nums[right];
      right++;

      // 如果当前窗口内的和大于目标值，收缩左边界
      while (sum > target && left < right) {
        sum -= nums[left];
        left++;
      }

      // 检查是否等于目标值，并更新最大长度
      if (sum === target) {
        maxLen = Math.max(maxLen, right - left);
      }
    }

    // 输出结果
    console.log(maxLen);
    rl.close();
  });
});
```

Go

Plain Text

```
package main

import (
  "fmt"
  "strings"
  "strconv"
)

func main() {
  // 读取序列输入
  var line1 string
  fmt.Scanln(&line1)
  // 读取目标值输入
  var line2 string
  fmt.Scanln(&line2)

  // 解析输入的序列
  tokens := strings.Split(line1, ",")
  nums := make([]int, len(tokens))
  for i, token := range tokens {
    nums[i], _ = strconv.Atoi(token)
  }

  target, _ := strconv.Atoi(line2)
  n := len(nums)
  left, right, sum, maxLen := 0, 0, 0, -1

  for right < n {
    // 不断扩大窗口，增加右边界
    sum += nums[right]
    right++

    // 如果当前窗口内的和大于目标值，收缩左边界
    for sum > target && left < right {
      sum -= nums[left]
      left++
    }

    // 检查是否等于目标值，并更新最大长度
    if sum == target {
      maxLen = max(maxLen, right-left)
    }
  }

  // 输出结果
```

```
  fmt.Println(maxLen)
}

// max 函数用于返回较大的数值
func max(a, b int) int {
  if a > b {
    return a
  }
  return b
}
```

> 来自: 华为OD 机试 - 最长连续子序列 (机考 2025 B卷 100分)-CSDN博客

华为od 上机考试 - 堆栈中的剩余数字 (2025 B卷 200分)-CSDN博客

堆栈中的剩余数字

华为OD机试真题目录: 点击去查看

> 2025 B卷 200分题型

题目描述

向一个空栈中依次存入正整数，假设入栈元素 n(1<=n<=2^31-1)按顺序依次为 nx...n4、 n3、n2、 n1，每当元素入栈时，如果 n1=n2+...+ny(y 的范围[2,x]， 1<=x<=1000)，则 n1~ny 全部元素出栈，重新入栈新元素 m(m=2*n1)。
如：依次向栈存入 6、 1、 2、 3，当存入 6、 1、 2 时，栈底至栈顶依次为[6、 1、 2]；当存入 3时，3=2+1， 3、 2、 1 全部出栈，重新入栈元素 6(6=2*3)，此时栈中有元素 6；
因为 6=6，所以两个 6 全部出栈，存入 12，最终栈中只剩一个元素 12。

输入描述

使用单个空格隔开的正整数的字符串，如”5 6 7 8”， 左边的数字先入栈，输入的正整数个数为 x，1<=x<=1000。

输出描述

最终栈中存留的元素值，元素值使用空格隔开，如”8 7 6 5”， 栈顶数字在左边。 6 1 2 3

用例1

输入

```
5 10 20 50 85 1
```

输出

```
1 170
```

说明

> 5+10+20+50=85， 输入 85 时， 5、 10、 20、 50、 85 全部出栈，入栈 170，最终依次出栈的数字为 1 和 170。

用例2

输入

```
6 7 8 13 9
```

输出

```
9 13 8 7 6
```

用例3

输入

```
1 2 5 7 9 1 2 2
```

输出

```
4 1 9 14 1
```

题解

思路： 模拟

1. 按照题目使用数组进行模拟就行。当一个元素 x 被压入栈的时候，判断是否存在连续后缀和等于当前元素，如果存在的话弹出连续后缀，压入 `2 *x + 1` .若不存在压入 `x` 即可。
2. 这道题主要考点就是 `1` 弹出连续后缀的操作，需要递归进行判断，例如压入 `2 * x + 1` 后，又存在一个新的连续后缀和等于 `2 * x + 1` .
3. 遍历完所有元素之后，输出数组中保留的元素即可。

C++

Plain Text

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

int main() {
    vector<int> ans;
    int tmp;
    while (cin >> tmp) {
        ans.push_back(tmp);
    }

    int n = ans.size();
    vector<int> stk(n,0);
    // 下一个要放入的位置
    int pos = 0;
    for (int i = 0; i < n; i++) {
        long sum = ans[i];
        long tmp = sum;

        // 递归进行压栈判断
        while (pos > 0 && tmp >=0) {
            int j = pos - 1;
            for (; j >=0; j--) {
                tmp -= stk[j];
                if (tmp <= 0) {
                    break;
                }
            }
            // 满足条件，更新元素
            if (tmp == 0) {
                sum = 2 * sum;
                pos = j;
                tmp = sum;
            } else {
                break;
            }
        }

        stk[pos++] = sum;
    }
```

```
46
47      for (int i = pos -1; i >=0;i--) {
48          cout << stk[i] << " ";
49      }
50      return 0;
51  }
```

JAVA

Plain Text

```
import java.util.*;

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

        // 读取输入直到结束
        while (sc.hasNextInt()) {
            ans.add(sc.nextInt());
        }

        int n = ans.size();

        int[] stk = new int[n];
        int pos = 0;

        for (int i = 0; i < n; i++) {
            long sum = ans.get(i);
            long tmp = sum;

            // 递归进行压栈判断
            while (pos > 0 && tmp >= 0) {
                int j = pos - 1;
                for (; j >= 0; j--) {
                    tmp -= stk[j];
                    if (tmp <= 0) {
                        break;
                    }
                }
                // 满足条件，更新元素
                if (tmp == 0) {
                    sum = 2 * sum;
                    pos = j;
                    tmp = sum;
                } else {
                    break;
                }
            }

            stk[pos++] = (int) sum;
        }

        // 输出栈中的元素
        for (int i = pos - 1; i >= 0; i--) {
```

```
                System.out.print(stk[i] + " ");
            }
        }
    }
```

Python

Plain Text

```
def main():
    ans = []

    # 读取输入数据，直到没有更多输入
    try:
        while True:
            # 读取一整行输入
            line = input().strip()
            # 拆分输入的字符串并将每个数字转换为整数
            ans.extend(map(int, line.split()))
    except EOFError:
        pass

    n = len(ans)
    stk = [0] * n
    pos = 0

    for i in range(n):
        sum_val = ans[i]
        tmp = sum_val

        # 递归进行压栈判断
        while pos > 0 and tmp >= 0:
            j = pos - 1
            while j >= 0:
                tmp -= stk[j]
                if tmp <= 0:
                    break
                j -= 1

            # 满足条件，更新元素
            if tmp == 0:
                sum_val = 2 * sum_val
                pos = j
                tmp = sum_val
            else:
                break

        stk[pos] = sum_val
        pos += 1

    # 输出栈中的元素
    for i in range(pos - 1, -1, -1):
        print(stk[i], end=" ")

```

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

JavaScript

Plain Text

```
/* JavaScript Node ACM模式 控制台输入获取 */
const readline = require('readline');

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

function main() {
    let ans = [];

    // 读取输入数据
    rl.on('line', (input) => {
        if (input.trim() === '') {
            // 结束输入
            rl.close();
        } else {
            // 将输入的字符串按空格分割，并转换为整数
            ans.push(...input.split(' ').map(Number));
        }
    });

    rl.on('close', () => {
        let n = ans.length;
        let stk = new Array(n).fill(0);
        let pos = 0;

        for (let i = 0; i < n; i++) {
            let sum = ans[i];
            let tmp = sum;

            // 递归进行压栈判断
            while (pos > 0 && tmp >= 0) {
                let j = pos - 1;
                while (j >= 0) {
                    tmp -= stk[j];
                    if (tmp <= 0) {
                        break;
                    }
                    j--;
                }

                // 满足条件，更新元素
                if (tmp === 0) {
                    sum = 2 * sum;
```

```
                    pos = j;
                    tmp = sum;
                } else {
                    break;
                }
            }

            stk[pos] = sum;
            pos++;
        }

        // 输出栈中的元素
         console.log(stk.slice(0, pos).reverse().join(' '));
    });
}

main();
```

Go

Plain Text

```
package main

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

func main() {
  // 从标准输入读取数据
  scanner := bufio.NewScanner(os.Stdin)

  var ans []int
  if scanner.Scan() {
    line := scanner.Text()
    inputs := strings.Fields(line)
    for _, val := range inputs {
      num, err := strconv.Atoi(val)
      if err == nil {
        ans = append(ans, num)
      }
    }
  }

  n := len(ans)
  stk := make([]int, n)
  pos := 0 // 下一个要放入的位置

  for i := 0; i < n; i++ {
    sum := ans[i]
    tmp := sum

    // 递归进行压栈判断
    for pos > 0 && tmp >= 0 {
      j := pos - 1
      for j >= 0 {
        tmp -= stk[j]
        if tmp <= 0 {
          break
        }
        j--
      }

```

```
        // 满足条件，更新元素
        if tmp == 0 {
          sum = 2 * sum
          pos = j
          tmp = sum
        } else {
          break
        }
      }

      stk[pos] = sum
      pos++
    }

    // 输出结果
    for i := pos - 1; i >= 0; i-- {
      fmt.Print(stk[i], " ")
    }
  }
```

> 来自: 华为od 上机考试 - 堆栈中的剩余数字 (2025 B卷 200分)-CSDN博客

华为OD 机试 - 篮球游戏 / 队列 (机考 2025 B卷 200分)-CSDN博客

篮球游戏 / 队列

华为OD机试真题目录: 点击去查看

> 2025B卷 200分题型

题目描述

幼儿园里有一个放倒的圆桶，它是一个线性结构，允许在桶的右边将篮球放入，可以在桶的左边和右边将篮球取出。
每个篮球有单独的编号，老师可以连续放入一个或多个篮球，小朋友可以在桶左边或右边将篮球取出，当桶只有一个篮球的情况下，必须从左边取出。
如老师按顺序放入1、2、3、4、5 共有 5 个编号的篮球，那么小朋友可以依次取出编号为1、2、3、4、5 或者 3、1、2、4、5 编号的篮球，无法取出 5、1、3、2、4 编号的篮球。
其中 3、1、2、4、5 的取出场景为：

- 连续放入1、2、3号
- 从右边取出3号
- 从左边取出1号
- 从左边取出2号
- 放入4号
- 从左边取出4号
- 放入5号
- 从左边取出5号

简答起见，我们以 L 表示左，R表示右，此时取出篮球的依次取出序列为“RLLLL”。

输入描述

每次输入包含一个测试用例：

1. 第一行的数字作为老师依次放入的篮球编号
2. 第二行的数字作为要检查是否能够按照放入的顺序取出给定的篮球的编号，其中篮球的编号用逗号进行分隔。

其中篮球编号用逗号进行分隔。

输出描述

对于每个篮球的取出序列，如果确实可以获取，请打印出其按照左右方向的操作取出顺序，如果无法获取则打印“NO”。

备注

- 1 ⩽ 篮球编号，篮球个数 ⩽ 200
- 篮球上的数字不重复
- 输出的结果中 LR 必须为大写

用例1

输入

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

输出

```Plain Text
RLRRRLL
```

说明

> 篮球的取出顺序依次为“右、左、右、右、右、左、左”

用例2

输入

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

输出

```Plain Text
NO
```

说明

> 无法取出对应序列的篮球

用例3

输入

```
1,2,3,4
1,2,3,5
```

输出

```
NO
```

说明

> 不存在编号为5的篮球，所以无法取出对应编号的数据

题解

思路： `模拟` ， 处理的代码逻辑如下：

1. 接收篮球的放入序列以及要求取出的序列，定义 `current = 0` 指向取出序列第一个位置。
2. 按照顺序放入指定篮球(队尾)，当发现output[current]被放入时，如果output[current]位于队首(L)和队尾®弹出对应篮球，current++。 如果此时不位于这两个位置，则说明指定取出序列是非法序列，终止迭代，直接输出NO。
3. 迭代2的步骤，直到将所有篮球全部放入和取出，记录其中的每个 `output[current]` 取出位置拼接而成就是答案。
4. 额外注意： 当桶只有一个篮球的情况下，必须从左边取出。

C++

Plain Text

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

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

int main() {
   string s1, s2;
   cin >> s1 >> s2;
   vector<string> input = split(s1, ",");
   vector<string> output = split(s2, ",");
   int n = input.size();
   vector<string> res;
   // 列表模拟队列
   list<string> lst;
   set<string> se;
   int pos = 0;

   for (int i = 0; i < input.size(); i++)  {
       lst.push_back(input[i]);
       se.insert(input[i]);
       // 迭代取出
       while (pos < n && se.find(output[pos]) != se.end()) {
      // 只有一个或者在最左边
           if (lst.size() == 1 || lst.front() == output[pos]) {
                res.push_back("L");
```

