

# t0729

---

[华为OD机考 2025C卷 - 微服务的集成测试 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD 机考 2025C卷 - CPU调度策略 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025C卷 - 提取字符串中最长数学表达式 \(C++ & Python & JAVA & JS & GO\)\\_华为od机试 - 提取字符串中的最长合法简单数学表达式-CSDN博客](#)

[华为OD上机考试 2025C卷 - 斗地主之顺子 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD 机考 2025C卷 - 数字序列比大小 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 内存资源分配-CSDN博客](#)

[华为OD机考 2025C卷 - 二维伞的雨滴效应 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD 机试 2025 B卷 - 字符串加密 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025 C卷 - 自然数分解 / 用连续自然数之和来表达整数 \(C++ & Pyth](#)

[华为OD机试 2025c卷 - 寻找最优的路测线路 \(C++ & Python & JAVA & J](#)

[华为OD 机考 2025C卷 - 数字序列比大小 \(C++ & Python & JAVA & JS](#)

[华为OD机考 2025C卷 - 分月饼 \(C++ & Python & JAVA & JS & GO](#)

[\[来自浏览器插件\]无标题](#)

[华为OD机考 2025 C卷 - 分苹果 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025C卷 - 5G网络建设 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

# 华为OD机考 2025C卷 - 微服务的集成测试 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 微服务的集成测试

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

华为OD机试 2025C卷 100分题型

### 题目描述

现在有n个容器服务，服务的启动可能有一定的依赖性（有些服务启动没有依赖），其次服务自身启动加载会消耗一些时间。

给你一个  $n \times n$  的二维矩阵useTime，其中

- `useTime[i][i] = 10` 表示服务i自身启动加载需要消耗10s
- `useTime[i][j] = 1` 表示服务i启动依赖服务j启动完成
- `useTime[i][k] = 0` 表示服务i启动不依赖服务k

其实  $0 \leq i, j, k < n$ 。

服务之间启动没有**循环依赖**（不会出现环），若想对任意一个服务i进行集成测试（服务i自身也需要加载），求最少需要等待多少时间。

### 输入描述

第一行输入服务总量 n，

之后的 n 行表示服务启动的依赖关系以及自身启动加载耗时

最后输入 k 表示计算需要等待多少时间后可以对服务 k 进行集成测试

其中  $1 \leq k \leq n$ ,  $1 \leq n \leq 100$

### 输出描述

最少需要等待多少时间(s)后可以对服务 k 进行集成测试

### 用例1

#### 输入

```
1 3  
2 5 0 0  
3 1 5 0  
4 0 1 5  
5 3
```

Plain Text |

## 输出

```
1 15
```

Plain Text |

## 说明

服务3启动依赖服务2，服务2启动依赖服务1，由于服务1，2，3自身加载需要消耗5s，所以 $5+5+5=15$ ，需要等待15s后可以对服务3进行集成测试

## 用例2

### 输入

```
1 3  
2 5 0 0  
3 1 10 1  
4 1 0 11  
5 2
```

Plain Text |

## 输出

```
1 26
```

Plain Text |

## 说明

服务2启动依赖服务1和服务3，服务3启动需要依赖服务1，服务1，2，3自身加载需要消耗5s，10s，11s，所以 $5+10+11=26$ s，需要等待26s后可以对服务2进行集成测试。

## 用例3

## 输入

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

Plain Text |

## 输出

```
1 12
```

Plain Text |

## 说明

服务3启动依赖服务1和服务2，服务4启动需要依赖服务1, 2, 3，服务1, 2, 3自身加载需要消耗2s,3s,4s,5s，所以 $3+4+5=12s$ （因为服务1和服务2可以同时启动），要等待12s后可以对服务4进行集成测试。

## 用例4

### 输入

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

Plain Text |

### 输出

```
1 11
```

Plain Text |

## 说明

服务3启动依赖服务1和服务2，服务4启动需要依赖服务1, 2，服务5启动需要依赖服务3, 5，服务1, 2, 3, 4, 5自身加载需要消耗1s,2s,3s,4s,5s，所以 $2+4+5=11s$ （因为服务1和服务2可以同时启动，服务3和服务4可以同时启动），要等待11s后可以对服务5进行集成测试。

## 题解

思路： DFS + 哈希数据结构

- 使用 哈希表 记录某个服务依赖的所有服务的列表。
- 指定启动  $k$  服务器时，递归启动它所有依赖的服务，递归结束条件为当前服务不存在依赖服务，每个服务器启动所需时间为 自身启动时间 +  $\max(\text{所有依赖服务启动时间})$ ，累加过程中的时间就为  $k$  服务启动的时间。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<map>
8 using namespace std;
9
10 // 递归遍历
11 int DFS(vector<vector<int>>& grid, map<int, vector<int>>& dependMap, int index) {
12     int res = 0;
13     vector<int> dependIds = dependMap[index];
14     if (!dependIds.empty()) {
15         for (int id : dependIds) {
16             res = max(res, DFS(grid, dependMap, id));
17         }
18     }
19     res += grid[index][index];
20     return res;
21 }
22
23
24
25 int main() {
26     int n;
27     cin >> n;
28     vector<vector<int>> grid(n, vector<int>(n));
29     // 记录指定容器依赖的服务
30     map<int, vector<int>> dependMap;
31     for (int i = 0; i < n; i++) {
32         for (int j = 0; j < n; j++) {
33             cin >> grid[i][j];
34             if (i != j && grid[i][j] == 1) {
35                 dependMap[i].push_back(j);
36             }
37         }
38     }
39     int k;
40     cin >> k;
41     int res = DFS(grid, dependMap, k-1);
42     cout << res;
43     return 0;
44 }
```

## JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // 递归遍历计算启动时间
5     static int DFS(int[][] grid, Map<Integer, List<Integer>> dependMap, int index) {
6         int res = 0;
7         List<Integer> dependIds = dependMap.getOrDefault(index, new ArrayList<>());
8         for (int id : dependIds) {
9             res = Math.max(res, DFS(grid, dependMap, id));
10        }
11        res += grid[index][index]; // 加上当前服务自身启动时间
12        return res;
13    }
14
15    public static void main(String[] args) {
16        Scanner scanner = new Scanner(System.in);
17        int n = scanner.nextInt();
18        int[][] grid = new int[n][n];
19
20        // 记录服务依赖关系
21        Map<Integer, List<Integer>> dependMap = new HashMap<>();
22
23        // 读取矩阵
24        for (int i = 0; i < n; i++) {
25            for (int j = 0; j < n; j++) {
26                grid[i][j] = scanner.nextInt();
27                if (i != j && grid[i][j] == 1) {
28                    dependMap.computeIfAbsent(i, k -> new ArrayList<>()).add(j);
29                }
30            }
31        }
32        int k = scanner.nextInt();
33        System.out.println(DFS(grid, dependMap, k - 1)); // 计算并输出结果
34    }
35 }
```

## Python

```
1 def dfs(grid, depend_map, index):
2     """ 递归计算服务启动时间 """
3     res = 0
4     for dep in depend_map.get(index, []):
5         res = max(res, dfs(grid, depend_map, dep))
6     return res + grid[index][index] # 加上自身启动时间
7
8 def main():
9     n = int(input())
10    grid = [list(map(int, input().split())) for _ in range(n)]
11
12    # 记录服务依赖关系
13    depend_map = {}
14    for i in range(n):
15        for j in range(n):
16            if i != j and grid[i][j] == 1:
17                depend_map.setdefault(i, []).append(j)
18
19    k = int(input())
20    print(dfs(grid, depend_map, k - 1)) # 计算并输出结果
21
22 if __name__ == "__main__":
23     main()
```

## JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9 rl.on("line", function (line) {
10     inputLines.push(line);
11 }).on("close", function () {
12     const n = parseInt(inputLines[0]);
13     const grid = [];
14     const dependMap = new Map();
15
16     for (let i = 1; i <= n; i++) {
17         grid.push(inputLines[i].split(" ").map(Number));
18         for (let j = 0; j < n; j++) {
19             if (i - 1 !== j && grid[i - 1][j] === 1) {
20                 if (!dependMap.has(i - 1)) {
21                     dependMap.set(i - 1, []);
22                 }
23                 dependMap.get(i - 1).push(j);
24             }
25         }
26     }
27
28     const k = parseInt(inputLines[n + 1]);
29
30     function dfs(grid, dependMap, index) {
31         let res = 0;
32         if (dependMap.has(index)) {
33             for (let dep of dependMap.get(index)) {
34                 res = Math.max(res, dfs(grid, dependMap, dep));
35             }
36         }
37         return res + grid[index][index];
38     }
39
40     console.log(dfs(grid, dependMap, k - 1)); // 输出结果
41 });
```

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 递归计算服务启动时间
12 func dfs(grid [][]int, dependMap map[int][]int, index int) int {
13     res := 0
14     for _, dep := range dependMap[index] {
15         res = max(res, dfs(grid, dependMap, dep))
16     }
17     return res + grid[index][index]
18 }
19
20 // 获取最大值
21 func max(a, b int) int {
22     if a > b {
23         return a
24     }
25     return b
26 }
27
28 func main() {
29     scanner := bufio.NewScanner(os.Stdin)
30     scanner.Scan()
31     n, _ := strconv.Atoi(scanner.Text())
32
33     grid := make([][]int, n)
34     dependMap := make(map[int][]int)
35
36     // 读取输入
37     for i := 0; i < n; i++ {
38         scanner.Scan()
39         row := strings.Split(scanner.Text(), " ")
40         grid[i] = make([]int, n)
41         for j := 0; j < n; j++ {
42             grid[i][j], _ = strconv.Atoi(row[j])
43             if i != j && grid[i][j] == 1 {
44                 dependMap[i] = append(dependMap[i], j)
45             }
46         }
47     }
48 }
```

```
46     }
47 }
48
49 scanner.Scan()
50 k, _ := strconv.Atoi(scanner.Text())
51
52 // 计算并输出结果
53 fmt.Println(dfs(grid, dependMap, k-1))
54 }
```

| 来自: 华为OD机考 2025C卷 – 微服务的集成测试 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机考2025C卷 - CPU调度策略(C++ & Python & Java & JS & Go)-CSDN博客

## CPU调度策略

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

华为OD机试2025C卷 200分题型

### 题目描述

现有一个CPU和一些任务需要处理，已提前获知每个任务的任务ID、优先级、所需执行时间和到达时间。CPU同时只能运行一个任务，请编写一个任务调度程序，采用“可抢占优先权调度”[调度算法](#)进行任务调度，规则如下：

- 如果一个任务到来时，CPU是空闲的，则CPU可以运行该任务直到任务执行完毕。但是如果运行中有一个更高优先级的任务到来，则CPU必须暂停当前任务去运行这个优先级更高的任务；
- 如果一个任务到来时，CPU正在运行一个比他优先级更高的任务时，新到达的任务必须等待；
- 当CPU空闲时，如果还有任务在等待，CPU会从这些任务中选择一个优先级最高的任务执行，相同优先级的任务选择到达时间最早的任务。