```
                lst.pop_front();
            } else if (lst.back() == output[pos]){
                res.push_back("R");
                lst.pop_back();
            }else {
              break;
            }
            pos++;
        }
     }
     //无法正确取出
    if (res.size() != input.size()) {
       cout << "NO";
     } else {
        for (int i = 0; i < res.size(); i++) {
            cout << res[i];
        }
     }
     return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 通用 split 函数
    private static List<String> split(String str, String delimiter) {
        return Arrays.asList(str.split(delimiter));
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String s1 = scanner.next();
        String s2 = scanner.next();
        scanner.close();

        List<String> input = split(s1, ",");
        List<String> output = split(s2, ",");
        int n = input.size();
        List<String> res = new ArrayList<>();

        // 列表模拟队列
        LinkedList<String> lst = new LinkedList<>();
        Set<String> se = new HashSet<>();
        int pos = 0;

        for (String item : input) {
            lst.add(item);
            se.add(item);

            // 迭代取出
            while (pos < n && se.contains(output.get(pos))) {
                // 只有一个或者在最左边
                if (lst.size() == 1 || lst.getFirst().equals(output.get(po
s))) {
                    res.add("L");
                    lst.removeFirst();
                } else if (lst.getLast().equals(output.get(pos))) {
                    res.add("R");
                    lst.removeLast();
                } else {
                    break;
                }
                pos++;
            }
        }

```

```
            // 无法正确取出
            if (res.size() != input.size()) {
                System.out.println("NO");
            } else {
                for (String op : res) {
                    System.out.print(op);
                }
            }
        }
    }
```

Python

Plain Text

```
import sys



def main():
    s1 = sys.stdin.readline().strip()
    s2 = sys.stdin.readline().strip()

    input_list = s1.split(",")
    output_list = s2.split(",")
    n = len(input_list)
    res = []

    # 列表模拟队列
    lst = []
    se = set()
    pos = 0

    for item in input_list:
        lst.append(item)
        se.add(item)

        # 迭代取出
        while pos < n and output_list[pos] in se:
            # 只有一个或者在最左边
            if len(lst) == 1 or lst[0] == output_list[pos]:
                res.append("L")
                lst.pop(0)
            elif lst[-1] == output_list[pos]:
                res.append("R")
                lst.pop()
            else:
                break
            pos += 1

    # 无法正确取出
    if len(res) != len(input_list):
        print("NO")
    else:
        print("".join(res))

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

```
const rl = require("readline").createInterface({ input: process.stdin });
var iter = rl[Symbol.asyncIterator]();
const readline = async () => (await iter.next()).value;

void (async function () {
  const inputs = (await readline()).split(",").map(Number);
  const outputs = (await readline()).split(",").map(Number);

  // 利用队列结构模拟圆桶
  const queue = [];
  // outputs[index]是要被取出的篮球的编号
  let index = 0;

  // 记录题解
  const res = [];

  for (let input of inputs) {
    // 按照放入顺序，从圆桶右边放入
    queue.push(input);

    // 然后开始尝试取出
    while (queue.length > 0) {
      // 圆桶左边的篮球的编号
      const left = queue[0];
      // 圆桶右边的篮球的编号
      const right = queue.at(-1);

      if (left == outputs[index]) {
        // 优先比较圆桶左边的篮球是不是当前要取出的篮球，优先左边的原因是：当桶只有一个篮球的情况下，必须从左边取出
        res.push("L");
        queue.shift();
        index++;
      } else if (right == outputs[index]) {
        // 比较圆桶右边的篮球是不是当前要取出的篮球
        res.push("R");
        queue.pop();
        index++;
      } else {
        // 如果圆桶左右两边都不是要取出的球，则本轮取出流程结束
        break;
      }
    }
  }

```

```
    // 最终如果圆桶空了，则说明所有球都取出了，否则按照给定要求无法取出所有球
    if (queue.length != 0) {
      console.log("NO");
    } else {
      console.log(res.join(""));
    }
  })();
```

Go

Plain Text

```
package main

import (
  "fmt"
  "strings"
)

func main() {
  var s1, s2 string
  // 从标准输入获取两个字符串
  fmt.Scanln(&s1)
  fmt.Scanln(&s2)

  // 使用 strings.Split 函数拆分输入字符串
  input := strings.Split(s1, ",")
  output := strings.Split(s2, ",")

  n := len(input) // 获取输入字符串的长度
  var res []string  // 用于存储结果
  lst := []string{} // 用切片模拟队列（存储输入的元素）
  se := map[string]struct{}{} // 用 map 模拟集合（用于查找）
  pos := 0  // 输出数组的指针

  // 处理输入的字符串和输出的字符串
  for i := 0; i < n; i++ {
    lst = append(lst, input[i]) // 将当前输入元素加入到 lst 中
    se[input[i]] = struct{}{}   // 将当前输入元素加入到集合 se 中

    // 遍历 output 数组，检查当前元素是否在集合中
    for pos < n {
      // 如果元素在集合中
      if _, found := se[output[pos]]; found {
        // 如果队列只剩一个元素或者队列的第一个元素和 output[pos] 相同
        if len(lst) == 1 || lst[0] == output[pos] {
          res = append(res, "L") // 将 "L" 加入结果
          lst = lst[1:]          // 从队列中弹出第一个元素
        } else if lst[len(lst)-1] == output[pos] { // 如果队列的最后一个元素和 output[pos] 相同
          res = append(res, "R") // 将 "R" 加入结果
          lst = lst[:len(lst)-1]  // 从队列中弹出最后一个元素
        } else {
          break // 如果当前元素既不是队列的头也不是尾，退出
        }
        pos++ // 移动输出数组的指针
      } else {
```

```
            break // 如果当前元素不在集合中，退出
          }
        }
      }

      // 输出结果
      if len(res) != n { // 如果结果的长度不等于输入的长度，说明无法成功排列
        fmt.Println("NO")
      } else {
        for _, r := range res {
          fmt.Print(r) // 输出每一个操作（"L" 或 "R"）
        }
      }
    }
```

篮球游戏 / 队列

华为OD机试真题目录：点击去查看

> 2025B卷 200分题型

题目描述

幼儿园里有一个放倒的圆桶，它是一个线性结构，允许在桶的右边将篮球放入，可以在桶的左边和右边将篮球取出。

每个篮球有单独的编号，老师可以连续放入一个或多个篮球，小朋友可以在桶左边或右边将篮球取出，当桶只有一个篮球的情况下，必须从左边取出。

如老师按顺序放入1、2、3、4、5 共有 5 个编号的篮球，那么小朋友可以依次取出编号为1、2、3、4、5 或者 3、1、2、4、5 编号的篮球，无法取出 5、1、3、2、4 编号的篮球。

其中 3、1、2、4、5 的取出场景为：

- 连续放入1、2、3号
- 从右边取出3号
- 从左边取出1号
- 从左边取出2号
- 放入4号
- 从左边取出4号
- 放入5号
- 从左边取出5号

简答起见，我们以 L 表示左，R表示右，此时取出篮球的依次取出序列为“RLLLL”。

输入描述

每次输入包含一个测试用例：

1. 第一行的数字作为老师依次放入的篮球编号
2. 第二行的数字作为要检查是否能够按照放入的顺序取出给定的篮球的编号，其中篮球的编号用逗号进

行分隔。

其中篮球编号用逗号进行分隔。

输出描述

对于每个篮球的取出序列，如果确实可以获取，请打印出其按照左右方向的操作取出顺序，如果无法获取则打印“NO”。

备注

- 1 ⩽ 篮球编号，篮球个数 ⩽ 200
- 篮球上的数字不重复
- 输出的结果中 LR 必须为大写

用例1

输入

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

输出

```Plain Text
RLRRRLL
```

说明

> 篮球的取出顺序依次为“右、左、右、右、右、左、左”

用例2

输入

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

输出

```
NO
```

说明

> 无法取出对应序列的篮球

用例3

输入

```
1,2,3,4
1,2,3,5
```

输出

```
NO
```

说明

> 不存在编号为5的篮球，所以无法取出对应编号的数据

题解

思路： `模拟` ， 处理的代码逻辑如下：

1. 接收篮球的放入序列以及要求取出的序列，定义 `current = 0` 指向取出序列第一个位置。
2. 按照顺序放入指定篮球(队尾)，当发现output[current]被放入时，如果output[current]位于队首(L)和队尾®弹出对应篮球，current++。 如果此时不位于这两个位置，则说明指定取出序列是非法序列，终止迭代，直接输出NO。
3. 迭代2的步骤，直到将所有篮球全部放入和取出，记录其中的每个 `output[current]` 取出位置拼接而成就是答案。
4. 额外注意： 当桶只有一个篮球的情况下，必须从左边取出。

C++

Plain Text

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

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

int main() {
   string s1, s2;
   cin >> s1 >> s2;
   vector<string> input = split(s1, ",");
   vector<string> output = split(s2, ",");
   int n = input.size();
   vector<string> res;
   // 列表模拟队列
   list<string> lst;
   set<string> se;
   int pos = 0;

   for (int i = 0; i < input.size(); i++)  {
       lst.push_back(input[i]);
       se.insert(input[i]);
       // 迭代取出
       while (pos < n && se.find(output[pos]) != se.end()) {
      // 只有一个或者在最左边
           if (lst.size() == 1 || lst.front() == output[pos]) {
                res.push_back("L");
```

```
                lst.pop_front();
            } else if (lst.back() == output[pos]){
                res.push_back("R");
                lst.pop_back();
            }else {
              break;
            }
            pos++;
        }
     }
     //无法正确取出
    if (res.size() != input.size()) {
       cout << "NO";
     } else {
        for (int i = 0; i < res.size(); i++) {
            cout << res[i];
        }
     }
     return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 通用 split 函数
    private static List<String> split(String str, String delimiter) {
        return Arrays.asList(str.split(delimiter));
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String s1 = scanner.next();
        String s2 = scanner.next();
        scanner.close();

        List<String> input = split(s1, ",");
        List<String> output = split(s2, ",");
        int n = input.size();
        List<String> res = new ArrayList<>();

        // 列表模拟队列
        LinkedList<String> lst = new LinkedList<>();
        Set<String> se = new HashSet<>();
        int pos = 0;

        for (String item : input) {
            lst.add(item);
            se.add(item);

            // 迭代取出
            while (pos < n && se.contains(output.get(pos))) {
                // 只有一个或者在最左边
                if (lst.size() == 1 || lst.getFirst().equals(output.get(po
s))) {
                    res.add("L");
                    lst.removeFirst();
                } else if (lst.getLast().equals(output.get(pos))) {
                    res.add("R");
                    lst.removeLast();
                } else {
                    break;
                }
                pos++;
            }
        }

```

```
            // 无法正确取出
            if (res.size() != input.size()) {
                System.out.println("NO");
            } else {
                for (String op : res) {
                    System.out.print(op);
                }
            }
        }
    }
```

Python

Plain Text

```
import sys



def main():
    s1 = sys.stdin.readline().strip()
    s2 = sys.stdin.readline().strip()

    input_list = s1.split(",")
    output_list = s2.split(",")
    n = len(input_list)
    res = []

    # 列表模拟队列
    lst = []
    se = set()
    pos = 0

    for item in input_list:
        lst.append(item)
        se.add(item)

        # 迭代取出
        while pos < n and output_list[pos] in se:
            # 只有一个或者在最左边
            if len(lst) == 1 or lst[0] == output_list[pos]:
                res.append("L")
                lst.pop(0)
            elif lst[-1] == output_list[pos]:
                res.append("R")
                lst.pop()
            else:
                break
            pos += 1

    # 无法正确取出
    if len(res) != len(input_list):
        print("NO")
    else:
        print("".join(res))

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

Plain Text

```
const rl = require("readline").createInterface({ input: process.stdin });
var iter = rl[Symbol.asyncIterator]();
const readline = async () => (await iter.next()).value;

void (async function () {
  const inputs = (await readline()).split(",").map(Number);
  const outputs = (await readline()).split(",").map(Number);

  // 利用队列结构模拟圆桶
  const queue = [];
  // outputs[index]是要被取出的篮球的编号
  let index = 0;

  // 记录题解
  const res = [];

  for (let input of inputs) {
    // 按照放入顺序，从圆桶右边放入
    queue.push(input);

    // 然后开始尝试取出
    while (queue.length > 0) {
      // 圆桶左边的篮球的编号
      const left = queue[0];
      // 圆桶右边的篮球的编号
      const right = queue.at(-1);

      if (left == outputs[index]) {
        // 优先比较圆桶左边的篮球是不是当前要取出的篮球，优先左边的原因是：当桶只有一个篮球的情况下，必须从左边取出
        res.push("L");
        queue.shift();
        index++;
      } else if (right == outputs[index]) {
        // 比较圆桶右边的篮球是不是当前要取出的篮球
        res.push("R");
        queue.pop();
        index++;
      } else {
        // 如果圆桶左右两边都不是要取出的球，则本轮取出流程结束
        break;
      }
    }
  }

```

```
    // 最终如果圆桶空了，则说明所有球都取出了，否则按照给定要求无法取出所有球
    if (queue.length != 0) {
      console.log("NO");
    } else {
      console.log(res.join(""));
    }
  })();
```

Go

Plain Text

```
package main

import (
  "fmt"
  "strings"
)

func main() {
  var s1, s2 string
  // 从标准输入获取两个字符串
  fmt.Scanln(&s1)
  fmt.Scanln(&s2)

  // 使用 strings.Split 函数拆分输入字符串
  input := strings.Split(s1, ",")
  output := strings.Split(s2, ",")

  n := len(input) // 获取输入字符串的长度
  var res []string  // 用于存储结果
  lst := []string{} // 用切片模拟队列（存储输入的元素）
  se := map[string]struct{}{} // 用 map 模拟集合（用于查找）
  pos := 0  // 输出数组的指针

  // 处理输入的字符串和输出的字符串
  for i := 0; i < n; i++ {
    lst = append(lst, input[i]) // 将当前输入元素加入到 lst 中
    se[input[i]] = struct{}{}   // 将当前输入元素加入到集合 se 中

    // 遍历 output 数组，检查当前元素是否在集合中
    for pos < n {
      // 如果元素在集合中
      if _, found := se[output[pos]]; found {
        // 如果队列只剩一个元素或者队列的第一个元素和 output[pos] 相同
        if len(lst) == 1 || lst[0] == output[pos] {
          res = append(res, "L") // 将 "L" 加入结果
          lst = lst[1:]          // 从队列中弹出第一个元素
        } else if lst[len(lst)-1] == output[pos] { // 如果队列的最后一个元素和 output[pos] 相同
          res = append(res, "R") // 将 "R" 加入结果
          lst = lst[:len(lst)-1]  // 从队列中弹出最后一个元素
        } else {
          break // 如果当前元素既不是队列的头也不是尾，退出
        }
        pos++ // 移动输出数组的指针
      } else {
```

```
            break // 如果当前元素不在集合中，退出
        }
      }
    }

    // 输出结果
    if len(res) != n { // 如果结果的长度不等于输入的长度，说明无法成功排列
      fmt.Println("NO")
    } else {
      for _, r := range res {
        fmt.Print(r) // 输出每一个操作（"L" 或 "R"）
      }
    }
  }
```

> 来自: 华为OD 机试 – 篮球游戏 / 队列 (机考 2025 B卷 200分)-CSDN博客

篮球游戏 / 队列

华为OD机试真题目录: 点击去查看

> `2025B卷 200分题型`

题目描述

幼儿园里有一个放倒的圆桶，它是一个线性结构，允许在桶的右边将篮球放入，可以在桶的左边和右边将篮球取出。

每个篮球有单独的编号，老师可以连续放入一个或多个篮球，小朋友可以在桶左边或右边将篮球取出，当桶只有一个篮球的情况下，必须从左边取出。

如老师按顺序放入1、2、3、4、5 共有 5 个编号的篮球，那么小朋友可以依次取出编号为1、2、3、4、5 或者 3、1、2、4、5 编号的篮球，无法取出 5、1、3、2、4 编号的篮球。

其中 3、1、2、4、5 的取出场景为：

- 连续放入1、2、3号
- 从右边取出3号
- 从左边取出1号
- 从左边取出2号
- 放入4号
- 从左边取出4号
- 放入5号
- 从左边取出5号

简答起见，我们以 L 表示左，R表示右，此时取出篮球的依次取出序列为“RLLLL”。

输入描述

每次输入包含一个测试用例：

1. 第一行的数字作为老师依次放入的篮球编号
2. 第二行的数字作为要检查是否能够按照放入的顺序取出给定的篮球的编号，其中篮球的编号用逗号进行分隔。

其中篮球编号用逗号进行分隔。

输出描述

对于每个篮球的取出序列，如果确实可以获取，请打印出其按照左右方向的操作取出顺序，如果无法获取则打印“NO”。

备注

- 1 ≤ 篮球编号，篮球个数 ≤ 200
- 篮球上的数字不重复
- 输出的结果中 LR 必须为大写

用例1

输入

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

输出

```Plain Text
RLRRRLL
```

说明

> 篮球的取出顺序依次为“右、左、右、右、右、左、左”

用例2

输入

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

输出

```
NO
```

说明

> 无法取出对应序列的篮球

用例3

输入

```
1,2,3,4
1,2,3,5
```

输出

```
NO
```

说明

> 不存在编号为5的篮球，所以无法取出对应编号的数据

题解

思路： `模拟` ， 处理的代码逻辑如下：

1. 接收篮球的放入序列以及要求取出的序列，定义 `current = 0` 指向取出序列第一个位置。
2. 按照顺序放入指定篮球(队尾)，当发现output[current]被放入时，如果output[current]位于队首(L)和队尾®弹出对应篮球，current++。 如果此时不位于这两个位置，则说明指定取出序列是非法序列，终止迭代，直接输出NO。
3. 迭代2的步骤，直到将所有篮球全部放入和取出，记录其中的每个 `output[current]` 取出位置拼接而成就是答案。
4. 额外注意： 当桶只有一个篮球的情况下，必须从左边取出。

C++

Plain Text

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

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

int main() {
   string s1, s2;
   cin >> s1 >> s2;
   vector<string> input = split(s1, ",");
   vector<string> output = split(s2, ",");
   int n = input.size();
   vector<string> res;
   // 列表模拟队列
   list<string> lst;
   set<string> se;
   int pos = 0;

   for (int i = 0; i < input.size(); i++)  {
       lst.push_back(input[i]);
       se.insert(input[i]);
       // 迭代取出
       while (pos < n && se.find(output[pos]) != se.end()) {
      // 只有一个或者在最左边
           if (lst.size() == 1 || lst.front() == output[pos]) {
                res.push_back("L");
```

```
                lst.pop_front();
            } else if (lst.back() == output[pos]){
                res.push_back("R");
                lst.pop_back();
            }else {
              break;
            }
            pos++;
        }
     }
     //无法正确取出
    if (res.size() != input.size()) {
       cout << "NO";
     } else {
        for (int i = 0; i < res.size(); i++) {
            cout << res[i];
        }
     }
     return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 通用 split 函数
    private static List<String> split(String str, String delimiter) {
        return Arrays.asList(str.split(delimiter));
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String s1 = scanner.next();
        String s2 = scanner.next();
        scanner.close();

        List<String> input = split(s1, ",");
        List<String> output = split(s2, ",");
        int n = input.size();
        List<String> res = new ArrayList<>();

        // 列表模拟队列
        LinkedList<String> lst = new LinkedList<>();
        Set<String> se = new HashSet<>();
        int pos = 0;

        for (String item : input) {
            lst.add(item);
            se.add(item);

            // 迭代取出
            while (pos < n && se.contains(output.get(pos))) {
                // 只有一个或者在最左边
                if (lst.size() == 1 || lst.getFirst().equals(output.get(po
s))) {
                    res.add("L");
                    lst.removeFirst();
                } else if (lst.getLast().equals(output.get(pos))) {
                    res.add("R");
                    lst.removeLast();
                } else {
                    break;
                }
                pos++;
            }
        }

```

```
            // 无法正确取出
            if (res.size() != input.size()) {
                System.out.println("NO");
            } else {
                for (String op : res) {
                    System.out.print(op);
                }
            }
        }
    }
```

Python

Plain Text

```
import sys



def main():
    s1 = sys.stdin.readline().strip()
    s2 = sys.stdin.readline().strip()

    input_list = s1.split(",")
    output_list = s2.split(",")
    n = len(input_list)
    res = []

    # 列表模拟队列
    lst = []
    se = set()
    pos = 0

    for item in input_list:
        lst.append(item)
        se.add(item)

        # 迭代取出
        while pos < n and output_list[pos] in se:
            # 只有一个或者在最左边
            if len(lst) == 1 or lst[0] == output_list[pos]:
                res.append("L")
                lst.pop(0)
            elif lst[-1] == output_list[pos]:
                res.append("R")
                lst.pop()
            else:
                break
            pos += 1

    # 无法正确取出
    if len(res) != len(input_list):
        print("NO")
    else:
        print("".join(res))

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

Plain Text

```
const rl = require("readline").createInterface({ input: process.stdin });
var iter = rl[Symbol.asyncIterator]();
const readline = async () => (await iter.next()).value;

void (async function () {
  const inputs = (await readline()).split(",").map(Number);
  const outputs = (await readline()).split(",").map(Number);

  // 利用队列结构模拟圆桶
  const queue = [];
  // outputs[index]是要被取出的篮球的编号
  let index = 0;

  // 记录题解
  const res = [];

  for (let input of inputs) {
    // 按照放入顺序，从圆桶右边放入
    queue.push(input);

    // 然后开始尝试取出
    while (queue.length > 0) {
      // 圆桶左边的篮球的编号
      const left = queue[0];
      // 圆桶右边的篮球的编号
      const right = queue.at(-1);

      if (left == outputs[index]) {
        // 优先比较圆桶左边的篮球是不是当前要取出的篮球，优先左边的原因是：当桶只有一个篮球的情况下，必须从左边取出
        res.push("L");
        queue.shift();
        index++;
      } else if (right == outputs[index]) {
        // 比较圆桶右边的篮球是不是当前要取出的篮球
        res.push("R");
        queue.pop();
        index++;
      } else {
        // 如果圆桶左右两边都不是要取出的球，则本轮取出流程结束
        break;
      }
    }
  }

```

```
    // 最终如果圆桶空了，则说明所有球都取出了，否则按照给定要求无法取出所有球
    if (queue.length != 0) {
      console.log("NO");
    } else {
      console.log(res.join(""));
    }
  })();
```

Go

Plain Text

```
package main

import (
  "fmt"
  "strings"
)

func main() {
  var s1, s2 string
  // 从标准输入获取两个字符串
  fmt.Scanln(&s1)
  fmt.Scanln(&s2)

  // 使用 strings.Split 函数拆分输入字符串
  input := strings.Split(s1, ",")
  output := strings.Split(s2, ",")

  n := len(input) // 获取输入字符串的长度
  var res []string  // 用于存储结果
  lst := []string{} // 用切片模拟队列（存储输入的元素）
  se := map[string]struct{}{} // 用 map 模拟集合（用于查找）
  pos := 0  // 输出数组的指针

  // 处理输入的字符串和输出的字符串
  for i := 0; i < n; i++ {
    lst = append(lst, input[i]) // 将当前输入元素加入到 lst 中
    se[input[i]] = struct{}{}   // 将当前输入元素加入到集合 se 中

    // 遍历 output 数组，检查当前元素是否在集合中
    for pos < n {
      // 如果元素在集合中
      if _, found := se[output[pos]]; found {
        // 如果队列只剩一个元素或者队列的第一个元素和 output[pos] 相同
        if len(lst) == 1 || lst[0] == output[pos] {
          res = append(res, "L") // 将 "L" 加入结果
          lst = lst[1:]          // 从队列中弹出第一个元素
        } else if lst[len(lst)-1] == output[pos] { // 如果队列的最后一个元素和 output[pos] 相同
          res = append(res, "R") // 将 "R" 加入结果
          lst = lst[:len(lst)-1]  // 从队列中弹出最后一个元素
        } else {
          break // 如果当前元素既不是队列的头也不是尾，退出
        }
        pos++ // 移动输出数组的指针
      } else {
```

```
            break // 如果当前元素不在集合中，退出
          }
        }
      }

      // 输出结果
      if len(res) != n { // 如果结果的长度不等于输入的长度，说明无法成功排列
        fmt.Println("NO")
      } else {
        for _, r := range res {
          fmt.Print(r) // 输出每一个操作（"L" 或 "R"）
        }
      }
    }
```

来自: 华为OD 机试 – 篮球游戏 / 队列 (机考 2025 B卷 200分)-CSDN博客

来自: 华为OD 机试 – 篮球游戏 / 队列 (机考 2025 B卷 200分)-CSDN博客

华为OD 机考 - 分苹果 (2025 B卷 100分)-CSDN 博客

分苹果

华为OD机试真题目录: 点击去查看

> `2025 B卷 100分题型`

题目描述

A、B两个人把苹果分为两堆，A希望按照他的计算规则等分苹果，他的计算规则是按照二进制加法计算，并且不计算进位

12+5=9（1100 + 0101 = 9），B的计算规则是十进制加法，包括正常进位，B希望在满足A的情况下获取苹果重量最多。

输入苹果的数量和每个苹果重量，输出满足A的情况下B获取的苹果总重量。

如果无法满足A的要求，输出-1。

数据范围:

- 1 <= 总苹果数量 <= 20000
- 1 <= 每个苹果重量 <= 10000

输入描述

输入第一行是苹果数量：3

输入第二行是每个苹果重量：3 5 6

输出描述

输出第一行是B获取的苹果总重量：11

示例1

输入

Plain Text

```
3
3 5 6
```

输出

```
11
```

示例2

输入

```
8
7258 6579 2602 6716 3050 3564 5396 1773
```

输出

```
35165
```

题解

思路： A进行运算实际上是 `异或运算` 。

1. 首先考虑什么时候能满足A的要求。要求A B 分配苹果内部之间的异或运算的结果相同,假设为x， `x ^ x = 0` ，所以所有苹果进行异或运算的结果最终要为0。不然就输出-1.
2. 现在考虑什么时候B能收到最大重量苹果.考虑一个公式 `x ^ 0 = x` ，让A分到的苹果最小就能保证B能收到最大重要，所以给A分配一个最轻的苹果即可。即本题的结果就是满足1的情况，最轻苹果的重量。

C++

Plain Text

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


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

int main() {
    int n ;
    cin >> n;
    vector<int> ans(n, 0);
    int sum = 0;
    int totle = 0;
    int minValue = 20000;
    for (int i = 0; i < n; i++) {
        int tmp;
        cin >> tmp;
        sum = sum ^ tmp;
        ans[i] = tmp;
        totle += ans[i];
        minValue = min(minValue, ans[i]);
    }
    // 如果所有异或运算的结果不为0，说明不会满足A的要求
    if (sum != 0) {
        cout << -1;

    } else  {
```

```
        // a ^ a = 0 , 让A 取最小值
        cout << totle - minValue;
    }
     return 0;
}
```

Java

Plain Text

```
import java.util.*;

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

        // 读取 n
        int n = scanner.nextInt();
        int[] ans = new int[n];
        int sum = 0;
        int total = 0;
        int minValue = 20000;

        for (int i = 0; i < n; i++) {
            int tmp = scanner.nextInt();
            sum ^= tmp;  // 计算异或和
            ans[i] = tmp;
            total += tmp;  // 计算总和
            minValue = Math.min(minValue, tmp);  // 取最小值
        }

        // 如果异或和不为 0，说明无法满足 A 的要求
        if (sum != 0) {
            System.out.println(-1);
        } else {
            // 利用异或性质，取出最小值
            System.out.println(total - minValue);
        }

        scanner.close();
    }
}
```

Python

Plain Text

```
import sys

def main():
    # 读取 n
    n = int(sys.stdin.readline().strip())

    ans = []
    total = 0
    xor_sum = 0
    min_value = 20000

    # 读取数据并计算异或值、总和和最小值
    numbers = list(map(int, sys.stdin.readline().strip().split()))
    for num in numbers:
        xor_sum ^= num
        total += num
        min_value = min(min_value, num)

    # 如果异或值不为 0，输出 -1，否则输出 total - min_value
    if xor_sum != 0:
        print(-1)
    else:
        print(total - min_value)

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

JavaScript

Plain Text

```
process.stdin.resume();
process.stdin.setEncoding("utf-8");

let input = "";
process.stdin.on("data", function (chunk) {
    input += chunk;
});

process.stdin.on("end", function () {
    let lines = input.trim().split("\n");

    // 读取 n
    let n = parseInt(lines[0]);
    let numbers = lines[1].split(" ").map(Number);

    let sum = 0;
    let total = 0;
    let minValue = 20000;

    // 计算异或值、总和、最小值
    for (let i = 0; i < n; i++) {
        sum ^= numbers[i];
        total += numbers[i];
        minValue = Math.min(minValue, numbers[i]);
    }

    // 如果异或值不为 0，输出 -1，否则输出 total - minValue
    if (sum !== 0) {
        console.log(-1);
    } else {
        console.log(total - minValue);
    }
});
```

Go

Plain Text

```
package main

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

func main() {
  // 使用 bufio 进行高效输入
  reader := bufio.NewReader(os.Stdin)

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

  // 读取所有数字
  line, _ = reader.ReadString('\n')
  numsStr := strings.Fields(line)

  nums := make([]int, n)
  sum := 0
  total := 0
  minValue := 20000

  // 计算异或值、总和、最小值
  for i := 0; i < n; i++ {
    nums[i], _ = strconv.Atoi(numsStr[i])
    sum ^= nums[i]
    total += nums[i]
    if nums[i] < minValue {
      minValue = nums[i]
    }
  }

  // 如果异或值不为 0，输出 -1，否则输出 total - minValue
  if sum != 0 {
    fmt.Println(-1)
  } else {
    fmt.Println(total - minValue)
  }
}
```

> 来自: 华为OD 机考 – 分苹果 (2025 B卷 100分)-CSDN博客

华为od 机试 - 停车费用统计 (2025 B卷 100分)- CSDN博客

停车费用统计

华为OD机试真题目录: 点击去查看

> 2025 B卷 100分题型

题目描述

停车场统计当日总收费，包月的车不统计，不包月的车半个小时收一块钱，不满半小时不收钱，如果超过半小时，零头不满半小时按半小时算，每天11:30-13:30时间段不收钱， 如果一辆车停车时间超过8小时后不收费(网友回忆，数值不一定为8，正式机考的时候注意一下)。输入第一行n（今日进出停车场的包月的车辆数）下一个行输出 包月车的车牌号以空格分割 接下来每一行输出进出停车场的车辆信息，进入格式为 （时间 车牌号 enter） 出去格式为（时间 车牌号 leave）

输入描述

输入第一行n（今日进出停车场的包月的车辆数）下一个行输出 包月车的车牌号以空格分割 接下来每一行输出进出停车场的车辆信息，进入格式为 （时间 车牌号 enter） 出去格式为（时间 车牌号 leave）

输出描述

输出今日总收费

用例1

输入

Plain Text

```
1
AAAA
8:00 AAA enter
9:00 AAA leave
```

输出

Plain Text

```
0
```

说明

> 只有包月车

用例2

输入

```Plain Text
1
AAAA
8:00 AAA enter
9:00 AAA leave
8:00 BBB enter
9:00 BBB leave
```

输出

```Plain Text
2
```

题解

思路：`模拟`

1. 接收包月车牌号，可以使用 `集合` 存储，翻遍后序尽心车辆是否为包月车判断。
2. 接收车辆进入情况，使用 `哈希表` 存储每辆非包月车的进出时间(使用1构建的集合尽心判断，去除包月的统计),，解析字符串将时间转换为分钟。
3. 分别模拟计算每辆车的交的停车位，每辆车的费用计算逻辑如下：
   a. `start end` 为每辆车的进场时间和出场时间。如果 `end - start < 30` 停车计费为零。
   b. 转换车的出厂时间为 `realEnd = min(end, start + 8 * 60)`， 连续停车时间超过 `8小时` 则停止计费。
   c. 车辆的实际收费区间为 `start realEnd`，现在额外需要考虑实际收费区间和免费区间的重叠情况。 `chargeTime = readlEnd - start - 免费区间重叠时间` 就是实际收费时长。
   d. 单辆车的收费为 `(chaegeTime + 29)/ 30` 元。向上取整。
4. 累加和每非包月辆车的收费即为当天总收费。

C++

Plain Text

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


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


// 将时间转换为int 统一以分钟表示
int parseTime(string& time) {
    vector<int> tmp  = split(time, ":");
    int res = 0;
    res = tmp[1] + tmp[0] * 60;
    return res;
}

// 计算每辆车的花费
int cost(int begin, int end) {
   if (end - begin < 30) {
       return 0;
    }
    string freeTimeBeginStr = "11:30";
    string freeTimeEndStr = "13:30";
    int freeTimeBegin =  parseTime(freeTimeBeginStr);
    int freeTimeEnd = parseTime(freeTimeEndStr);

    // 8个小时是考友回忆的，实际可能不是8个小时，考的时候根据题目描述来更改
```

```
    int maxTime = 8 * 60;
    // 到达停车收费最长时间停止收费
    end = min(end, begin + maxTime);

    // 收费时间
    int chargeTime = end - begin;

    // 忽略免费时间段 和免费时间有交集
    if (begin <= freeTimeEnd && end >= freeTimeBegin ) {
        // 跨免费时间的时长
        chargeTime -= min(freeTimeEnd, end) - max(freeTimeBegin, begin);
    }
    return (chargeTime + 29) / 30;
}


int main() {
    int n;
    cin >> n;
    string time, card, direct;
    set<string> monthlyPaymentCard;
    for (int i = 0; i <  n; i++) {
        string card;
        cin >> card;
        monthlyPaymentCard.insert(card);
    }

    // 记录非包月车的进出时间
    map<string, pair<int, int>> mp;
    while (cin >> time >> card >> direct) {
        if (time == "") {
            break;
        }
        // 忽略包月车
        if (monthlyPaymentCard.count(card)) {
            continue;
        }
        int timeMimute = parseTime(time);
        if (direct == "enter") {
            mp[card].first = timeMimute;
        } else {
            mp[card].second = timeMimute;
        }
    }


    int res = 0;
    for (auto p : mp) {
```

```
            int start = p.second.first;
            int end = p.second.second;
            // 计算每辆车的花费
            res += cost(start, end);
        }

        cout << res;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 将时间字符串转换为分钟数
    public static int parseTime(String time) {
        String[] parts = time.split(":");
        return Integer.parseInt(parts[0]) * 60 + Integer.parseInt(parts
[1]);
    }

    // 计算费用（忽略免费时段 + 超过8小时不收费）
    public static int cost(int begin, int end) {
        if (end - begin < 30) return 0;

        int freeBegin = parseTime("11:30");
        int freeEnd = parseTime("13:30");
        int maxTime = 8 * 60;

        // 限制最大收费时长为8小时
        end = Math.min(end, begin + maxTime);

        int chargeTime = end - begin;

        // 跨越中午免费时间段的部分不收费
        if (begin <= freeEnd && end >= freeBegin) {
            chargeTime -= Math.min(end, freeEnd) - Math.max(begin, freeBeg
in);
        }

        // 每30分钟1元，向上取整
        return (chargeTime + 29) / 30;
    }

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

        // 读取包月车牌号
        Set<String> monthlyCars = new HashSet<>();
        String[] monthly = sc.nextLine().split(" ");
        if (n > 0) {
            monthlyCars.addAll(Arrays.asList(monthly));
        }

        // 存储非包月车的进出时间
```

```
        Map<String, int[]> carTimes = new HashMap<>();

        while (sc.hasNextLine()) {
            String line = sc.nextLine().trim();
            if (line.isEmpty()) continue;

            String[] parts = line.split(" ");
            String time = parts[0];
            String plate = parts[1];
            String status = parts[2];

            // 包月车跳过
            if (monthlyCars.contains(plate)) continue;

            int minutes = parseTime(time);
            carTimes.putIfAbsent(plate, new int[2]);

            if (status.equals("enter")) {
                carTimes.get(plate)[0] = minutes;
            } else {
                carTimes.get(plate)[1] = minutes;
            }
        }

        int total = 0;
        for (int[] times : carTimes.values()) {
            total += cost(times[0], times[1]);
        }

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

Python

Plain Text

```
# 将时间字符串转换为分钟数
def parse_time(t):
    h, m = map(int, t.split(":"))
    return h * 60 + m

# 计算每辆车的花费
def cost(begin, end):
    if end - begin < 30:
        return 0

    free_begin = parse_time("11:30")
    free_end = parse_time("13:30")
    max_time = 8 * 60

    # 到达停车收费最长时间停止收费
    end = min(end, begin + max_time)

    # 收费时间
    charge_time = end - begin

    # 跨越中午免费时间段的部分不收费
    if begin <= free_end and end >= free_begin:
        charge_time -= min(end, free_end) - max(begin, free_begin)

    # 每30分钟1元，向上取整
    return (charge_time + 29) // 30

def main():
    import sys

    lines = [line.strip() for line in sys.stdin if line.strip()]
    n = int(lines[0])
    monthly = set()

    if n > 0:
        monthly = set(lines[1].split())

    records = lines[1 + (1 if n > 0 else 0):]

    # 存储非包月车的进出时间
    car_times = {}

    for record in records:
        time_str, plate, status = record.split()
        # 忽略包月车
```

```
            if plate in monthly:
                continue
            t = parse_time(time_str)
            if plate not in car_times:
                car_times[plate] = [0, 0]
            if status == 'enter':
                car_times[plate][0] = t
            else:
                car_times[plate][1] = t

        total = 0
        for start, end in car_times.values():
            total += cost(start, end)

        print(total)

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

JavaScript

Plain Text

```
// 将时间字符串转换为分钟数
function parseTime(t) {
    let [h, m] = t.split(":").map(Number);
    return h * 60 + m;
}

// 计算每辆车的花费
function cost(begin, end) {
    if (end - begin < 30) return 0;

    let freeBegin = parseTime("11:30");
    let freeEnd = parseTime("13:30");
    let maxTime = 8 * 60;

    // 限制最大收费时间
    end = Math.min(end, begin + maxTime);

    let chargeTime = end - begin;