### 输入描述

输入有若干行，每一行有四个数字（均小于 $10^8$ ），分别为任务ID，任务优先级，执行时间和到达时间。每个任务的任务ID不同，优先级数字越大优先级越高，并且相同优先级的任务不会同时到达。输入的任务已按照到达时间从小到大排序，并且保证在任何时间，处于等待的任务不超过10000个。

### 输出描述

按照任务执行结束的顺序，输出每个任务的任务ID和对应的结束时间。

### 用例1

#### 输入

```
1 1 3 5 1
2 2 1 5 10
3 3 2 7 12
4 4 3 2 20
5 5 4 9 21
6 6 4 2 22
```

Plain Text

# 输出

```
Plain Text |  
1 1 6  
2 3 19  
3 5 30  
4 6 32  
5 4 33  
6 2 35
```

## 题解

思路： 模拟 + 优先队列

1. 结合时间线 + 优先队列来处理这个问题。
2. 定义 优先队列 排序规则按照题目要求 优先级高的任务位置堆顶，优先级相同的到达时间早的位于堆顶 规则进行排序，用优先队列来 处理当前时间和下一个任务到达之前哪个优先任务执行 的问题。
3. 记录 当前时间 主要是计算下一个任务到达时间之间有多少时间可供当前队列中的任务执行。
4. 讲一下基本逻辑：
  - a. 假设当前时间为 `currentTime`，下一个任务到达时间为 `nextArriveTime`，这中间 `idleTime = nextArriveTime - currentTime` 这段时间，就是可供优先队列中优先级最高任务执行时间。假设当前优先级最高任务为 `currentTask` 处理情况如下： 下一个任务到达之前不用考虑什么下一个任务优先级问题
    - i. `currentTask` 的剩余执行时间小于等于 `idleTime`，说明 `currentTask` 可以在这个时间段内执行结束且空闲时间还有剩余。更新 `idleTime -= currentTask 执行时间`，并更新 `currentTask` 为下一个优先级最高的任务。 注意：这是一个循环迭代的过程
    - ii. `currentTask` 的剩余执行时间大于 `idleTime`，不能在这个空闲结束，但是可以减少 `currentTask` 的剩余执行时间 `currentTask 剩余执行时间 -= idleTime`。
  - b. 当 `currentTime = nextArriveTime` 时，将下一个任务压入堆中即可。重复1 2逻辑即可，所有任务都压入优先队列结束第一阶段处理。 抢占问题 由优先队列统一去进行处理
  - c. 当前所有任务都已经压入优先队列，要不在过程已经完成，要不然任务都存在优先队列。统一处理优先队列剩余任务即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 #include<queue>
10 using namespace std;
11
12 struct Task{
13     int id;
14     int priority;
15     int executeTime;
16     int arriveTime;
17
18     Task(int id, int priority, int executeTime, int arriveTime):id(id),priority(priority),executeTime(executeTime),arriveTime(arriveTime) {}
19 };
20
21
22
23 // 自定义比较函数:
24 struct CompareTask {
25     bool operator()(const Task& a, const Task& b) const {
26         if (a.priority == b.priority) {
27             return a.arriveTime > b.arriveTime; // 到达时间 小的优先
28         }
29         return a.priority < b.priority; // priority 大的优先
30     }
31 };
32
33
34 // 通用 切割函数 函数 将字符串str根据delimiter进行切割
35 vector<int> split(const string& str, const string& delimiter) {
36     vector<int> result;
37     size_t start = 0;
38     size_t end = str.find(delimiter);
39     while (end != string::npos) {
40         result.push_back(stoi(str.substr(start, end - start)));
41         start = end + delimiter.length();
42         end = str.find(delimiter, start);
43     }
44     // 添加最后一个部分
```

```

45     result.push_back(stoi(str.substr(start)));
46     return result;
47 }
48
49 int main() {
50     vector<Task> tasks;
51     string input;
52     while (getline(cin, input)) {
53         if (input.empty()) {
54             break;
55         }
56         vector<int> arr = split(input, " ");
57         tasks.push_back({arr[0], arr[1], arr[2], arr[3]});
58     }
59     // 使用优先队列
60     priority_queue<Task, vector<Task>, CompareTask> pq;
61     int n = tasks.size();
62     int index = 0;
63     pq.push(tasks[index]);
64     index++;
65
66     int currentTime = pq.top().arriveTime;
67     while (index < n) {
68         Task currentTask = pq.top();
69         pq.pop();
70         Task nextTask = tasks[index];
71         index++;
72
73         int currentTaskEndTime = currentTime + currentTask.executeTime;
74         // 当前任务执行结束时间 大于下一个任务到达时间，更新剩余执行时间。优先级抢占
75         等问题统一让优先队列处理
76         if (currentTaskEndTime > nextTask.arriveTime) {
77             currentTask.executeTime -= nextTask.arriveTime - currentTime;
78             currentTime = nextTask.arriveTime;
79             pq.push(currentTask);
80         // 当前栈顶元素可以执行完成
81     } else {
82         cout << currentTask.id << " " << currentTaskEndTime << endl;
83         currentTime = currentTaskEndTime;
84         // 当前执行任务执行完成时间和下一个任务到达时间之前空闲时间，此时优先队
85         列中存在的任务就是等待任务，让其中优先级最高执行，充分利用空闲时间
86         if (nextTask.arriveTime > currentTime) {
87             while (!pq.empty()) {
88                 Task idleTask = pq.top();
89                 pq.pop();
90                 int idleTaskEndTime = currentTime + idleTask.execute
91                 Time;
92             // 同上

```

```
90                     if (idleTaskEndTime > nextTask.arriveTime) {
91                         idleTask.executeTime -= nextTask.arriveTime - cur-
92                         rentTime;
93                         pq.push(idleTask);
94                         break;
95                     // 同上
96                     } else {
97                         cout << idleTask.id << " " << idleTaskEndTime <<
98                         endl;
99                         currentTime = idleTaskEndTime;
100                        }
101                    }
102                }
103                // 入队
104                pq.push(nextTask);
105            }
106            // 剩余未执行任务都在队列中,不存在抢占问题,直接按照优先队列安排输出即可
107            while (!pq.empty()) {
108                Task currentTask = pq.top();
109                pq.pop();
110                int currentTaskEndTime = currentTime + currentTask.executeTime;
111                cout << currentTask.id << " " << currentTaskEndTime << endl;
112                currentTime = currentTaskEndTime;
113            }
114        }
115    }
```

## JAVA

```
1 import java.util.*;
2
3 class Task {
4     int id, priority, executeTime, arriveTime;
5
6     Task(int id, int priority, int executeTime, int arriveTime) {
7         this.id = id;
8         this.priority = priority;
9         this.executeTime = executeTime;
10        this.arriveTime = arriveTime;
11    }
12 }
13
14 // 自定义比较器
15 class CompareTask implements Comparator<Task> {
16     public int compare(Task a, Task b) {
17         if (a.priority == b.priority) {
18             return Integer.compare(a.arriveTime, b.arriveTime); // 到达时间
19             小的优先
20         }
21         return Integer.compare(b.priority, a.priority); // 优先级大的优先
22     }
23 }
24
25 public class Main {
26     public static void main(String[] args) {
27         Scanner sc = new Scanner(System.in);
28         List<Task> tasks = new ArrayList<>();
29         // 接收输入
30         while (sc.hasNextLine()) {
31             String line = sc.nextLine().trim();
32             if (line.isEmpty()) break;
33             String[] parts = line.split(" ");
34             tasks.add(new Task(Integer.parseInt(parts[0]), Integer.parseInt(parts[1]),
35                               Integer.parseInt(parts[2]), Integer.parseInt(parts[3])));
36         }
37
38         PriorityQueue<Task> pq = new PriorityQueue<>(new CompareTask());
39         int n = tasks.size(), index = 0;
40
41         pq.add(tasks.get(index++));
42         int currentTime = pq.peek().arriveTime;
```

```

43     while (index < n) {
44         // 当前优先级最高任务
45         Task currentTask = pq.poll();
46         Task nextTask = tasks.get(index++);
47         int currentTaskEndTime = currentTime + currentTask.executeTim
48         e;
49
50         // 当前任务执行结束时间 > 下一个任务到达时间
51         if (currentTaskEndTime > nextTask.arriveTime) {
52             currentTask.executeTime -= (nextTask.arriveTime - currentT
53             ime);
54             currentTime = nextTask.arriveTime;
55             pq.add(currentTask);
56         } else {
57             // 当前任务可以执行完成
58             System.out.println(currentTask.id + " " + currentTaskEndTi
59             me);
56             currentTime = currentTaskEndTime;
60
61             // 如果当前时间 < 下一个任务到达时间, 执行等待队列中的任务
62             while (!pq.isEmpty() && nextTask.arriveTime > currentTim
63             e) {
64                 Task idleTask = pq.poll();
65                 int idleEnd = currentTime + idleTask.executeTime;
66                 if (idleEnd > nextTask.arriveTime) {
67                     idleTask.executeTime -= (nextTask.arriveTime - cur
68                     rentTime);
69                     pq.add(idleTask);
70                     break;
71                 } else {
72                     System.out.println(idleTask.id + " " + idleEnd);
73                     currentTime = idleEnd;
74                 }
75             }
76             currentTime = nextTask.arriveTime;
77         }
78
79         pq.add(nextTask);
80     }
81
82     // 处理剩余任务
83     while (!pq.isEmpty()) {
84         Task t = pq.poll();
85         int end = currentTime + t.executeTime;
86         System.out.println(t.id + " " + end);
87         currentTime = end;
88     }
89 }

```

## Python

```
1 import sys
2 import heapq
3
4 # 定义任务类
5 class Task:
6     def __init__(self, id, priority, execute_time, arrive_time):
7         self.id = id
8         self.priority = priority
9         self.execute_time = execute_time
10        self.arrive_time = arrive_time
11
12    # 优先队列比较: 优先级高的先来, 若相同则到达时间小的优先
13    def __lt__(self, other):
14        if self.priority == other.priority:
15            return self.arrive_time < other.arrive_time
16        return self.priority > other.priority # priority 大的优先
17
18 tasks = []
19
20 # 读入任务信息
21 for line in sys.stdin:
22     if line.strip() == "":
23         break
24     arr = list(map(int, line.strip().split()))
25     tasks.append(Task(*arr))
26
27 pq = [] # 优先队列
28 index = 0
29 heapq.heappush(pq, tasks[index]) # 将第一个任务入队
30 index += 1
31 current_time = pq[0].arrive_time # 当前时间从第一个任务到达时间开始
32
33 while index < len(tasks):
34     current_task = heapq.heappop(pq) # 当前执行任务
35     next_task = tasks[index] # 下一个任务
36     index += 1
37
38     current_task_end_time = current_time + current_task.execute_time
39
40     # 当前任务执行结束时间 大于下一个任务到达时间,
41     if current_task_end_time > next_task.arrive_time:
42         current_task.execute_time -= (next_task.arrive_time - current_time)
43         current_time = next_task.arrive_time
44         heapq.heappush(pq, current_task) # 任务未执行完, 重新入队
```