    // 跨中午免费时间段的不计费
    if (begin <= freeEnd && end >= freeBegin) {
        chargeTime -= Math.min(end, freeEnd) - Math.max(begin, freeBegin);
    }

    // 每30分钟1元，向上取整
    return Math.floor((chargeTime + 29) / 30);
}

// 读取标准输入
const readline = require("readline");
const rl = readline.createInterface({ input: process.stdin });

let input = [];

rl.on("line", (line) => {
    if (line.trim() !== "") input.push(line.trim());
});

rl.on("close", () => {
    let n = parseInt(input[0]);
    let monthly = new Set();
    if (n > 0) {
        input[1].split(" ").forEach(p => monthly.add(p));
    }

```

```
    let carTimes = {};
    let records = input.slice(1 + (n > 0 ? 1 : 0));

    for (let line of records) {
        let [time, plate, status] = line.split(" ");
        // 忽略包月车
        if (monthly.has(plate)) continue;

        let t = parseTime(time);
        if (!carTimes[plate]) carTimes[plate] = [0, 0];
        if (status === "enter") {
            carTimes[plate][0] = t;
        } else {
            carTimes[plate][1] = t;
        }
    }

    let total = 0;
    for (let [start, end] of Object.values(carTimes)) {
        total += cost(start, end);
    }

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

Go

Plain Text

```
package main

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

// 将时间字符串转换为分钟数
func parseTime(t string) int {
    parts := strings.Split(t, ":")
    h, _ := strconv.Atoi(parts[0])
    m, _ := strconv.Atoi(parts[1])
    return h*60 + m
}

// 计算每辆车的花费
func cost(begin, end int) int {
    if end-begin < 30 {
        return 0
    }

    freeBegin := parseTime("11:30")
    freeEnd := parseTime("13:30")
    maxTime := 8 * 60

    // 限制最大收费时间
    if end > begin+maxTime {
        end = begin + maxTime
    }

    chargeTime := end - begin

    // 跨中午免费时间段的不计费
    if begin <= freeEnd && end >= freeBegin {
        overlap := min(end, freeEnd) - max(begin, freeBegin)
        chargeTime -= overlap
    }

    // 每30分钟1元，向上取整
    return (chargeTime + 29) / 30
}

```

```
func max(a, b int) int {
    if a > b {
        return a
    }
    return b
}

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

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    n, _ := strconv.Atoi(scanner.Text())

    monthly := make(map[string]bool)
    if n > 0 {
        scanner.Scan()
        plates := strings.Fields(scanner.Text())
        for _, plate := range plates {
            monthly[plate] = true
        }
    }

    carTimes := make(map[string][2]int)

    for scanner.Scan() {
        line := scanner.Text()
        if line == "" {
            continue
        }

        parts := strings.Fields(line)
        timeStr, plate, status := parts[0], parts[1], parts[2]
        // 忽略包月车
        if monthly[plate] {
            continue
        }

        t := parseTime(timeStr)
        record := carTimes[plate]

        if status == "enter" {
            record[0] = t
```

```
            } else {
                record[1] = t
            }
            carTimes[plate] = record
        }

        total := 0
        for _, times := range carTimes {
            total += cost(times[0], times[1])
        }

        fmt.Println(total)
    }
```

> 来自: 华为od 机试 – 停车费用统计 (2025 B卷 100分)-CSDN博客

华为OD 机试 - 字符串解密 (2025 B卷 100分)_华为od机考-CSDN博客

字符串解密

华为OD机试真题目录: 点击去查看

> 2025 B卷 100分题型

题目描述

给定两个字符串string1和string2。 string1是一个被加扰的字符串。string1由小写英文字母（’a’~ ’z’）和数字字符（’0’~ ’9’）组成，而加扰字符串由’0’~ ’9’、’a’~’f’组成。
string1里面可能包含0个或多个加扰子串，剩下可能有0个或多个有效子串，这些有效子串被加扰子串隔开。string2是一个参考字符串，仅由小写英文字母（’a’~’z’）组成。你需要在string1字符串里找到一个有效子串，这个有效子串要同时满足下面两个条件： （1）这个有效子串里不同字母的数量不超过且最接近于string2里不同字母的数量，即小于或等于string2里不同字母的数量的同时且最大。（2）这个有效子串是满足条件（1）里的所有子串（如果有多个的话）里字典序最大的一个。如果没有找到合适条件的子串的话，请输出”Not Found”

输入描述

input_string1 input_string2

输出描述

output_string

用例1

输入

Plain Text

```
123admyffc79pt
ssyy
```

输出

```
pt
```

用例2

输入

```
123admyffc79ptaagghi2222smeersst88mnrt
ssyyfgh
```

输出

```
mnrt
```

题解

思路： 正则+模拟

1. 通过正则表达式 `[0-9a-f]+` 分割出有效字串. 可以理解成数字[0-9]和[a-f]字符会将字符串进行切割。
2. 如果有效字串个数为0，直接输出 `Not Found` .循环判断每个有效字串是否满足第一个要求中(字母种类限制)。满足的话加入结果候选集。
3. 把结果候选集按照要求2和要求1进行排序,得出结果

C++

Plain Text

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

// 按照正则表达式进行切割
vector<string> splitByRegex(string str, string pattern) {
    regex re(pattern);
    sregex_token_iterator it(str.begin(), str.end(), re, -1);
    sregex_token_iterator end;
    vector<string> res;
    while (it != end) {
        // 忽略空字符串
        if (!it->str().empty()) {
            res.push_back(*it);
        }
        ++it;
    }
    return res;
}



bool cmp(string s1, string s2) {
    set<char> set1(s1.begin(), s1.end());  // 计算 s1 唯一字符个数
    set<char> set2(s2.begin(), s2.end());  // 计算 s2 唯一字符个数

    if (set1.size() == set2.size()) {
        return s1 > s2;  // 按字典序降序排序
    }
    return set1.size() > set2.size();  // 唯一字符多的排前面
}

int main() {
    string input1, input2;
    getline(cin, input1);
    getline(cin, input2);
    // 分割取出 有效字串
    vector<string> splitString = splitByRegex(input1, "[0-9a-f]+");
```

```
    if (splitString.empty()) {
        cout << "Not Found";
        return 0;
    }



    set<char> s(input2.begin(), input2.end());
    vector<string> valids;
    //取出所有满足条件的有效字串
    for (int i = 0; i < splitString.size(); i++) {
        set<char> tmp(splitString[i].begin(), splitString[i].end());
        if (tmp.size() > s.size()) {
            continue;
        }
        valids.push_back(splitString[i]);
    }

    if (valids.empty()) {
        cout << "Not Found";
        return 0;
    }
   // 按照要求一和要求二进行排序
    sort(valids.begin(), valids.end(),cmp);
    cout << valids[0];
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;
import java.util.regex.*;

public class Main {
    // 按正则表达式进行切割
    public static List<String> splitByRegex(String str, String pattern) {
        List<String> res = new ArrayList<>();
        Pattern regex = Pattern.compile(pattern);
        Matcher matcher = regex.matcher(str);
        int lastEnd = 0;

        while (matcher.find()) {
            if (lastEnd < matcher.start()) {
                res.add(str.substring(lastEnd, matcher.start()));
            }
            lastEnd = matcher.end();
        }
        if (lastEnd < str.length()) {
            res.add(str.substring(lastEnd));
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input1 = scanner.nextLine();
        String input2 = scanner.nextLine();
        scanner.close();

        // 分割取出有效字串
        List<String> splitString = splitByRegex(input1, "[0-9a-f]+");

        if (splitString.isEmpty()) {
            System.out.println("Not Found");
            return;
        }

        Set<Character> validChars = new HashSet<>();
        for (char c : input2.toCharArray()) {
            validChars.add(c);
        }

        List<String> valids = new ArrayList<>();

        // 取出所有满足条件的有效子串
```

```
        for (String str : splitString) {
            Set<Character> uniqueChars = new HashSet<>();
            for (char c : str.toCharArray()) {
                uniqueChars.add(c);
            }

            if (uniqueChars.size() > validChars.size()) {
                continue;
            }
            valids.add(str);
        }

        if (valids.isEmpty()) {
            System.out.println("Not Found");
            return;
        }

        // 自定义排序：唯一字符数优先，字典序降序
        valids.sort((a, b) -> {
            int uniqueA = new HashSet<>(Arrays.asList(a.chars().mapToObj(c -> (char) c).toArray(Character[]::new))).size();
            int uniqueB = new HashSet<>(Arrays.asList(b.chars().mapToObj(c -> (char) c).toArray(Character[]::new))).size();

            if (uniqueA != uniqueB) {
                return Integer.compare(uniqueB, uniqueA);
            }
            return b.compareTo(a);
        });

        System.out.println(valids.get(0));
    }
}
```

Python

Plain Text

```
import sys
import re
from functools import cmp_to_key

def split_by_regex(s, pattern):
    return [x for x in re.split(pattern, s) if x]

def cmp(s1, s2):
    set1, set2 = set(s1), set(s2)
    if len(set1) == len(set2):
        return -1 if s1 > s2 else (1 if s1 < s2 else 0)  # 字典序降序
    return -1 if len(set1) > len(set2) else 1  # 唯一字符数降序

def main():
    input1 = sys.stdin.readline().strip()
    input2 = sys.stdin.readline().strip()

    split_string = split_by_regex(input1, r'[0-9a-f]+')
    if not split_string:
        print("Not Found")
        return

    s_set = set(input2)
    valids = [word for word in split_string if len(set(word)) <= len(s_se
t)]

    if not valids:
        print("Not Found")
        return

    valids.sort(key=cmp_to_key(cmp))
    print(valids[0])

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

JavaScript

Plain Text

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

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

let inputs = [];

rl.on('line', (line) => {
    inputs.push(line);
    if (inputs.length === 2) {
        rl.close();
    }
});

rl.on('close', () => {
    let input1 = inputs[0];
    let input2 = inputs[1];

    // 按正则表达式分割
    let splitString = input1.split(/[0-9a-f]+/).filter(s => s.length > 0);

    if (splitString.length === 0) {
        console.log("Not Found");
        return;
    }

    let validChars = new Set(input2);
    let valids = [];

    // 取出所有满足条件的有效子串
    splitString.forEach(str => {
        let uniqueChars = new Set(str);
        if (uniqueChars.size <= validChars.size) {
            valids.push(str);
        }
    });

    if (valids.length === 0) {
        console.log("Not Found");
        return;
    }

    // 自定义排序：唯一字符数优先，字典序降序
```

```
    valids.sort((a, b) => {
        let uniqueA = new Set(a).size;
        let uniqueB = new Set(b).size;

        if (uniqueA !== uniqueB) {
            return uniqueB - uniqueA;
        }
        return b.localeCompare(a);
    });

    console.log(valids[0]);
});
```

Go

Plain Text

```
package main

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

// 按照正则表达式切割字符串
func splitByRegex(str, pattern string) []string {
  re := regexp.MustCompile(pattern)
  split := re.Split(str, -1)
  var res []string
  for _, s := range split {
    if len(s) > 0 {
      res = append(res, s)
    }
  }
  return res
}

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

  // 读取两行输入
  scanner.Scan()
  input1 := scanner.Text()
  scanner.Scan()
  input2 := scanner.Text()

  // 分割取出有效字串
  splitString := splitByRegex(input1, "[0-9a-f]+")

  if len(splitString) == 0 {
    fmt.Println("Not Found")
    return
  }

  validChars := make(map[rune]bool)
  for _, c := range input2 {
    validChars[c] = true
  }

```

```
  var valids []string

  // 取出所有满足条件的有效子串
  for _, str := range splitString {
    uniqueChars := make(map[rune]bool)
    for _, c := range str {
      uniqueChars[c] = true
    }

    if len(uniqueChars) > len(validChars) {
      continue
    }
    valids = append(valids, str)
  }

  if len(valids) == 0 {
    fmt.Println("Not Found")
    return
  }

  // 自定义排序
  sort.Slice(valids, func(i, j int) bool {
    uniqueI := len(mapify(valids[i]))
    uniqueJ := len(mapify(valids[j]))

    if uniqueI != uniqueJ {
      return uniqueI > uniqueJ
    }
    return valids[i] > valids[j] // 字典序降序
  })

  fmt.Println(valids[0])
}

// mapify 计算字符串的唯一字符数
func mapify(s string) map[rune]bool {
  m := make(map[rune]bool)
  for _, c := range s {
    m[c] = true
  }
  return m
}
```

> 来自: 华为OD 机试 – 字符串解密 (2025 B卷 100分)_华为od机考-CSDN博客

华为OD机试 - 构建数列 (2025 B卷 100分)_题目描述构造数列,第一个数为n,后面的数不大于前一个的一半,数列奇偶相间或许全为-CSDN博客

构建数列

华为OD机试真题目录: 点击去查看

> `2025B卷 100分题型`

题目描述

构造数列，第一个数为n，后面的数不大于前一个的一半，数列奇偶相间或许全为奇数或者全为偶数，数列的元素都是正整数，能构造多少数列。

输入描述

输入一个n

备注

- `1 <= n < 10000`

输出描述

输出可以构造的序列个数

用例1

输入

```
Plain Text
1  7
```

输出

```
Plain Text
1  6
```

说明

> 可以构成 [7], [7,3], [7,2], [7, 1], [7,3,1],[7,2,1]

用例2

输入

```
4
```

输出

```
3
```

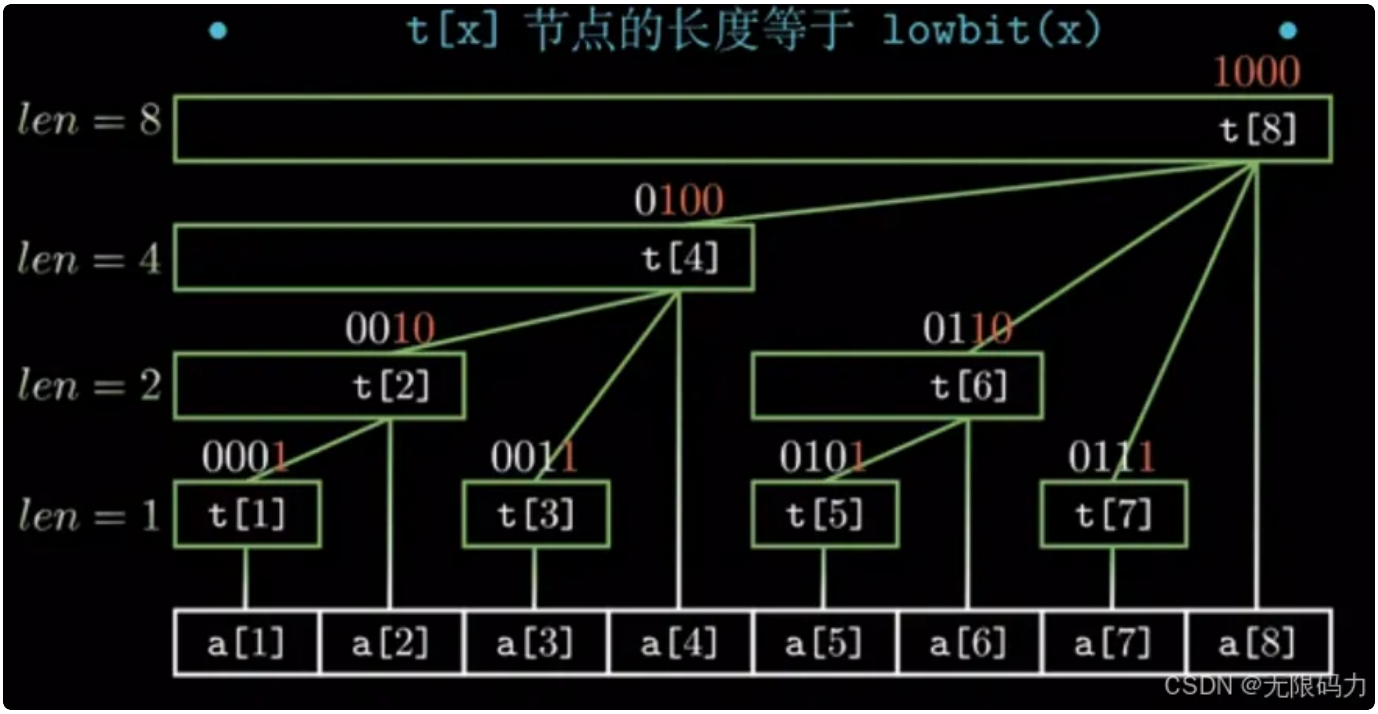
说明

> 可以构成 [4],[4,2],[4,1]

题解

思路：`动态规划 + 树状数组`

1. 状态转移规律：
   - 全奇序列：以奇数x开头的序列的前一个节点必须 `i <= x /2` 的奇数， `dp[x] = sum(dp[j]) +1 其中j 的取值[1, x / 2], 且j为奇数` 。
   - 全偶序列：以偶数x开头的序列的前一个节点必须 `i <= x /2` 的偶数， `dp[x] = sum(dp[j]) + 1 其中j 的取值[1, x / 2], 且j为偶数` 。
   - 奇偶交替序列：需要利用到两个数组 `dp_even 和 dp_odd` 分别表示 `奇偶交替，存储结尾为偶数。` 和 `奇偶交替，存储结尾为奇数` 。如果当前x为偶数，它的前一个节点必须是[1, x/2]范围的奇数并且需要满足奇偶交替，那么递推公式为 `dp_even[x] = sum(dp_odd[j]) + 1 j为奇数，并且j的范围为[1, x / 2]` . 类似x为奇数的情况下 `dp_odd[x] = sum(dp_even[j]) + 1 j为偶数，并且j的范围为[1,x/2]`
   - 解释为什么每次为什么额外+1，因为单独的一个x也可以组成一个序列。
2. 通过装态转移方程可以看出，这道题涉及到大量 `区间求和` ，数据量 `n <= 10000` ，使用暴力方式进行区间求和会超时，这里引入 `树状数组` 进行优化。树状数组可以快速求出一段区间和。 线段树其实也是ok的，可以自行百度了解这两个数据结构。下图为线段树数据结构的存储逻辑。



1. 明白1 2 的规律之后，定义 `fenw0_odd[] fenw1_even[] fenw2_even[] fenw2_odd[]` 数组表示 `全奇序列、全偶序列、奇偶交替，最后一个为偶数、奇偶交替，最后一个为奇数`，枚举 `1 – n` 递推对应的状态方程即可。n为偶数的情况下结果为 `fenw1_even[n] + fenw2_even[n]`，n为奇数的情况下 `fenw1_odd[n] + fenw2_odd[n]`

> 额外注意一点，递归过程中会超过int的范围，除python外的语言推荐选择比int数据范围大的数据类型。

C++

Plain Text

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

// 树状数组 更新值
// 参数：tree 数组，index 更新的位置，delta 增加的值
void update(vector<long long> &tree, int index, long long delta) {
    int n = (int)tree.size() - 1;
    while (index <= n) {
        tree[index] += delta;  // 更新当前位置的值
        index += index & (-index);  // 跳到所有包含这个位置的父区间
    }
}

// 树状数组 查询区间和
// 参数：tree 数组，index 查询的前缀位置
// 返回值：从1到index的区间和
long long query(const vector<long long> &tree, int index) {
    long long res = 0;
    while (index > 0) {
        res += tree[index];  // 累加当前节点值
        index -= index & (-index);  // 跳到上一个区间的父区间
    }
    return res;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;

    // 初始化四个树状数组，长度为 n+1，位置从1开始
    vector<long long> fenw0_odd(n + 1, 0);   // 全奇数序列
    vector<long long> fenw1_even(n + 1, 0);  // 全偶数序列
    vector<long long> fenw2_even(n + 1, 0);  // 奇偶交替，下一位需为奇数
    vector<long long> fenw2_odd(n + 1, 0);   // 奇偶交替，下一位需为偶数

    long long ans0 = 0, ans1 = 0, ans2 = 0;

    // 从1遍历到n
    for (int x = 1; x <= n; ++x) {
        int half = x / 2;  // 二分之一，用于查询前面结果
        if (x % 2 == 1) {  // x 是奇数
```

```
            // 查询前一半长度的奇数序列和奇偶交替序列
            long long dp0_val = query(fenw0_odd, half);
            long long dp2_val = query(fenw2_even, half);

            // 更新当前奇数位置的dp值
            update(fenw0_odd, x, 1 + dp0_val);
            update(fenw2_odd, x, 1 + dp2_val);

            // 到最后一个元素时保存答案
            if (x == n) {
                ans0 = dp0_val;
                ans2 = dp2_val;
            }
        } else {  // x 是偶数
            // 查询前一半长度的偶数序列和奇偶交替序列
            long long dp1_val = query(fenw1_even, half);
            long long dp2_val = query(fenw2_odd, half);

            // 更新当前偶数位置的dp值
            update(fenw1_even, x, 1 + dp1_val);
            update(fenw2_even, x, 1 + dp2_val);

            if (x == n) {
                ans1 = dp1_val;
                ans2 = dp2_val;
            }
        }
    }

    // 总序列数 = 1 + 三个状态的答案和
    cout << 1 + ans0 + ans1 + ans2 << "\n";

    return 0;
}
```

JAVA

Plain Text

```
import java.util.Scanner;

public class Main {

    // 树状数组 更新操作
    // tree：树状数组，index：更新的位置，delta：增加的值
    static void update(long[] tree, int index, long delta) {
        int n = tree.length - 1;
        while (index <= n) {
            tree[index] += delta; // 更新当前位置的值
            index += index & (-index); // 跳到所有包含该位置的父区间
        }
    }

    // 查询1到index的前缀和
    static long query(long[] tree, int index) {
        long res = 0;
        while (index > 0) {
            res += tree[index]; // 累加当前节点的值
            index -= index & (-index); // 跳到上一个区间
        }
        return res;
    }

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

        int n = sc.nextInt();

        // 初始化四个树状数组，长度为n+1，索引从1开始
        long[] fenw0_odd = new long[n + 1];   // 全奇数序列
        long[] fenw1_even = new long[n + 1];  // 全偶数序列
        long[] fenw2_even = new long[n + 1];  // 奇偶交替，下一位需为奇数
        long[] fenw2_odd = new long[n + 1];   // 奇偶交替，下一位需为偶数

        long ans0 = 0, ans1 = 0, ans2 = 0;

        for (int x = 1; x <= n; x++) {
            int half = x / 2;
            if (x % 2 == 1) {  // 当前是奇数位置
                // 查询前半部分奇数序列和奇偶交替序列的和
                long dp0_val = query(fenw0_odd, half);
                long dp2_val = query(fenw2_even, half);

                // 更新当前位置奇数序列和奇偶交替序列
```

```
                update(fenw0_odd, x, 1 + dp0_val);
                update(fenw2_odd, x, 1 + dp2_val);

                if (x == n) {
                    ans0 = dp0_val;
                    ans2 = dp2_val;
                }
            } else {  // 当前是偶数位置
                // 查询前半部分偶数序列和奇偶交替序列的和
                long dp1_val = query(fenw1_even, half);
                long dp2_val = query(fenw2_odd, half);

                // 更新当前位置偶数序列和奇偶交替序列
                update(fenw1_even, x, 1 + dp1_val);
                update(fenw2_even, x, 1 + dp2_val);

                if (x == n) {
                    ans1 = dp1_val;
                    ans2 = dp2_val;
                }
            }
        }

        // 结果为 1 + 三个状态的答案之和
        System.out.println(1 + ans0 + ans1 + ans2);

        sc.close();
    }
}
```

Python

Plain Text

```
import sys

sys.setrecursionlimit(10**7)

# 树状数组 更新值
def update(tree, index, delta):
    n = len(tree) - 1
    while index <= n:
        tree[index] += delta
        index += index & -index

# 树状查询组 区间和
def query(tree, index):
    res = 0
    while index:
        res += tree[index]
        index -= index & -index
    return res


def main():
    n = int(input())

    size = n
    fenw0_odd = [0] * (size + 1)  # 全奇数序列
    fenw1_even = [0] * (size + 1)  # 全偶数序列

    fenw2_even = [0] * (size + 1)  # 奇偶交替，下一位需为奇数
    fenw2_odd = [0] * (size + 1)   # 奇偶交替，下一位需为偶数

    ans0 = ans1 = ans2 = 0

    for x in range(1, n + 1):
        half = x // 2
        if x % 2 == 1:  # x 是奇数
            dp0_val = query(fenw0_odd, half)
            dp2_val = query(fenw2_even, half)

            update(fenw0_odd, x, 1 + dp0_val)
            update(fenw2_odd, x, 1 + dp2_val)

            if x == n:
                ans0 = dp0_val
                ans2 = dp2_val
        else:  # x 是偶数
```

```
            dp1_val = query(fenw1_even, half)
            dp2_val = query(fenw2_odd, half)

            update(fenw1_even, x, 1 + dp1_val)
            update(fenw2_even, x, 1 + dp2_val)

            if x == n:
                ans1 = dp1_val
                ans2 = dp2_val

    total_sequences = 1 + ans0 + ans1 + ans2
    print(total_sequences)


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

JavaScript

Plain Text

```
// 树状数组 更新值，delta 是 BigInt 类型
function update(tree, index, delta) {
    let n = tree.length - 1;
    while (index <= n) {
        tree[index] += delta; // 更新当前节点
        index += index & -index; // 跳转父区间
    }
}

// 树状数组 查询区间和，返回 BigInt
function query(tree, index) {
    let res = 0n; // 0n 表示 BigInt 类型0
    while (index > 0) {
        res += tree[index]; // 累加当前节点值
        index -= index & -index; // 跳转到上一个区间
    }
    return res;
}