```
45     else:
46         # 当前任务可以完全执行完成
47         print(current_task.id, current_task_end_time)
48         current_time = current_task_end_time
49
50     # 若当前任务完成时间 < 下一个任务到达时间, 可空闲执行其他任务
51     while pq and next_task.arrive_time > current_time:
52         idle_task = heapq.heappop(pq)
53         idle_end = current_time + idle_task.execute_time
54         if idle_end > next_task.arrive_time:
55             idle_task.execute_time -= (next_task.arrive_time - current
56             _time)
57             heapq.heappush(pq, idle_task)
58             break
59         else:
60             print(idle_task.id, idle_end)
61             current_time = idle_end
62
63         current_time = next_task.arrive_time
64
65     # 新任务入队
66     heapq.heappush(pq, next_task)
67
68 # 处理剩余未执行的任务 (无抢占问题)
69 while pq:
70     t = heapq.heappop(pq)
71     end = current_time + t.execute_time
72     print(t.id, end)
73     current_time = end
```

## JavaScript

```
1 const readline = require('readline');
2
3 class Task {
4     constructor(id, priority, executeTime, arriveTime) {
5         this.id = id;
6         this.priority = priority;
7         this.executeTime = executeTime;
8         this.arriveTime = arriveTime;
9     }
10 }
11
12 // 自定义最大堆 (priority大 > arriveTime\)
13 class PriorityQueue {
14     constructor() {
15         this.heap = [];
16     }
17
18     compare(a, b) {
19         if (a.priority === b.priority) {
20             return b.id - a.id; // id 小的优先
21         }
22         return a.priority - b.priority; // priority 大的优先
23     }
24
25     push(task) {
26         this.heap.push(task);
27         this._siftUp(this.heap.length - 1);
28     }
29
30     pop() {
31         if (this.isEmpty()) return null;
32         const top = this.heap[0];
33         const end = this.heap.pop();
34         if (this.heap.length) {
35             this.heap[0] = end;
36             this._siftDown(0);
37         }
38         return top;
39     }
40
41     peek() {
42         return this.heap[0];
43     }
44
45     isEmpty() {
```

```

46         return this.heap.length === 0;
47     }
48
49     _siftUp(i) {
50         while (i > 0) {
51             const p = Math.floor((i - 1) / 2);
52             if (this.compare(this.heap[i], this.heap[p]) > 0) {
53                 [this.heap[i], this.heap[p]] = [this.heap[p], this.heap
54 [i]];
55                 i = p;
56             } else break;
57         }
58     }
59
60     _siftDown(i) {
61         const n = this.heap.length;
62         while (true) {
63             let largest = i;
64             const l = 2 * i + 1;
65             const r = 2 * i + 2;
66             if (l < n && this.compare(this.heap[l], this.heap[largest])
67 > 0) largest = l;
68             if (r < n && this.compare(this.heap[r], this.heap[largest])
69 > 0) largest = r;
70             if (largest !== i) {
71                 [this.heap[i], this.heap[largest]] = [this.heap[larges
72 t], this.heap[i]];
73                 i = largest;
74             } else break;
75         }
76     }
77 }
78
79 const rl = readline.createInterface({
80     input: process.stdin,
81     output: process.stdout
82 });
83
84 const tasks = [];
85
86 rl.on('line', (line) => {
87     if (line.trim() === '') {
88         rl.close();
89         return;
90     }
91     const [id, p, t, a] = line.trim().split(' ').map(Number);
92     tasks.push(new Task(id, p, t, a));
93 });

```

```

90
91     rl.on('close', () => {
92         const pq = new PriorityQueue();
93         let index = 0;
94         pq.push(tasks[index++]);
95         let currentTime = pq.peek().arriveTime;
96
97         while (index < tasks.length) {
98             const currentTask = pq.pop();
99             const nextTask = tasks[index++];
100
101            const endTime = currentTime + currentTask.executeTime;
102
103            // 当前任务不能执行完，下一个任务先到
104            if (endTime > nextTask.arriveTime) {
105                currentTask.executeTime -= (nextTask.arriveTime - currentTime);
106
107                currentTime = nextTask.arriveTime;
108                pq.push(currentTask); // 放回队列
109            } else {
110                // 当前任务可以完整执行
111                console.log(`#${currentTask.id} ${endTime}`);
112                currentTime = endTime;
113
114                // 执行等待队列中优先级高的任务，直到新任务到达
115                while (!pq.isEmpty() && currentTime < nextTask.arriveTime) {
116                    const idleTask = pq.pop();
117                    const idleEnd = currentTime + idleTask.executeTime;
118                    if (idleEnd > nextTask.arriveTime) {
119                        idleTask.executeTime -= (nextTask.arriveTime - currentTime);
120
121                        pq.push(idleTask);
122                        break;
123                    } else {
124                        console.log(`#${idleTask.id} ${idleEnd}`);
125                        currentTime = idleEnd;
126                    }
127
128                }
129
130                pq.push(nextTask);
131            }
132
133            // 队列中剩下的任务无抢占，直接执行
134            while (!pq.isEmpty()) {
135                const task = pq.pop();

```

```
136     const end = currentTime + task.executeTime;
137     console.log(`#${task.id} ${end}`);
138     currentTime = end;
139   }
140 );
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "container/heap"
6     "fmt"
7     "os"
8     "strconv"
9     "strings"
10    )
11
12 // 定义任务结构体
13 type Task struct {
14     id         int
15     priority   int
16     executeTime int
17     arriveTime int
18 }
19
20 // 优先队列实现, 基于 container/heap
21 type TaskPQ []*Task
22
23 func (pq TaskPQ) Len() int { return len(pq) }
24
25 // 优先队列排序规则:
26 // priority 大的优先, priority 相等时 id 小的优先
27 func (pq TaskPQ) Less(i, j int) bool {
28     if pq[i].priority == pq[j].priority {
29         return pq[i].id < pq[j].id
30     }
31     return pq[i].priority > pq[j].priority
32 }
33
34 func (pq TaskPQ) Swap(i, j int) {
35     pq[i], pq[j] = pq[j], pq[i]
36 }
37
38 func (pq *TaskPQ) Push(x interface{}) {
39     *pq = append(*pq, x.(*Task))
40 }
41
42 func (pq *TaskPQ) Pop() interface{} {
43     old := *pq
44     n := len(old)
45     item := old[n-1]
```

```
46     *pq = old[:n-1]
47     return item
48 }
49
50 func main() {
51     reader := bufio.NewReader(os.Stdin)
52     var tasks []*Task
53
54     // 读入任务，空行结束
55     for {
56         line, err := reader.ReadString('\n')
57         if err != nil {
58             break
59         }
60         line = strings.TrimSpace(line)
61         if line == "" {
62             break
63         }
64         parts := strings.Fields(line)
65         if len(parts) < 4 {
66             continue
67         }
68         id, _ := strconv.Atoi(parts[0])
69         p, _ := strconv.Atoi(parts[1])
70         t, _ := strconv.Atoi(parts[2])
71         a, _ := strconv.Atoi(parts[3])
72         tasks = append(tasks, &Task{id: id, priority: p, executeTime: t,
73                               arriveTime: a})
74     }
75
76     pq := &TaskPQ{}
77     heap.Init(pq)
78
79     // 入队第一个任务
80     index := 0
81     heap.Push(pq, tasks[index])
82     index++
83     currentTime := (*pq)[0].arriveTime
84
85     // 处理所有任务
86     for index < len(tasks) {
87         currentTask := heap.Pop(pq).(*Task)
88         nextTask := tasks[index]
89         index++
90
91         currentTaskEndTime := currentTime + currentTask.executeTime
92
93         // 当前任务执行结束时间 大于 下一个任务到达时间
```

```
93     if currentTaskEndTime > nextTask.arriveTime {
94         // 扣除已执行时间
95         currentTask.executeTime -= (nextTask.arriveTime - currentTim
96     e)
97         currentTime = nextTask.arriveTime
98         heap.Push(pq, currentTask) // 重新入队剩余任务
99     } else {
100        // 当前任务执行完成
101        fmt.Println(currentTask.id, currentTaskEndTime)
102        currentTime = currentTaskEndTime
103
104        // 利用空闲时间执行等待队列中的任务
105        for pq.Len() > 0 && currentTime < nextTask.arriveTime {
106            idleTask := heap.Pop(pq).(*Task)
107            idleTaskEndTime := currentTime + idleTask.executeTime
108
109            if idleTaskEndTime > nextTask.arriveTime {
110                // 执行一部分, 剩余继续入队
111                idleTask.executeTime -= (nextTask.arriveTime - curren
112                tTime)
113                heap.Push(pq, idleTask)
114                break
115            } else {
116                // 执行完当前任务
117                fmt.Println(idleTask.id, idleTaskEndTime)
118                currentTime = idleTaskEndTime
119            }
120
121            currentTime = nextTask.arriveTime
122        }
123
124        // 新任务入队
125        heap.Push(pq, nextTask)
126    }
127
128    // 处理剩余任务 (无抢占)
129    for pq.Len() > 0 {
130        t := heap.Pop(pq).(*Task)
131        endTime := currentTime + t.executeTime
132        fmt.Println(t.id, endTime)
133        currentTime = endTime
134    }
}
```



# 华为OD机试 2025C卷 - 提取字符串中最长数学表达式 (C++ & Python & JAVA & JS & GO)\_华为od机试 - 提取字符串中的最长合法简单数学表达式-CSDN博客

## 提取字符串中最长数学表达式

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

华为OD机试2025C卷 100分题型

### 题目描述

提取字符串中的最长合法简单数学表达式，[字符串长度](#)最长的，并计算表达式的值。如果没有，则返回 0。

简单数学表达式只能包含以下内容：

- 0-9数字，符号 $\pm\ast$

说明：

1. 所有数字，计算结果都不超过long
2. 如果有多个长度一样的，请返回第一个表达式的结果
3. 数学表达式，必须是最长的，合法的
4. 操作符不能连续出现，如  $\pm-+1$  是不合法的

### 输入描述

字符串

### 输出描述

表达式值

### 示例1

#### 输入

|           |            |
|-----------|------------|
| ▼         | Plain Text |
| 1 1-2abcd |            |

# 输出

```
Plain Text |  
1 -1
```

## 题解

思路： 双指针 + 字符串处理 + 字符串表达式计算

1. 表达式提取： 使用 双指针 进行表达式提取，根据题目描述可知 一个正常的表达式组成为 数字 + (运算符 数字) 重复 组成。基于此可以使用双指针来解析字符串中表达式。解析表达式的大致逻辑如下：
  - a. 确定起点： 尝试表达式起点位置，找到上一个遍历结束位置之后的第一个数字字符(表达式第一部 分一定是数字)。可以定义一个数组/栈 stk 按照顺序存储表达式各个部分由两个好处：1. 遍历过 程中方便判断表达式是否合法。2. 方便后续进行表达式值运算。
  - b. 贪婪尽可能让表达式更长： 确定第一个数字部分，将其存入数组。就是向后遍历不断进行提取， 直到出现非法情况结束，非法情况大致包括以下几种：
    - 连续出现运算符
    - 出现非运算符和数字
  - c. 根据2逻辑检查数组 stk 中表达式是否合法？如果stk末尾为运算符先弹出末尾元素，否则不用 做处理。然后判断数组/栈长度是否 $\geq 3$ ？如果是，那么就是一个合法的表达式。
  - d. 更新下一次枚举起点为本次枚举终点的下一个位置。
2. 表达式计算：如果没有合法表达式直接输出 0 即可。有表达式此时需要进行计算，运算符包含 + - \*，其中 \* 优先级会高于 + -，题目指出 简单数学表达式只能包含以下内容：0-9数字，符号+-\*，这种情况下处理优先级就比较简单了。使用两个栈(操作数栈、 运算符栈)执行运算即可，具体逻辑可 参照下面代码。

C++