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

rl.on("line", (line) => {
    let n = Number(line.trim());

    // 初始化四个fenw数组，长度 n+1，全部填充0n(BigInt 0)
    let fenw0_odd = new Array(n + 1).fill(0n);   // 全奇数序列
    let fenw1_even = new Array(n + 1).fill(0n);  // 全偶数序列
    let fenw2_even = new Array(n + 1).fill(0n);  // 奇偶交替，下一位需为奇数
    let fenw2_odd = new Array(n + 1).fill(0n);   // 奇偶交替，下一位需为偶数

    let ans0 = 0n, ans1 = 0n, ans2 = 0n;

    for (let x = 1; x <= n; x++) {
        let half = Math.floor(x / 2);
        if (x % 2 === 1) { // x 是奇数
            // 查询前半部分奇数序列和奇偶交替序列
            let dp0_val = query(fenw0_odd, half);
            let dp2_val = query(fenw2_even, half);

            // 更新奇数位置的dp值
            update(fenw0_odd, x, 1n + dp0_val);
            update(fenw2_odd, x, 1n + dp2_val);
```

```
            if (x === n) {
                ans0 = dp0_val;
                ans2 = dp2_val;
            }
        } else { // x 是偶数
            // 查询前半部分偶数序列和奇偶交替序列
            let dp1_val = query(fenw1_even, half);
            let dp2_val = query(fenw2_odd, half);

            // 更新偶数位置的dp值
            update(fenw1_even, x, 1n + dp1_val);
            update(fenw2_even, x, 1n + dp2_val);

            if (x === n) {
                ans1 = dp1_val;
                ans2 = dp2_val;
            }
        }
    }

    // 总序列数 = 1 + 三种状态的答案和，转换为字符串输出
    console.log((1n + ans0 + ans1 + ans2).toString());
    rl.close();
});
```

Go

Plain Text

```
package main

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

// 树状数组 更新操作，delta为int64类型
func update(tree []int64, index int, delta int64) {
    n := len(tree) - 1
    for index <= n {
        tree[index] += delta // 更新当前位置值
        index += index & (-index) // 跳到所有包含该位置的父区间
    }
}

// 查询区间和，返回1到index区间和
func query(tree []int64, index int) int64 {
    var res int64 = 0
    for index > 0 {
        res += tree[index] // 累加当前节点的值
        index -= index & (-index) // 跳到上一个区间
    }
    return res
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    n, _ := strconv.Atoi(scanner.Text())

    // 初始化四个树状数组，全部用0填充，长度为n+1
    fenw0_odd := make([]int64, n+1)  // 全奇数序列
    fenw1_even := make([]int64, n+1) // 全偶数序列
    fenw2_even := make([]int64, n+1) // 奇偶交替，下一位需为奇数
    fenw2_odd := make([]int64, n+1)  // 奇偶交替，下一位需为偶数

    var ans0, ans1, ans2 int64

    // 遍历1到n
    for x := 1; x <= n; x++ {
        half := x / 2
        if x%2 == 1 { // 奇数位置
```

```
            dp0_val := query(fenw0_odd, half)  // 查询前半长度奇数序列
            dp2_val := query(fenw2_even, half) // 查询前半长度奇偶交替序列

            update(fenw0_odd, x, 1+dp0_val) // 更新奇数序列
            update(fenw2_odd, x, 1+dp2_val) // 更新奇偶交替序列

            if x == n {
                ans0 = dp0_val
                ans2 = dp2_val
            }
        } else { // 偶数位置
            dp1_val := query(fenw1_even, half) // 查询前半长度偶数序列
            dp2_val := query(fenw2_odd, half)  // 查询前半长度奇偶交替序列

            update(fenw1_even, x, 1+dp1_val)  // 更新偶数序列
            update(fenw2_even, x, 1+dp2_val)  // 更新奇偶交替序列

            if x == n {
                ans1 = dp1_val
                ans2 = dp2_val
            }
        }
    }

    // 输出结果：1 + 三个状态和
    fmt.Println(1 + ans0 + ans1 + ans2)
}
```

> 来自：华为OD机试 - 构建数列 (2025 B卷 100分)_题目描述构造数列,第一个数为n,后面的数不大于前一个的一半,数列奇偶相间或许全为-CSDN博客

华为OD 机考 - 最少交付时间 (2025 B卷 100分)_华为机考题2025-CSDN博客

最少交付时间

华为OD机试真题目录: 点击去查看

> `华为od 2025 B卷 200分题型`

题目描述

项目组共有N个开发人员，项目经理接到了M个独立的需求，每个需求的工作量不同，且每个需求只能由一个开发人员独立完成，不能多人合作。
假定各个需求直接无任何先后依赖关系，请设计算法帮助项目经理进行工作安排，使整个项目能用最少的时间交付。

输入描述

第一行输入为M个需求的工作量，单位为天，用逗号隔开。 例如：X1 X2 X3 ... Xm 表示共有M个需求，每个需求的工作量分别为X1天，X2天...Xm天。其中0<M<30；0<Xm<200 第二行输入为项目组人员数量N 例如： 5 表示共有5名员工，其中 `0<N<10` .

输出描述

最快完成所有工作的天数 例如： 25 表示最短需要25天能完成所有工作。

用例1

输入

Plain Text

```
8 8 8 8 8 1
3
```

输出

Plain Text

```
16
```

用例2

输入

```
10 10 10 10
2
```

输出

```
20
```

说明

> 两名员工各分配两项任务，最大负载为20天。

题解

思路： `二分 + DFS剪枝` 实现

- 使用二分尝试设置完成的天数(days)，初始设置 `left = 0, right = sum(需求时间)` . 每次判断 `mid = (left + right) >> 1` 进行尝试，使用 `DFS` 判断给定天数下是否能满足条件。然后更新 `left right` , 直接 `left == right`
  - 判断是否满足条件下逻辑大概是，有n个员工，设置n个容量为days的容器，每个需求的天数看作重量，尝试将所有需求放入这些容器内，在不能超过容器的容量情况下，所有需求是否能全部放入。
  - 满足情况下设置 `right = mid`
  - 不满足情况下设置 `left = mid + 1`
- 终止二分的条件为 `left == right` , 此时的left就是题目所需要的最少交付时间。

C++

Plain Text

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


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

// 判断是否满足要求
bool judge(vector<int>& ans, int personNum, int target, int index, vector<int>& load) {
    if (index < 0) {
        return true;
    }
    for (int i = 0; i < personNum; i++) {
        // 剪枝
        if (i > 0 && load[i] == load[i-1]) {
            continue;
        }

        // 递归回溯
        if (load[i] + ans[index] <= target) {
            load[i] += ans[index];
            if (judge(ans, personNum, target, index - 1, load)) {
                return true;
            }
```

```
                load[i] -= ans[index];
            }
            //剪枝：如果当前人是第一个任务都没分配过的，直接跳出，避免不必要尝试
            if (load[i] == 0) {
                break;
            }
        }
        return false;
    }

    // 二分 + 递归回溯解决
    int solve(vector<int>& ans, int personNum) {
        int sum = 0;
        int n = ans.size();
        for (int i = 0; i < n; i++) {
            sum += ans[i];
        }
        // 二分
        int left = 0, right = sum;
        while (left < right) {
            int target = (left + right) >> 1;
            vector<int> load(personNum, 0);
            if (judge(ans, personNum, target, n -1, load)) {
                right = target;
            } else {
                left = target + 1;
            }
        }
        return left;
    }

    int main() {
        string s;
        getline(cin, s);
        int personNum;
        cin >> personNum;

        vector<string> tmp = split(s, ",");
        int n = tmp.size();
        vector<int> ans(n, 0);
        for (int i = 0; i < n; i++) {
            ans[i] = stoi(tmp[i]);
        }
        // 从小到大排序
        sort(ans.begin(), ans.end());
        int res = solve(ans, personNum);
        cout << res;
        return 0;
```

```
93    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 判断是否满足要求
    public static boolean judge(List<Integer> ans, int personNum, int target, int index, int[] load) {
        if (index < 0) {
            return true;
        }
        for (int i = 0; i < personNum; i++) {
            // 剪枝
            if (i > 0 && load[i] == load[i - 1]) {
                continue;
            }

            // 递归回溯
            if (load[i] + ans.get(index) <= target) {
                load[i] += ans.get(index);
                if (judge(ans, personNum, target, index - 1, load)) {
                    return true;
                }
                load[i] -= ans.get(index);
            }
            // 剪枝：如果当前人是第一个任务都没分配过的，直接跳出，避免不必要尝试
            if (load[i] == 0) {
                break;
            }
        }
        return false;
    }

    // 二分 + 递归回溯
    public static int solve(List<Integer> ans, int personNum) {
        int sum = ans.stream().mapToInt(Integer::intValue).sum();
        int left = 0, right = sum;

        while (left < right) {
            int target = (left + right) / 2;
            int[] load = new int[personNum];

            if (judge(ans, personNum, target, ans.size() - 1, load)) {
                right = target;
            } else {
                left = target + 1;
            }
```

```
            }
            return left;
        }

        public static void main(String[] args) {
            Scanner scanner = new Scanner(System.in);
            String s = scanner.nextLine();
            int personNum = scanner.nextInt();
            scanner.close();

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

            // 从小到大排序
            Collections.sort(ans);
            int res = solve(ans, personNum);
            System.out.println(res);
        }
    }
```

Python

Plain Text

```
def judge(ans, person_num, target, index, load):
    if index < 0:
        return True
    for i in range(person_num):
        # 剪枝
        if i > 0 and load[i] == load[i - 1]:
            continue
        if load[i] + ans[index] <= target:
            load[i] += ans[index]
            if judge(ans, person_num, target, index - 1, load):
                return True
            load[i] -= ans[index]
        # 剪枝
        if load[i] == 0:
            break
    return False

def solve(ans, person_num):
    left, right = 0, sum(ans)
    while left < right:
        target = (left + right) // 2
        load = [0] * person_num
        if judge(ans, person_num, target, len(ans) - 1, load):
            right = target
        else:
            left = target + 1
    return left

if __name__ == "__main__":
    s = input().strip()
    person_num = int(input().strip())

    ans = list(map(int, s.split(",")))
    ans.sort()
    res = solve(ans, person_num)
    print(res)
```

JavaScript

Plain Text

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

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

// 判断是否满足要求
function judge(ans, personNum, target, index, load) {
    if (index < 0) return true;
    for (let i = 0; i < personNum; i++) {
        // 剪枝
        if (i > 0 && load[i] === load[i - 1]) continue;

        if (load[i] + ans[index] <= target) {
            load[i] += ans[index];
            if (judge(ans, personNum, target, index - 1, load)) {
                return true;
            }
            load[i] -= ans[index];
        }
        // 剪枝
        if (load[i] === 0) break;
    }
    return false;
}

// 二分 + 递归回溯
function solve(ans, personNum) {
    let left = 0, right = ans.reduce((a, b) => a + b, 0);
    while (left < right) {
        let target = Math.floor((left + right) / 2);
        let load = new Array(personNum).fill(0);
        if (judge(ans, personNum, target, ans.length - 1, load)) {
            right = target;
        } else {
            left = target + 1;
        }
    }
    return left;
}

// 读取输入
let input = [];
rl.on("line", (line) => {
```

```
    input.push(line.trim());
    if (input.length === 2) {
        rl.close();
    }
});

rl.on("close", () => {
    let ans = input[0].split(",").map(Number);
    let personNum = parseInt(input[1]);

    ans.sort((a, b) => a - b);
    console.log(solve(ans, personNum));
});
```

Go

```
Plain Text
```

```
package main

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

// 判断是否满足要求
func judge(ans []int, personNum int, target int, index int, load []int) bo
ol {
  if index < 0 {
    return true
  }
  for i := 0; i < personNum; i++ {
    // 剪枝
    if i > 0 && load[i] == load[i-1] {
      continue
    }
    if load[i]+ans[index] <= target {
      load[i] += ans[index]
      if judge(ans, personNum, target, index-1, load) {
        return true
      }
      load[i] -= ans[index]
    }
    // 剪枝
    if load[i] == 0 {
      break
    }
  }
  return false
}

// 二分 + 递归回溯
func solve(ans []int, personNum int) int {
  left, right := 0, 0
  for _, v := range ans {
    right += v
  }

  for left < right {
```

```
      target := (left + right) / 2
      load := make([]int, personNum)

      if judge(ans, personNum, target, len(ans)-1, load) {
        right = target
      } else {
        left = target + 1
      }
    }
    return left
}

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    scanner.Scan()
    s := scanner.Text()
    scanner.Scan()
    personNum, _ := strconv.Atoi(scanner.Text())

    nums := strings.Split(s, ",")
    ans := make([]int, len(nums))
    for i, num := range nums {
      ans[i], _ = strconv.Atoi(num)
    }