```
1 #include <cctype>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 #include<cmath>
9 #include<map>
10 #include<stack>
11 using namespace std;
12
13 // 判断是否为运算符
14 int isOperator(string c) {
15     return c == "+" || c == "-" || c == "*";
16 }
17
18 // 使用栈处理表达式计算
19 int calc(vector<string>& express) {
20     // 操作符栈
21     stack<string> operatorStack;
22     // 操作数栈
23     stack<int> numberStack;
24     int n = express.size();
25     for (int i = 0; i < n; i++) {
26         if (isOperator(express[i])) {
27             if (operatorStack.empty()) {
28                 operatorStack.push(express[i]);
29             } else {
30                 // 当前操作符优先级小于上一个，上一个应该可以直接进行运行
31                 while ((express[i] == "+" || express[i] == "-") && !operatorStack.empty() && operatorStack.top() == "*") {
32                     int number2 = numberStack.top(); numberStack.pop();
33                     int number1 = numberStack.top(); numberStack.pop();
34                     operatorStack.pop();
35                     int res = number1 * number2;
36                     numberStack.push(res);
37                 }
38                 operatorStack.push(express[i]);
39             }
40         } else {
41             numberStack.push(stoi(express[i]));
42         }
43     }
44 }
```

```

45     while (!operatorStack.empty()) {
46         string operat = operatorStack.top();
47         operatorStack.pop();
48         int number2 = numberStack.top();numberStack.pop();
49         int number1 = numberStack.top();numberStack.pop();
50         int res = 0;
51         if (operat == "+") {
52             res = number1 + number2;
53         } else if (operat == "-"){
54             res = number1 - number2;
55         } else {
56             res = number1 * number2;
57         }
58         numberStack.push(res);
59     }
60     int res = numberStack.top();
61     return res;
62 }
63
64 int main() {
65     string input;
66     getline(cin, input);
67     int start = 0;
68     int n = input.size();
69
70     vector<string> res;
71     int maxLen = 0;
72
73     while (start < n) {
74         int end = start;
75         // 表达式以数字开始
76         if (!isdigit(input[end])) {
77             start = end + 1;
78             continue;
79         }
80         // 存储合法的表达式片段
81         stack<string> stk;
82         while (end < n) {
83             string tmp = "";
84             if (isdigit(input[end])) {
85                 // 获取连续数字字符
86                 tmp.push_back(input[end]);
87                 // 贪婪的一次获取所有的数字
88                 while (end + 1< n && isdigit(input[end + 1])) {
89                     end += 1;
90                     tmp.push_back(input[end]);
91                 }
92             }

```

```

93         } else if (input[end] == '*' || input[end] == '+' || input[en
94 d] == '-') {
95             tmp.push_back(input[end]);
96         } else {
97             break;
98         }
99         // 判断是否合法
100
101         // 当前字符串是运算符, 前一个必须为数字
102         if (isOperator(tmp)) {
103             if (stk.empty()) {
104                 break;
105             }
106             string top = stk.top();
107             // 不能连续出现两个运算符
108             if (isOperator(top)) {
109                 break;
110             }
111             stk.push(tmp);
112             // 当前为数字 栈为空或者前一个是运算符
113         } else {
114             if (stk.empty() || isOperator(stk.top())) {
115                 stk.push(tmp);
116             } else {
117                 break;
118             }
119         }
120         end++;
121     }
122
123     // 栈中如果最后为运算符直接移除
124     if (!stk.empty() && isOperator(stk.top())) {
125         stk.pop();
126     }
127
128     // 大于3个这样才会构成一个合法的表达式
129     if (stk.size() >= 3) {
130         vector<string> tmp;
131         // 计算长度
132         int currentLen = 0;
133         while (!stk.empty()) {
134             string top = stk.top();
135             stk.pop();
136             currentLen += top.size();
137             tmp.push_back(top);
138         }
139         // 当前表达式大于之前表达式长度时
140         if (currentLen > maxLen) {

```

```
140             // 反转一下,
141             reverse(tmp.begin(), tmp.end());
142             res = tmp;
143             maxLen = currentLen;
144         }
145     }
146     start = end + 1;
147 }
148
149
150 // 没有合法表达式
151 if (maxLen == 0) {
152     cout << 0;
153 // 计算值
154 } else {
155     int calRes = calc(res);
156     cout << calRes;
157 }
158
159 return 0;
160
```

,

## Java

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为运算符
5     static boolean isOperator(String c) {
6         return c.equals("+") || c.equals("-") || c.equals("*");
7     }
8
9     // 使用栈处理表达式计算
10    static int calc(List<String> express) {
11        // 操作符栈
12        Stack<String> operatorStack = new Stack<>();
13        // 操作数栈
14        Stack<Integer> numberStack = new Stack<>();
15
16        for (String token : express) {
17            if (isOperator(token)) {
18                if (operatorStack.isEmpty()) {
19                    operatorStack.push(token);
20                } else {
21                    // 当前操作符优先级小于上一个，上一个应该可以直接进行运行
22                    while ((token.equals("+") || token.equals("-")) &&
23                           !operatorStack.isEmpty() && operatorStack.peek()
24                           .equals("*")) {
25                        int number2 = numberStack.pop();
26                        int number1 = numberStack.pop();
27                        operatorStack.pop();
28                        numberStack.push(number1 * number2);
29                    }
30                    operatorStack.push(token);
31                }
32            } else {
33                numberStack.push(Integer.parseInt(token));
34            }
35
36
37            while (!operatorStack.isEmpty()) {
38                String op = operatorStack.pop();
39                int number2 = numberStack.pop();
40                int number1 = numberStack.pop();
41                int res = 0;
42                if (op.equals("+")) res = number1 + number2;
43                else if (op.equals("-")) res = number1 - number2;
44                else res = number1 * number2;
45            }
46        }
47    }
48}
```

```

45             numberStack.push(res);
46         }
47
48         return numberStack.pop();
49     }
50
51     public static void main(String[] args) {
52         Scanner sc = new Scanner(System.in);
53         String input = sc.nextLine();
54         int start = 0;
55         int n = input.length();
56         List<String> res = new ArrayList<>();
57         int maxLen = 0;
58
59         while (start < n) {
60             int end = start;
61             // 表达式以数字开始
62             if (!Character.isDigit(input.charAt(end))) {
63                 start = end + 1;
64                 continue;
65             }
66             Stack<String> stk = new Stack<>();
67             while (end < n) {
68                 StringBuilder tmp = new StringBuilder();
69                 if (Character.isDigit(input.charAt(end))) {
70                     tmp.append(input.charAt(end));
71                     // 贪婪的一次获取所有的数字
72                     while (end + 1 < n && Character.isDigit(input.charAt(
73                         (end + 1))) {
74                         end++;
75                         tmp.append(input.charAt(end));
76                     }
77                     } else if ("+-*".indexOf(input.charAt(end)) != -1) {
78                         tmp.append(input.charAt(end));
79                     } else {
80                         break;
81                     }
82
83                     // 判断是否合法
84                     if (isOperator(tmp.toString())) {
85                         if (stk.isEmpty()) break;
86                         String top = stk.peek();
87                         // 不能连续出现两个运算符
88                         if (isOperator(top)) break;
89                         stk.push(tmp.toString());
90                     } else {
91                         if (stk.isEmpty() || isOperator(stk.peek())) {
92                             stk.push(tmp.toString());
93                         }
94                     }
95                 }
96             }
97         }
98     }
99 }
```

```

92                     } else {
93                         break;
94                     }
95                 }
96             end++;
97         }
98
99         // 栈中如果最后为运算符直接移除
100        if (!stk.isEmpty() && isOperator(stk.peek())) {
101            stk.pop();
102        }
103
104        // 大于3个才构成合法表达式
105        if (stk.size() >= 3) {
106            List<String> tmp = new ArrayList<>();
107            int currentLen = 0;
108            while (!stk.isEmpty()) {
109                String top = stk.pop();
110                currentLen += top.length();
111                tmp.add(top);
112            }
113            if (currentLen > maxLen) {
114                Collections.reverse(tmp);
115                res = tmp;
116                maxLen = currentLen;
117            }
118        }
119
120        start = end + 1;
121    }
122
123    // 没有合法表达式
124    if (maxLen == 0) {
125        System.out.println(0);
126    } else {
127        int result = calc(res);
128        System.out.println(result);
129    }
130}
131}

```

## Python

```
1 def is_operator(c):
2     # 判断是否为运算符
3     return c in {"+", "-", "*"}
4
5 # 使用栈处理表达式计算
6 def calc(express):
7     # 操作符栈
8     operator_stack = []
9     # 操作数栈
10    number_stack = []
11
12    for token in express:
13        if is_operator(token):
14            if not operator_stack:
15                operator_stack.append(token)
16            else:
17                # 当前操作符优先级小于上一个，上一个应该可以直接进行运行
18                while token in "+-" and operator_stack and operator_stack[-1] == "*":
19                    b = number_stack.pop()
20                    a = number_stack.pop()
21                    operator_stack.pop()
22                    number_stack.append(a * b)
23                    operator_stack.append(token)
24            else:
25                number_stack.append(int(token))
26
27
28        while operator_stack:
29            op = operator_stack.pop()
30            b = number_stack.pop()
31            a = number_stack.pop()
32            if op == "+":
33                number_stack.append(a + b)
34            elif op == "-":
35                number_stack.append(a - b)
36            else:
37                number_stack.append(a * b)
38
39    return number_stack[0]
40
41 input_str = input()
42 n = len(input_str)
43 start = 0
44 res = []
```

```

45 max_len = 0
46
47 while start < n:
48     end = start
49     # 表达式以数字开始
50     if not input_str[end].isdigit():
51         start = end + 1
52         continue
53     stk = []
54     while end < n:
55         tmp = ''
56         if input_str[end].isdigit():
57             tmp += input_str[end]
58             while end + 1 < n and input_str[end + 1].isdigit():
59                 end += 1
60                 tmp += input_str[end]
61             elif input_str[end] in "+-*":
62                 tmp += input_str[end]
63             else:
64                 break
65
66             # 判断是否合法
67             if is_operator(tmp):
68                 # 连续出现两个操作符
69                 if not stk or is_operator(stk[-1]):
70                     break
71                 stk.append(tmp)
72             else:
73                 if not stk or is_operator(stk[-1]):
74                     stk.append(tmp)
75                 else:
76                     break
77             end += 1
78
79             if stk and is_operator(stk[-1]):
80                 stk.pop()
81
82             if len(stk) >= 3:
83                 current_len = sum(len(x) for x in stk)
84                 if current_len > max_len:
85                     res = stk[:]
86                     max_len = current_len
87
88             start = end + 1
89
90         if max_len == 0:
91             print(0)
92         else:

```

```
93     print(calc(res))
```

## JavaScript

```
1 const readline = require('readline');
2
3 function isOperator(c) {
4     // 判断是否为运算符
5     return c === '+' || c === '-' || c === '*';
6 }
7
8 // 使用栈处理表达式计算
9 function calc(express) {
10    // 操作符栈
11    const operatorStack = [];
12    // 操作数栈
13    const numberStack = [];
14
15    for (let token of express) {
16        if (isOperator(token)) {
17            if (operatorStack.length === 0) {
18                operatorStack.push(token);
19            } else {
20                // 当前操作符优先级小于上一个
21                while ((token === '+' || token === '-') &&
22                    operatorStack.length > 0 &&
23                    operatorStack[operatorStack.length - 1] === '*') {
24                    let b = numberStack.pop();
25                    let a = numberStack.pop();
26                    operatorStack.pop();
27                    numberStack.push(a * b);
28                }
29                operatorStack.push(token);
30            }
31        } else {
32            numberStack.push(Number(token));
33        }
34    }
35
36
37    while (operatorStack.length > 0) {
38        let op = operatorStack.pop();
39        let b = numberStack.pop();
40        let a = numberStack.pop();
41        let res = 0;
42        if (op === '+') res = a + b;
43        else if (op === '-') res = a - b;
44        else res = a * b;
45        numberStack.push(res);
46    }
47}
```

```
46     }
47
48     return numberStack[0];
49 }
50
51 const rl = readline.createInterface({
52     input: process.stdin,
53     output: process.stdout
54 });
55
56 rl.on('line', function (input) {
57     let start = 0;
58     let n = input.length;
59     let res = [];
60     let maxLen = 0;
61
62     while (start < n) {
63         let end = start;
64         // 表达式以数字开始
65         if (isNaN(parseInt(input[end]))) {
66             start = end + 1;
67             continue;
68         }
69         let stk = [];
70         while (end < n) {
71             let tmp = '';
72             // 贪婪一次获取所有连续数字
73             if (!isNaN(parseInt(input[end]))) {
74                 tmp += input[end];
75                 while (end + 1 < n && !isNaN(parseInt(input[end + 1]))) {
76                     end++;
77                     tmp += input[end];
78                 }
79             } else if ('+-*'.includes(input[end])) {
80                 tmp += input[end];
81             } else {
82                 break;
83             }
84
85             // 判断是否合法
86             if (isOperator(tmp)) {
87                 // 出现两个操作符
88                 if (stk.length === 0 || isOperator(stk[stk.length - 1]))
89                     break;
90                 stk.push(tmp);
91             } else {
92                 if (stk.length === 0 || isOperator(stk[stk.length - 1]))
93             }
94
95         }
96         if (maxLen < end - start + 1)
97             maxLen = end - start + 1;
98     }
99
100    res.push(maxLen);
101
102    rl.close();
103 }
104
105 module.exports = {
106     calculate,
107     isOperator
108 }
```

```
92             stk.push(tmp);
93         } else break;
94     }
95     end++;
96 }
97 // 末尾是操作符的话直接移除
98 if (stk.length && isOperator(stk[stk.length - 1])) {
99     stk.pop();
100 }
101 // 大于3才能构成合法表达式
102 if (stk.length >= 3) {
103     let len = stk.reduce((a, b) => a + b.length, 0);
104     if (len > maxLen) {
105         res = [...stk];
106         maxLen = len;
107     }
108 }
109
110     start = end + 1;
111 }
112
113 if (maxLen === 0) {
114     console.log(0);
115 } else {
116     console.log(calc(res));
117 }
118
119 rl.close();
120 });
});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 判断是否为运算符
12 func isOperator(c string) bool {
13     return c == "+" || c == "-" || c == "*"
14 }
15
16 // 使用栈处理表达式计算
17 func calc(express []string) int {
18     // 操作符栈
19     operatorStack := []string{}
20     // 操作数栈
21     numberStack := []int{}
22
23     for _, token := range express {
24         if isOperator(token) {
25             if len(operatorStack) == 0 {
26                 operatorStack = append(operatorStack, token)
27             } else {
28                 // 当前操作符优先级小于上一个，上一个应该可以直接进行运算
29                 for (token == "+" || token == "-") && len(operatorStack) > 0 && operatorStack[len(operatorStack)-1] == "*" {
30                     n2 := numberStack[len(numberStack)-1]
31                     numberStack = numberStack[:len(numberStack)-1]
32                     n1 := numberStack[len(numberStack)-1]
33                     numberStack = numberStack[:len(numberStack)-1]
34                     operatorStack = operatorStack[:len(operatorStack)-1]
35                     numberStack = append(numberStack, n1*n2)
36                 }
37                 operatorStack = append(operatorStack, token)
38             }
39         } else {
40             val, _ := strconv.Atoi(token)
41             numberStack = append(numberStack, val)
42         }
43     }
44 }
```

```

45
46     for len(operatorStack) > 0 {
47         oper := operatorStack[len(operatorStack)-1]
48         operatorStack = operatorStack[:len(operatorStack)-1]
49         n2 := numberStack[len(numberStack)-1]
50         numberStack = numberStack[:len(numberStack)-1]
51         n1 := numberStack[len(numberStack)-1]
52         numberStack = numberStack[:len(numberStack)-1]
53         res := 0
54         if oper == "+" {
55             res = n1 + n2
56         } else if oper == "-" {
57             res = n1 - n2
58         } else {
59             res = n1 * n2
60         }
61         numberStack = append(numberStack, res)
62     }
63     return numberStack[0]
64 }
65
66 func main() {
67     reader := bufio.NewReader(os.Stdin)
68     input, _ := reader.ReadString('\n')
69     input = strings.TrimSpace(input)
70     n := len(input)
71     start := 0
72     res := []string{}
73     maxlen := 0
74
75     for start < n {
76         end := start
77         // 表达式以数字开始
78         if !('0' <= input[end] && input[end] <= '9') {
79             start = end + 1
80             continue
81         }
82         // 存储合法的表达式片段
83         stk := []string{}
84         for end < n {
85             tmp := ""
86             if '0' <= input[end] && input[end] <= '9' {
87                 tmp += string(input[end])
88                 for end+1 < n && '0' <= input[end+1] && input[end+1] <= '9' {
89                     end++
90                     tmp += string(input[end])
91                 }
92             }

```

```

    } else if input[end] == '*' || input[end] == '+' || input[end] ==
93  '-' {
94      tmp += string(input[end])
95  } else {
96      break
97  }
98 // 判断是否合法
99  if isOperator(tmp) {
100     if len(stk) == 0 {
101         break
102     }
103     top := stk[len(stk)-1]
104     if isOperator(top) {
105         break
106     }
107     stk = append(stk, tmp)
108 } else {
109     if len(stk) == 0 || isOperator(stk[len(stk)-1]) {
110         stk = append(stk, tmp)
111     } else {
112         break
113     }
114 }
115 end++
116 }
117

118 // 栈中如果最后为运算符直接移除
119 if len(stk) > 0 && isOperator(stk[len(stk)-1]) {
120     stk = stk[:len(stk)-1]
121 }
122

123 // 大于3个这样才会构成一个合法的表达式
124 if len(stk) >= 3 {
125     tmp := []string{}
126     currentLen := 0
127     for i := len(stk) - 1; i >= 0; i-- {
128         top := stk[i]
129         currentLen += len(top)
130         tmp = append(tmp, top)
131     }
132     if currentLen > maxLen {
133         // 反转
134         for i, j := 0, len(tmp)-1; i < j; i, j = i+1, j-1 {
135             tmp[i], tmp[j] = tmp[j], tmp[i]
136         }
137         res = tmp
138         maxLen = currentLen
139     }

```

```
140     }
141     start = end + 1
142 }
143
144 // 没有合法表达式
145 if maxLen == 0 {
146     fmt.Println(0)
147 } else {
148     // 计算值
149     result := calc(res)
150     fmt.Println(result)
151 }
```

## 提取字符串中最长数学表达式

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

华为OD机试2025C卷 100分题型

### 题目描述

提取字符串中的最长合法简单数学表达式，[字符串长度](#)最长的，并计算表达式的值。如果没有，则返回 0。

简单数学表达式只能包含以下内容：

- 0-9数字，符号 $\pm\ast$

说明：

1. 所有数字，计算结果都不超过long
2. 如果有多个长度一样的，请返回第一个表达式的结果
3. 数学表达式，必须是最长的，合法的
4. 操作符不能连续出现，如  $\pm-+1$  是不合法的

### 输入描述

字符串

### 输出描述

表达式值

### 示例1

#### 输入

▼ Plain Text |  
1 1-2abcd

## 输出

```
Plain Text |  
1 -1
```

## 题解

思路： 双指针 + 字符串处理 + 字符串表达式计算

1. 表达式提取： 使用 双指针 进行表达式提取，根据题目描述可知 一个正常的表达式组成为 数字 + (运算符 数字) 重复 组成。基于此可以使用双指针来解析字符串中表达式。解析表达式的大致逻辑如下：
  - a. 确定起点： 尝试表达式起点位置，找到上一个遍历结束位置之后的第一个数字字符(表达式第一部 分一定是数字)。可以定义一个数组/栈 stk 按照顺序存储表达式各个部分由两个好处：1. 遍历过 程中方便判断表达式是否合法。2. 方便后续进行表达式值运算。
  - b. 贪婪尽可能让表达式更长： 确定第一个数字部分，将其存入数组。就是向后遍历不断进行提取， 直到出现非法情况结束，非法情况大致包括以下几种：
    - 连续出现运算符
    - 出现非运算符和数字
  - c. 根据2逻辑检查数组 stk 中表达式是否合法？如果stk末尾为运算符先弹出末尾元素，否则不用 做处理。然后判断数组/栈长度是否 $\geq 3$ ？如果是，那么就是一个合法的表达式。
  - d. 更新下一次枚举起点为本次枚举终点的下一个位置。
2. 表达式计算：如果没有合法表达式直接输出 0 即可。有表达式此时需要进行计算，运算符包含 + - \*，其中 \* 优先级会高于 + -，题目指出 简单数学表达式只能包含以下内容：0-9数字，符号+-\*，这种情况下处理优先级就比较简单了。使用两个栈(操作数栈、 运算符栈)执行运算即可，具体逻辑可 参照下面代码。

C++

```
1 #include <cctype>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 #include<cmath>
9 #include<map>
10 #include<stack>
11 using namespace std;
12
13 // 判断是否为运算符
14 int isOperator(string c) {
15     return c == "+" || c == "-" || c == "*";
16 }
17
18 // 使用栈处理表达式计算
19 int calc(vector<string>& express) {
20     // 操作符栈
21     stack<string> operatorStack;
22     // 操作数栈
23     stack<int> numberStack;
24     int n = express.size();
25     for (int i = 0; i < n; i++) {
26         if (isOperator(express[i])) {
27             if (operatorStack.empty()) {
28                 operatorStack.push(express[i]);
29             } else {
30                 // 当前操作符优先级小于上一个，上一个应该可以直接进行运行
31                 while ((express[i] == "+" || express[i] == "-") && !operatorStack.empty() && operatorStack.top() == "*") {
32                     int number2 = numberStack.top(); numberStack.pop();
33                     int number1 = numberStack.top(); numberStack.pop();
34                     operatorStack.pop();
35                     int res = number1 * number2;
36                     numberStack.push(res);
37                 }
38                 operatorStack.push(express[i]);
39             }
40         } else {
41             numberStack.push(stoi(express[i]));
42         }
43     }
44 }
```

```

45     while (!operatorStack.empty()) {
46         string operat = operatorStack.top();
47         operatorStack.pop();
48         int number2 = numberStack.top();numberStack.pop();
49         int number1 = numberStack.top();numberStack.pop();
50         int res = 0;
51         if (operat == "+") {
52             res = number1 + number2;
53         } else if (operat == "-"){
54             res = number1 - number2;
55         } else {
56             res = number1 * number2;
57         }
58         numberStack.push(res);
59     }
60     int res = numberStack.top();
61     return res;
62 }
63
64 int main() {
65     string input;
66     getline(cin, input);
67     int start = 0;
68     int n = input.size();
69
70     vector<string> res;
71     int maxLen = 0;
72
73     while (start < n) {
74         int end = start;
75         // 表达式以数字开始
76         if (!isdigit(input[end])) {
77             start = end + 1;
78             continue;
79         }
80         // 存储合法的表达式片段
81         stack<string> stk;
82         while (end < n) {
83             string tmp = "";
84             if (isdigit(input[end])) {
85                 // 获取连续数字字符
86                 tmp.push_back(input[end]);
87                 // 贪婪的一次获取所有的数字
88                 while (end + 1< n && isdigit(input[end + 1])) {
89                     end += 1;
90                     tmp.push_back(input[end]);
91                 }
92             }

```

```

93         } else if (input[end] == '*' || input[end] == '+' || input[en
94 d] == '-') {
95             tmp.push_back(input[end]);
96         } else {
97             break;
98         }
99         // 判断是否合法
100
101         // 当前字符串是运算符, 前一个必须为数字
102         if (isOperator(tmp)) {
103             if (stk.empty()) {
104                 break;
105             }
106             string top = stk.top();
107             // 不能连续出现两个运算符
108             if (isOperator(top)) {
109                 break;
110             }
111             stk.push(tmp);
112             // 当前为数字 栈为空或者前一个是运算符
113         } else {
114             if (stk.empty() || isOperator(stk.top())) {
115                 stk.push(tmp);
116             } else {
117                 break;
118             }
119         }
120         end++;
121     }
122
123     // 栈中如果最后为运算符直接移除
124     if (!stk.empty() && isOperator(stk.top())) {
125         stk.pop();
126     }
127
128     // 大于3个这样才会构成一个合法的表达式
129     if (stk.size() >= 3) {
130         vector<string> tmp;
131         // 计算长度
132         int currentLen = 0;
133         while (!stk.empty()) {
134             string top = stk.top();
135             stk.pop();
136             currentLen += top.size();
137             tmp.push_back(top);
138         }
139         // 当前表达式大于之前表达式长度时
140         if (currentLen > maxLen) {

```

```
140             // 反转一下,
141             reverse(tmp.begin(), tmp.end());
142             res = tmp;
143             maxLen = currentLen;
144         }
145     }
146     start = end + 1;
147 }
148
149
150 // 没有合法表达式
151 if (maxLen == 0) {
152     cout << 0;
153 // 计算值
154 } else {
155     int calRes = calc(res);
156     cout << calRes;
157 }
158
159 return 0;
160
```

## Java

```
1 import java.util.*;
2
3 public class Main {
4     // 判断是否为运算符
5     static boolean isOperator(String c) {
6         return c.equals("+") || c.equals("-") || c.equals("*");
7     }
8
9     // 使用栈处理表达式计算
10    static int calc(List<String> express) {
11        // 操作符栈
12        Stack<String> operatorStack = new Stack<>();
13        // 操作数栈
14        Stack<Integer> numberStack = new Stack<>();
15
16        for (String token : express) {
17            if (isOperator(token)) {
18                if (operatorStack.isEmpty()) {
19                    operatorStack.push(token);
20                } else {
21                    // 当前操作符优先级小于上一个，上一个应该可以直接进行运行
22                    while ((token.equals("+") || token.equals("-")) &&
23                           !operatorStack.isEmpty() && operatorStack.peek()
24                           .equals("*")) {
25                        int number2 = numberStack.pop();
26                        int number1 = numberStack.pop();
27                        operatorStack.pop();
28                        numberStack.push(number1 * number2);
29                    }
30                    operatorStack.push(token);
31                }
32            } else {
33                numberStack.push(Integer.parseInt(token));
34            }
35
36
37            while (!operatorStack.isEmpty()) {
38                String op = operatorStack.pop();
39                int number2 = numberStack.pop();
40                int number1 = numberStack.pop();
41                int res = 0;
42                if (op.equals("+")) res = number1 + number2;
43                else if (op.equals("-")) res = number1 - number2;
44                else res = number1 * number2;
45            }
46        }
47    }
48}
```

```

45             numberStack.push(res);
46         }
47
48         return numberStack.pop();
49     }
50
51     public static void main(String[] args) {
52         Scanner sc = new Scanner(System.in);
53         String input = sc.nextLine();
54         int start = 0;
55         int n = input.length();
56         List<String> res = new ArrayList<>();
57         int maxLen = 0;
58
59         while (start < n) {
60             int end = start;
61             // 表达式以数字开始
62             if (!Character.isDigit(input.charAt(end))) {
63                 start = end + 1;
64                 continue;
65             }
66             Stack<String> stk = new Stack<>();
67             while (end < n) {
68                 StringBuilder tmp = new StringBuilder();
69                 if (Character.isDigit(input.charAt(end))) {
70                     tmp.append(input.charAt(end));
71                     // 贪婪的一次获取所有的数字
72                     while (end + 1 < n && Character.isDigit(input.charAt(
73                         (end + 1))) {
74                         end++;
75                         tmp.append(input.charAt(end));
76                     }
77                     } else if ("+-*".indexOf(input.charAt(end)) != -1) {
78                         tmp.append(input.charAt(end));
79                     } else {
80                         break;
81                     }
82
83                     // 判断是否合法
84                     if (isOperator(tmp.toString())) {
85                         if (stk.isEmpty()) break;
86                         String top = stk.peek();
87                         // 不能连续出现两个运算符
88                         if (isOperator(top)) break;
89                         stk.push(tmp.toString());
90                     } else {
91                         if (stk.isEmpty() || isOperator(stk.peek())) {
92                             stk.push(tmp.toString());
93                         }
94                     }
95                 }
96             }
97         }
98     }
99 }
```

```

92                     } else {
93                         break;
94                     }
95                 }
96             end++;
97         }
98
99         // 栈中如果最后为运算符直接移除
100        if (!stk.isEmpty() && isOperator(stk.peek())) {
101            stk.pop();
102        }
103
104        // 大于3个才构成合法表达式
105        if (stk.size() >= 3) {
106            List<String> tmp = new ArrayList<>();
107            int currentLen = 0;
108            while (!stk.isEmpty()) {
109                String top = stk.pop();
110                currentLen += top.length();
111                tmp.add(top);
112            }
113            if (currentLen > maxLen) {
114                Collections.reverse(tmp);
115                res = tmp;
116                maxLen = currentLen;
117            }
118        }
119
120        start = end + 1;
121    }
122
123    // 没有合法表达式
124    if (maxLen == 0) {
125        System.out.println(0);
126    } else {
127        int result = calc(res);
128        System.out.println(result);
129    }
130}
131}

```

## Python

```
1 def is_operator(c):
2     # 判断是否为运算符
3     return c in {"+", "-", "*"}
4
5 # 使用栈处理表达式计算
6 def calc(express):
7     # 操作符栈
8     operator_stack = []
9     # 操作数栈
10    number_stack = []
11
12    for token in express:
13        if is_operator(token):
14            if not operator_stack:
15                operator_stack.append(token)
16            else:
17                # 当前操作符优先级小于上一个，上一个应该可以直接进行运行
18                while token in "+-" and operator_stack and operator_stack[-1] == "*":
19                    b = number_stack.pop()
20                    a = number_stack.pop()
21                    operator_stack.pop()
22                    number_stack.append(a * b)
23                    operator_stack.append(token)
24            else:
25                number_stack.append(int(token))
26
27
28        while operator_stack:
29            op = operator_stack.pop()
30            b = number_stack.pop()
31            a = number_stack.pop()
32            if op == "+":
33                number_stack.append(a + b)
34            elif op == "-":
35                number_stack.append(a - b)
36            else:
37                number_stack.append(a * b)
38
39    return number_stack[0]
40
41 input_str = input()
42 n = len(input_str)
43 start = 0
44 res = []
```

```

45 max_len = 0
46
47 while start < n:
48     end = start
49     # 表达式以数字开始
50     if not input_str[end].isdigit():
51         start = end + 1
52         continue
53     stk = []
54     while end < n:
55         tmp = ''
56         if input_str[end].isdigit():
57             tmp += input_str[end]
58             while end + 1 < n and input_str[end + 1].isdigit():
59                 end += 1
60                 tmp += input_str[end]
61             elif input_str[end] in "+-*":
62                 tmp += input_str[end]
63             else:
64                 break
65
66             # 判断是否合法
67             if is_operator(tmp):
68                 # 连续出现两个操作符
69                 if not stk or is_operator(stk[-1]):
70                     break
71                 stk.append(tmp)
72             else:
73                 if not stk or is_operator(stk[-1]):
74                     stk.append(tmp)
75                 else:
76                     break
77             end += 1
78
79             if stk and is_operator(stk[-1]):
80                 stk.pop()
81
82             if len(stk) >= 3:
83                 current_len = sum(len(x) for x in stk)
84                 if current_len > max_len:
85                     res = stk[:]
86                     max_len = current_len
87
88             start = end + 1
89
90         if max_len == 0:
91             print(0)
92         else:

```

```
93     print(calc(res))
```

## JavaScript

```
1 const readline = require('readline');
2
3 function isOperator(c) {
4     // 判断是否为运算符
5     return c === '+' || c === '-' || c === '*';
6 }
7
8 // 使用栈处理表达式计算
9 function calc(express) {
10    // 操作符栈
11    const operatorStack = [];
12    // 操作数栈
13    const numberStack = [];
14
15    for (let token of express) {
16        if (isOperator(token)) {
17            if (operatorStack.length === 0) {
18                operatorStack.push(token);
19            } else {
20                // 当前操作符优先级小于上一个
21                while ((token === '+' || token === '-') &&
22                    operatorStack.length > 0 &&
23                    operatorStack[operatorStack.length - 1] === '*') {
24                    let b = numberStack.pop();
25                    let a = numberStack.pop();
26                    operatorStack.pop();
27                    numberStack.push(a * b);
28                }
29                operatorStack.push(token);
30            }
31        } else {
32            numberStack.push(Number(token));
33        }
34    }
35
36
37    while (operatorStack.length > 0) {
38        let op = operatorStack.pop();
39        let b = numberStack.pop();
40        let a = numberStack.pop();
41        let res = 0;
42        if (op === '+') res = a + b;
43        else if (op === '-') res = a - b;
44        else res = a * b;
45        numberStack.push(res);
46    }
47}
```

```
46     }
47
48     return numberStack[0];
49 }
50
51 const rl = readline.createInterface({
52     input: process.stdin,
53     output: process.stdout
54 });
55
56 rl.on('line', function (input) {
57     let start = 0;
58     let n = input.length;
59     let res = [];
60     let maxLen = 0;
61
62     while (start < n) {
63         let end = start;
64         // 表达式以数字开始
65         if (isNaN(parseInt(input[end]))) {
66             start = end + 1;
67             continue;
68         }
69         let stk = [];
70         while (end < n) {
71             let tmp = '';
72             // 贪婪一次获取所有连续数字
73             if (!isNaN(parseInt(input[end]))) {
74                 tmp += input[end];
75                 while (end + 1 < n && !isNaN(parseInt(input[end + 1]))) {
76                     end++;
77                     tmp += input[end];
78                 }
79             } else if ('+-*'.includes(input[end])) {
80                 tmp += input[end];
81             } else {
82                 break;
83             }
84
85             // 判断是否合法
86             if (isOperator(tmp)) {
87                 // 出现两个操作符
88                 if (stk.length === 0 || isOperator(stk[stk.length - 1]))
89                     break;
90                 stk.push(tmp);
91             } else {
92                 if (stk.length === 0 || isOperator(stk[stk.length - 1]))
93             }
94
95         }
96         if (maxLen < end - start + 1)
97             maxLen = end - start + 1;
98     }
99
100    res.push(maxLen);
101
102    rl.close();
103 }
104
105 module.exports = {
106     calculate,
107     isOperator
108 }
```

```
92             stk.push(tmp);
93         } else break;
94     }
95     end++;
96 }
97 // 末尾是操作符的话直接移除
98 if (stk.length && isOperator(stk[stk.length - 1])) {
99     stk.pop();
100 }
101 // 大于3才能构成合法表达式
102 if (stk.length >= 3) {
103     let len = stk.reduce((a, b) => a + b.length, 0);
104     if (len > maxLen) {
105         res = [...stk];
106         maxLen = len;
107     }
108 }
109
110     start = end + 1;
111 }
112
113 if (maxLen === 0) {
114     console.log(0);
115 } else {
116     console.log(calc(res));
117 }
118
119 rl.close();
120 });
});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 判断是否为运算符
12 func isOperator(c string) bool {
13     return c == "+" || c == "-" || c == "*"
14 }
15
16 // 使用栈处理表达式计算
17 func calc(express []string) int {
18     // 操作符栈
19     operatorStack := []string{}
20     // 操作数栈
21     numberStack := []int{}
22
23     for _, token := range express {
24         if isOperator(token) {
25             if len(operatorStack) == 0 {
26                 operatorStack = append(operatorStack, token)
27             } else {
28                 // 当前操作符优先级小于上一个，上一个应该可以直接进行运算
29                 for (token == "+" || token == "-") && len(operatorStack) > 0 && operatorStack[len(operatorStack)-1] == "*" {
30                     n2 := numberStack[len(numberStack)-1]
31                     numberStack = numberStack[:len(numberStack)-1]
32                     n1 := numberStack[len(numberStack)-1]
33                     numberStack = numberStack[:len(numberStack)-1]
34                     operatorStack = operatorStack[:len(operatorStack)-1]
35                     numberStack = append(numberStack, n1*n2)
36                 }
37                 operatorStack = append(operatorStack, token)
38             }
39         } else {
40             val, _ := strconv.Atoi(token)
41             numberStack = append(numberStack, val)
42         }
43     }
44 }
```

```

45
46     for len(operatorStack) > 0 {
47         oper := operatorStack[len(operatorStack)-1]
48         operatorStack = operatorStack[:len(operatorStack)-1]
49         n2 := numberStack[len(numberStack)-1]
50         numberStack = numberStack[:len(numberStack)-1]
51         n1 := numberStack[len(numberStack)-1]
52         numberStack = numberStack[:len(numberStack)-1]
53         res := 0
54         if oper == "+" {
55             res = n1 + n2
56         } else if oper == "-" {
57             res = n1 - n2
58         } else {
59             res = n1 * n2
60         }
61         numberStack = append(numberStack, res)
62     }
63     return numberStack[0]
64 }
65
66 func main() {
67     reader := bufio.NewReader(os.Stdin)
68     input, _ := reader.ReadString('\n')
69     input = strings.TrimSpace(input)
70     n := len(input)
71     start := 0
72     res := []string{}
73     maxlen := 0
74
75     for start < n {
76         end := start
77         // 表达式以数字开始
78         if !('0' <= input[end] && input[end] <= '9') {
79             start = end + 1
80             continue
81         }
82         // 存储合法的表达式片段
83         stk := []string{}
84         for end < n {
85             tmp := ""
86             if '0' <= input[end] && input[end] <= '9' {
87                 tmp += string(input[end])
88                 for end+1 < n && '0' <= input[end+1] && input[end+1] <= '9' {
89                     end++
90                     tmp += string(input[end])
91                 }
92             }

```

```

    } else if input[end] == '*' || input[end] == '+' || input[end] ==
93  '-' {
94      tmp += string(input[end])
95  } else {
96      break
97  }
98 // 判断是否合法
99  if isOperator(tmp) {
100     if len(stk) == 0 {
101         break
102     }
103     top := stk[len(stk)-1]
104     if isOperator(top) {
105         break
106     }
107     stk = append(stk, tmp)
108 } else {
109     if len(stk) == 0 || isOperator(stk[len(stk)-1]) {
110         stk = append(stk, tmp)
111     } else {
112         break
113     }
114 }
115 end++
116 }
117

118 // 栈中如果最后为运算符直接移除
119 if len(stk) > 0 && isOperator(stk[len(stk)-1]) {
120     stk = stk[:len(stk)-1]
121 }
122

123 // 大于3个这样才会构成一个合法的表达式
124 if len(stk) >= 3 {
125     tmp := []string{}
126     currentLen := 0
127     for i := len(stk) - 1; i >= 0; i-- {
128         top := stk[i]
129         currentLen += len(top)
130         tmp = append(tmp, top)
131     }
132     if currentLen > maxLen {
133         // 反转
134         for i, j := 0, len(tmp)-1; i < j; i, j = i+1, j-1 {
135             tmp[i], tmp[j] = tmp[j], tmp[i]
136         }
137         res = tmp
138         maxLen = currentLen
139     }

```

```
140      }
141      start = end + 1
142  }
143
144 // 没有合法表达式
145 if maxLen == 0 {
146     fmt.Println(0)
147 } else {
148     // 计算值
149     result := calc(res)
150     fmt.Println(result)
151 }
```

来自: 华为OD机试 2025C卷 – 提取字符串中最长数学表达式 (C++ & Python & JAVA & JS & GO)\_  
华为od机试 – 提取字符串中的最长合法简单数学表达式-CSDN博客

来自: 华为OD机试 2025C卷 – 提取字符串中最长数学表达式 (C++ & Python & JAVA & JS & GO)\_  
华为od机试 – 提取字符串中的最长合法简单数学表达式-CSDN博客

# 华为OD上机考试 2025C卷 - 斗地主之顺子 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 斗地主之顺子

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

华为OD机试 2025C卷 100分题型

### 题目描述

在斗地主扑克牌游戏中，扑克牌由小到大的顺序为：3,4,5,6,7,8,9,10,J,Q,K,A,2，玩家可以出的扑克牌阵型有：单张、对子、顺子、飞机、炸弹等。

其中顺子的出牌规则为：由至少5张由小到大连续递增的扑克牌组成，且不能包含2。

例如：{3,4,5,6,7}、{3,4,5,6,7,8,9,10,J,Q,K,A}都是有效的顺子；而{J,Q,K,A,2}、{2,3,4,5,6}、{3,4,5,6}、{3,4,5,6,8}等都不是顺子。

给定一个包含13张牌的数组，如果有满足出牌规则的顺子，请输出顺子。

如果存在多个顺子，请每行输出一个顺子，且需要按顺子的第一张牌的大小（必须从小到大）依次输出。

如果没有满足出牌规则的顺子，请输出No。

### 输入描述

13张任意顺序的扑克牌，每张扑克牌数字用空格隔开，每张扑克牌的数字都是合法的，并且不包括大小王：

2 9 J 2 3 4 K A 7 9 A 5 6

不需要考虑输入为异常字符的情况

### 输出描述

组成的顺子，每张扑克牌数字用空格隔开：

3 4 5 6 7

### 示例1

#### 输入

```
1 2 9 J 2 3 4 K A 7 9 A 5 6
```

#### 输出

Plain Text

▼

Plain Text |

1 3 4 5 6 7

## 说明

| 13张牌中，可以组成的顺子只有1组：3 4 5 6 7。

## 示例2

### 输入

▼

Plain Text |

1 2 9 J 10 3 4 K A 7 Q A 5 6

### 输出

▼

Plain Text |

1 3 4 5 6 7  
2 9 10 J Q K A

## 说明

| 13张牌中，可以组成2组顺子，从小到大分别为：3 4 5 6 7 和 9 10 J Q K A

## 示例3

### 输入

▼

Plain Text |

1 2 9 9 9 3 4 K A 10 Q A 5 6

### 输出

▼

Plain Text |

1 No

## 说明

| 13张牌中，无法组成顺子。

## 题解

思路： 贪心

1. 定义牌面进行数值的映射，方便后续进行贪心拼接顺子比较。特殊点 2 映射成16，因为顺子中不能包含2.
2. 对输入的牌根据映射值进行升序排序。
3. 贪心尝试拼接顺子，贪心逻辑如下：当遍历到一张牌是尽量和之前尝试顺子进行匹配，能和之前的顺子连接，则直接拼接到后面。无法与之前的顺子进行匹配，则将这张牌作为一个新顺子的起始牌。
4. 输出结果,因为构建顺子的时候本身就是从小到大存储的，直接按照顺序输出所有长度  $\geq 5$  的顺子即可。