    // 从小到大排序
    sort.Ints(ans)
    fmt.Println(solve(ans, personNum))
}
```

> 来自: 华为OD 机考 – 最少交付时间 (2025 B卷 100分)_华为机考题2025-CSDN博客

华为OD 机考 - AI面板识别 (2025 B卷 100分)-CSDN博客

AI面板识别

华为OD机试真题目录: 点击去查看

> 华为OD 2025B 卷 100分题型

题目描述

AI识别到面板上有N（1 ≤ N ≤ 100）个指示灯，灯大小一样，任意两个之间无重叠。
由于AI识别误差，每次识别到的指示灯位置可能有差异，以4个坐标值描述AI识别的指示灯的大小和位置(左上角x1,y1，右下角x2,y2)，
请输出先行后列排序的指示灯的编号，排序规则：

1. 每次在尚未排序的灯中挑选最高的灯作为的基准灯，
2. 找出和基准灯属于同一行所有的灯进行排序。两个灯高低偏差不超过灯半径算同一行（即两个灯坐标的差 ≤ 灯高度的一半）。

输入描述

第一行为N，表示灯的个数
接下来N行，每行为1个灯的坐标信息，格式为：

> 编号 x1 y1 x2 y2

- 编号全局唯一
- 1<= 编号<=100
- 0<=x1<x2<=1000
- 0<=y1<y2<=1000

输出描述

排序后的编号列表，编号之间以空格分隔。

用例1

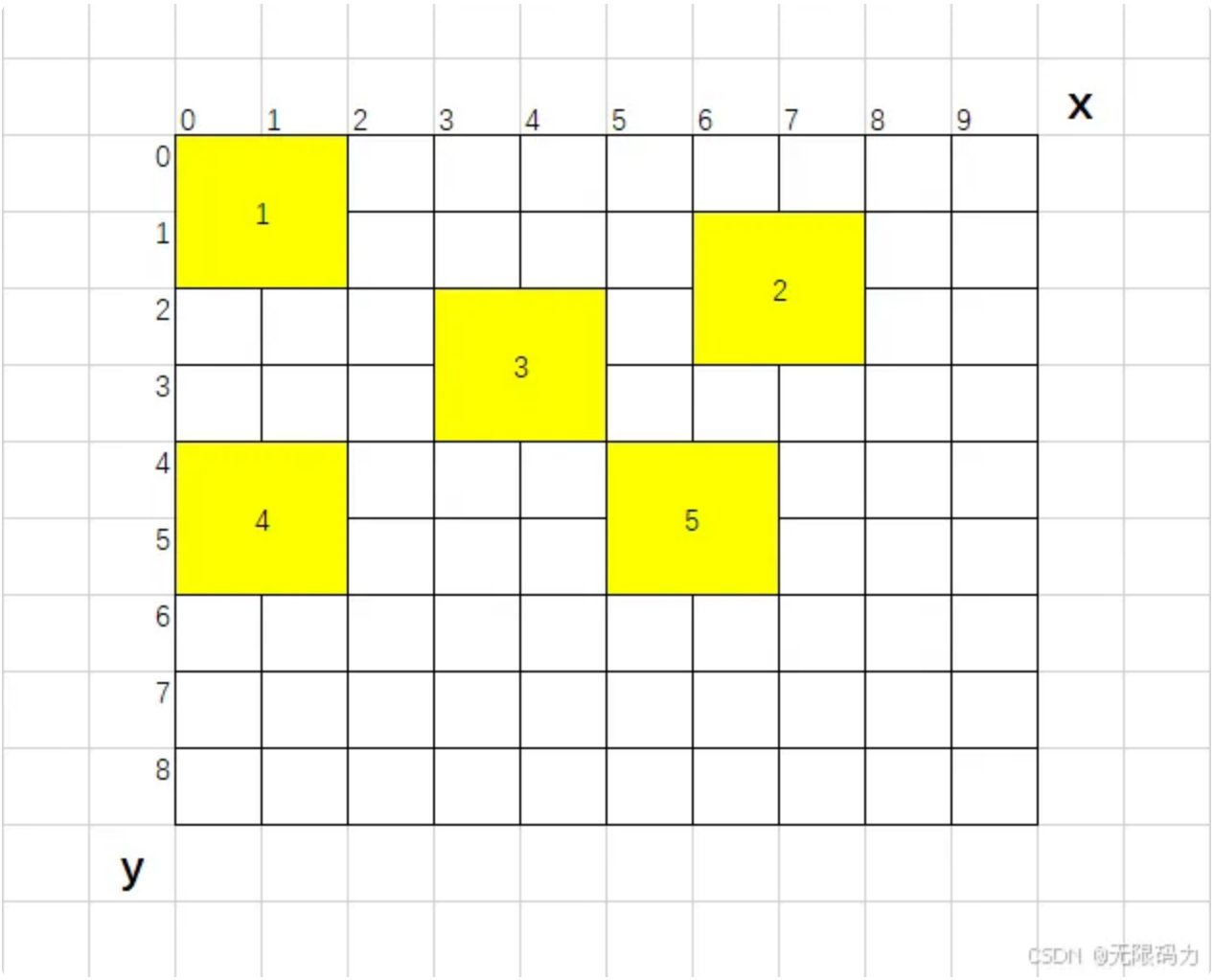
输入

```
5
1 0 0 2 2
2 6 1 8 3
3 3 2 5 4
5 5 4 7 6
4 0 4 2 6
```

输出

```
1 2 3 4 5
```

说明



题解

思路： `模拟 + 排序`

1. 接收所有输入的灯的坐标信息，接下来按照y进行排序(升序)，可以使用 `自定义排序或者优先队列` ，下面代码使用优先队列实现。
2. 循环遍历，根据条件(两个灯高低偏差不超过灯半径算同一行) 处理一行数据。
    a. 如果当前行只有一个数据，那么可以直接输出当前坐标的id。
    b. 如果有 `其它灯与当前遍历灯的y差值 <= 灯的半径` ，则说明这一行存在多个灯，将 多个灯需要按照x进行排序，排序之后按照顺序进行输出对应id即可。

C++

Plain Text

```
#include <cerrno>
#include <iostream>
#include<queue>
#include <vector>
#include<algorithm>
using namespace std;

struct Item {
    int id;
    int x;
    int y;
    bool operator<(const Item& other) const {
        return y > other.y;
    }

};

bool cmp(Item i1, Item i2) {
  return i1.x < i2.x;
}


int main() {
    int n;
    cin >> n;
    // 使用优先队列进行y的升序排序
    priority_queue<Item> pq;
    double height;
    for (int i = 0; i < n; i++) {
        int id;
        int x1, x2, y1, y2;
        cin >> id >> x1 >> y1 >> x2 >> y2;
        pq.push({id, x1, y1});
        height =  (y2 - y1 * 1.0) / 2;
    }

    while (!pq.empty()) {
        Item first = pq.top();
        pq.pop();
        // 当前行只有一个元素
        if (pq.empty() || pq.top().y - first.y > height) {
           cout << first.id << " ";
        } else {
          vector<Item> ans;
          ans.push_back(first);
```

```
            // 迭代获取所有行
            while (!pq.empty() && pq.top().y - first.y <= height) {
               Item tmp = pq.top();
               pq.pop();
               ans.push_back(tmp);
            }
            // 按照行进行排序
            sort(ans.begin(), ans.end(), cmp);
            for (int i = 0; i < ans.size(); i++) {
              cout << ans[i].id << " ";
            }
          }
      }

      return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

class Item implements Comparable<Item> {
    int id;
    int x;
    int y;

    Item(int id, int x, int y) {
        this.id = id;
        this.x = x;
        this.y = y;
    }

    @Override
    public int compareTo(Item other) {
        // 按 y 升序排序
        return Integer.compare(this.y, other.y);
    }
}

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        PriorityQueue<Item> pq = new PriorityQueue<>();
        double height = 0;

        for (int i = 0; i < n; i++) {
            int id = scanner.nextInt();
            int x1 = scanner.nextInt();
            int y1 = scanner.nextInt();
            int x2 = scanner.nextInt();
            int y2 = scanner.nextInt();
            pq.add(new Item(id, x1, y1));
            height = (y2 - y1) * 0.5;
        }

        while (!pq.isEmpty()) {
            Item first = pq.poll();
            // 当前行只有一个元素
            if (pq.isEmpty() || pq.peek().y - first.y > height) {
                System.out.print(first.id + " ");
            } else {
                List<Item> ans = new ArrayList<>();
```

```
                    ans.add(first);

                    // 迭代获取所有行的元素
                    while (!pq.isEmpty() && pq.peek().y - first.y <= height) {
                        ans.add(pq.poll());
                    }

                    // 按 x 升序排序
                    ans.sort(Comparator.comparingInt(o -> o.x));

                    for (Item item : ans) {
                        System.out.print(item.id + " ");
                    }
                }
            }

            scanner.close();
        }
    }
```

Python

Plain Text

```
import heapq

class Item:
    def __init__(self, id, x, y):
        self.id = id
        self.x = x
        self.y = y

    def __lt__(self, other):
        # 按 y 升序排序
        return self.y < other.y

def main():
    n = int(input())
    pq = []  # 使用 heapq 实现优先队列
    height = 0

    for _ in range(n):
        id, x1, y1, x2, y2 = map(int, input().split())
        heapq.heappush(pq, Item(id, x1, y1))
        height = (y2 - y1) * 0.5

    while pq:
        first = heapq.heappop(pq)
        # 当前行只有一个元素
        if not pq or pq[0].y - first.y > height:
            print(first.id, end=" ")
        else:
            ans = [first]

            # 迭代获取所有行的元素
            while pq and pq[0].y - first.y <= height:
                ans.append(heapq.heappop(pq))

            # 按 x 升序排序
            ans.sort(key=lambda item: item.x)

            for item in ans:
                print(item.id, end=" ")

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

JavaScript

Plain Text

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

class Item {
    constructor(id, x, y) {
        this.id = id;
        this.x = x;
        this.y = y;
    }
}

function main() {
    let n; // 输入的数量
    const pq = []; // 存储所有的项目
    let height = 0; // 高度变量
    let inputLines = []; // 用于存储所有输入

    rl.on('line', (input) => {
        inputLines.push(input); // 收集每一行输入

        // 当接收到 n + 1 行时，停止输入并开始处理
        if (inputLines.length === 1) {
            n = parseInt(inputLines[0]); // 第一行是数量
        } else if (inputLines.length === n + 1) {
            // 解析输入数据
            for (let i = 1; i <= n; i++) {
                const [id, x1, y1, x2, y2] = inputLines[i].split(' ').map
(Number);
                pq.push(new Item(id, x1, y1));
                height = (y2 - y1) * 0.5; // 计算高度
            }

            // 按 y 升序排序
            pq.sort((a, b) => a.y - b.y);

            // 开始处理项目
            while (pq.length > 0) {
                const first = pq.shift(); // 取出队列中的第一个元素， 基准元素

                // 当前行只有一个元素
                if (pq.length === 0 || pq[0].y - first.y > height) {
                    process.stdout.write(first.id + " ");
```

```
                } else {
                    const ans = [first]; // 存储当前行的所有元素

                    // 收集当前行的所有元素
                    while (pq.length > 0 && pq[0].y - first.y <= height) {
                        ans.push(pq.shift());
                    }

                    // 按 x 升序排序
                    ans.sort((a, b) => a.x - b.x);

                    // 输出当前行的所有元素 ID
                    for (let item of ans) {
                        process.stdout.write(item.id + " ");
                    }
                }
            }

            rl.close(); // 所有数据处理完毕后，关闭输入流
        }
    });
}

main();
```

Go

Plain Text

```
package main

import (
  "fmt"
  "sort"
)

type Item struct {
  id int
  x  int
  y  int
}

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

  items := make([]Item, n)
  var height float64

  // 输入数据
  for i := 0; i < n; i++ {
    var id, x1, x2, y1, y2 int
    fmt.Scan(&id, &x1, &y1, &x2, &y2)
    height = float64(y2-y1) / 2
    items[i] = Item{id: id, x: x1, y: y1}
  }

  // 按照 y 升序排序
  sort.Slice(items, func(i, j int) bool {
    return items[i].y < items[j].y
  })

  for len(items) > 0 {
    first := items[0]
    items = items[1:]

    if len(items) == 0 || float64(items[0].y-first.y) > height {
      // 当前行只有一个元素
      fmt.Printf("%d ", first.id)
    } else {
      // 收集同一行的元素
      group := []Item{first}
      for len(items) > 0 && float64(items[0].y-first.y) <= height {
        group = append(group, items[0])
```

```
            items = items[1:]
        }

        // 按 x 排序
        sort.Slice(group, func(i, j int) bool {
            return group[i].x < group[j].x
        })

        for _, item := range group {
            fmt.Printf("%d ", item.id)
        }
    }
  }
}
```

来自: 华为OD 机考 – AI面板识别 (2025 B卷 100分)–CSDN博客