C++

```
1 #include <cstdio>
2 #include <ctime>
3 #include<iostream>
4 #include<vector>
5 #include<string>
6 #include <utility>
7 #include <sstream>
8 #include<algorithm>
9 #include<map>
10 using namespace std;
11
12
13 // 通用 split 函数
14 vector<string> split(const string& str, const string& delimiter) {
15     vector<string> result;
16     size_t start = 0;
17     size_t end = str.find(delimiter);
18     while (end != string::npos) {
19         result.push_back(str.substr(start, end - start));
20         start = end + delimiter.length();
21         end = str.find(delimiter, start);
22     }
23     // 添加最后一个部分
24     result.push_back(str.substr(start));
25     return result;
26 }
27
28
29
30 int main() {
31     // 定义牌面和值的映射
32     map<string, int> card_to_number = {
33         {"3", 3}, {"4", 4}, {"5", 5}, {"6", 6}, {"7", 7},
34         {"8", 8}, {"9", 9}, {"10", 10}, {"J", 11}, {"Q", 12},
35         {"K", 13}, {"A", 14}, {"2", 16}
36     };
37     string s = "";
38     getline(cin, s);
39     vector<string> cards = split(s, " ");
40     // 对输入的牌根据映射值进行升序排序
41     sort(cards.begin(), cards.end(), [&card_to_number](const string& a, const string& b) {
42         return card_to_number[a] < card_to_number[b];
43     });
44 }
```

```

45     vector<vector<string>> res;
46     vector<string> tmp;
47     tmp.push_back(cards[0]);
48     res.push_back(tmp);
49
50     for (int i = 1; i < cards.size(); i++) {
51         bool is_match = false;
52         for (auto &tmp : res) {
53             // 优先选择之前顺子连接
54             if (card_to_number[cards[i]] - card_to_number[tmp.back()] ==
55                 1) {
56                 is_match = true;
57                 tmp.push_back(cards[i]);
58                 break;
59             }
60         // 作为新顺子的起始牌
61         if (!is_match) {
62             vector<string> newItem;
63             newItem.push_back(cards[i]);
64             res.push_back(newItem);
65         }
66     }
67
68     bool is_match = false;
69     for (int i = 0; i < res.size(); i++) {
70         if (res[i].size() >= 5) {
71             is_match = true;
72             for (int j = 0; j < res[i].size(); j++) {
73                 if (j != 0) {
74                     cout << " ";
75                 }
76                 cout << res[i][j];
77             }
78             cout << endl;
79         }
80     }
81
82     if (!is_match) {
83         cout << "No" << endl;
84     }
85     return 0;
86 }
```

## Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         String s = scanner.nextLine();
7         scanner.close();
8
9         // 定义牌面和值的映射
10        Map<String, Integer> cardToNumber = new HashMap<>();
11        String[] cardOrder = {"3", "4", "5", "6", "7", "8", "9", "10",
12        "J", "Q", "K", "A", "2"};
13        for (int i = 0; i < cardOrder.length; i++) {
14            cardToNumber.put(cardOrder[i], i + 3);
15        }
16        // 2不能组成顺子
17        cardToNumber.put("2", 16);
18
19        // 解析输入并排序
20        String[] cards = s.split(" ");
21        Arrays.sort(cards, Comparator.comparingInt(cardToNumber::get));
22
23        // 存储连续的牌组
24        List<List<String>> res = new ArrayList<>();
25        List<String> tmp = new ArrayList<>();
26        tmp.add(cards[0]);
27        res.add(tmp);
28
29        for (int i = 1; i < cards.length; i++) {
30            boolean isMatch = false;
31            // 优先选择之前顺子连接
32            for (List<String> seq : res) {
33                if (cardToNumber.get(cards[i]) - cardToNumber.get(seq.get(
34                    seq.size() - 1)) == 1) {
35                    isMatch = true;
36                    seq.add(cards[i]);
37                    break;
38                }
39            }
40            // 作为新顺子的起始牌
41            if (!isMatch) {
42                List<String> newSeq = new ArrayList<>();
43                newSeq.add(cards[i]);
44                res.add(newSeq);
45            }
46        }
47    }
48}
```

```
44    }
45
46    // 输出符合条件的顺子
47    boolean isMatch = false;
48    for (List<String> seq : res) {
49        if (seq.size() >= 5) {
50            isMatch = true;
51            System.out.println(String.join(" ", seq));
52        }
53    }
54
55    if (!isMatch) {
56        System.out.println("No");
57    }
58}
59}
```

## Python

```
1 def main():
2     import sys
3     input_data = sys.stdin.readline().strip()
4
5     # 定义牌面和值的映射
6     card_order = ["3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K",
7       "A", "2"]
8     card_to_number = {card: i + 3 for i, card in enumerate(card_order)}
9     # 2不能组成顺子
10    card_to_number["2"] = 16
11
12    # 解析输入并排序
13    cards = input_data.split()
14    cards.sort(key=lambda x: card_to_number[x])
15
16    # 存储连续的牌组
17    res = [[cards[0]]]
18
19    for i in range(1, len(cards)):
20        is_match = False
21        # 优先选择之前顺子连接
22        for seq in res:
23            if card_to_number[cards[i]] - card_to_number[seq[-1]] == 1:
24                is_match = True
25                seq.append(cards[i])
26                break
27            # 作为新顺子的起始牌
28        if not is_match:
29            res.append([cards[i]])
30
31    # 输出符合条件的最长连续牌组
32    found = False
33    for seq in res:
34        if len(seq) >= 5:
35            found = True
36            print(" ".join(seq))
37
38    if not found:
39        print("No")
40
41 if __name__ == "__main__":
42     main()
```

# JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 // 定义牌面和值的映射
9 const cardOrder = ["3", "4", "5", "6", "7", "8", "9", "10", "J", "Q",
10 "K", "A", "2"];
11 const cardToNumber = {};
12 cardOrder.forEach((card, index) => {
13     cardToNumber[card] = index + 3;
14 });
15 // 2不能组成顺子
16 cardToNumber["2"] = 16;
17
18 rl.on("line", (input) => {
19     let cards = input.trim().split(" ").sort((a, b) => cardToNumber[a] - c
ardToNumber[b]);
20
21     let res = [[cards[0]]];
22
23     for (let i = 1; i < cards.length; i++) {
24         let isMatch = false;
25         // 优先选择之前顺子连接
26         for (let seq of res) {
27             if (cardToNumber[cards[i]] - cardToNumber[seq[seq.length - 1]] === 1) {
28                 isMatch = true;
29                 seq.push(cards[i]);
30                 break;
31             }
32         }
33         // 作为新顺子的起始牌
34         if (!isMatch) {
35             res.push([cards[i]]);
36         }
37     }
38     let found = false;
39     for (let seq of res) {
40         if (seq.length >= 5) {
41             found = true;
42             console.log(seq.join(" "));
43         }
44     }
45 }
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
788
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
848
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
918
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1167
1168
1169
1170
1171
1172
1173
1174
1175
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1185
1186
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1385
1386
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1485
1486
1487
1488
1489
1489
1490
1491
1492
1493
1494
1494
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1577
1578
1579
1580
1581
1582
1583
1584
1585
1585
1586
1587
1588
1589
1589
1590
1591
1592
1593
1594
1594
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1677
1678
1679
1680
1681
1682
1683
1684
1685
1685
1686
1687
1688
1689
1689
1690
1691
1692
1693
1694
1694
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1779
1780
1781
1782
1783
1784
1785
1785
1786
1787
1788
1789
1789
1790
1791
1792
1793
1794
1794
1795
1796
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1880
1881
1882
1883
1884
1885
1885
1886
1887
1888
1889
1889
1890
1891
1892
1893
1894
1894
1895
1896
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1980
1981
1982
1983
1984
1985
1985
1986
1987
1988
1989
1989
1990
1991
1992
1993
1994
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2080
2081
2082
2083
2084
2085
2085
2086
2087
2088
2089
2089
2090
2091
2092
2093
2094
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2177
2178
2179
2180
2181
2182
2183
2184
2185
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2208
2209
2210
2211
2212
2213
2214
2215
2216
2
```

```
43      }
44    }
45
46    if (!found) {
47      console.log("No");
48    }
49
50    rl.close();
51  });
});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strings"
9 )
10
11 // 定义牌面和值的映射
12 var cardToNumber = map[string]int{
13     "3": 3, "4": 4, "5": 5, "6": 6, "7": 7,
14     "8": 8, "9": 9, "10": 10, "J": 11, "Q": 12,
15     "K": 13, "A": 14, "2": 16,
16 }
17
18 func main() {
19     reader := bufio.NewReader(os.Stdin)
20     s, _ := reader.ReadString('\n')
21     s = strings.TrimSpace(s)
22
23     // 解析输入并排序
24     cards := strings.Split(s, " ")
25     sort.Slice(cards, func(i, j int) bool {
26         return cardToNumber[cards[i]] < cardToNumber[cards[j]]
27     })
28
29     // 存储连续的牌组
30     var res [][]string
31     res = append(res, []string{cards[0]})
32
33     for i := 1; i < len(cards); i++ {
34         isMatch := false
35             // 优先选择之前顺子连接
36         for j := range res {
37             if cardToNumber[cards[i]]-cardToNumber[res[j][len(res[j])-1]] == 1 {
38                 isMatch = true
39                 res[j] = append(res[j], cards[i])
40                 break
41             }
42         }
43             // 作为新顺子的起始牌
44         if !isMatch {
45             res = append(res, []string{cards[i]})
```

```
46     }
47 }
48
49 // 输出符合条件的最长连续牌组
50 found := false
51 for _, seq := range res {
52     if len(seq) >= 5 {
53         found = true
54         fmt.Println(strings.Join(seq, " "))
55     }
56 }
57
58 if !found {
59     fmt.Println("No")
60 }
61 }
```

| 来自: 华为OD上机考试 2025C卷 – 斗地主之顺子 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机考2025C卷 - 数字序列比大小(C++ & Python & JAVA & JS & GO)-CSDN博客

## 数字序列比大小

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

华为OD机试2025C卷 200分题型

### 题目描述

A, B两个人玩一个数字比大小的游戏，在游戏前，两个人会拿到相同长度的两个数字序列，两个数字序列不相同的，且其中的数字是随机的。

A, B各自从数字序列中挑选出一个数字进行[大小比较](#)，赢的人得1分，输的人扣1分，相等则各自的分数不变。用过的数字需要丢弃。

求A可能赢B的最大分数。

### 输入描述

输入数据的第1个数字表示数字序列的长度N，后面紧跟着两个长度为N的数字序列。

### 输出描述

A可能赢B的最大分数

### 备注

1. 这里要求计算A可能赢B的最大分数，不妨假设，A知道B的数字序列，且总是B先挑选数字并明示。
2. 可以采用贪心策略，能赢的一定要赢，要输的尽量减少损失。

### 用例1

#### 输入

```
Plain Text |  
1 3  
2 4 8 10  
3 3 6 4
```

#### 输出

```
1 3
```

Plain Text

## 说明

```
1 输入数据第1个数字表示数字序列长度为3，后面紧跟着两个长度为3的数字序列。
2 序列A: 4 8 10
3 序列B: 3 6 4
4 A可以赢的最大分数是3。获得该分数的比大小过程可以是：
5 1) A: 4   B: 3
6 2) A: 8   B: 6
7 3) A: 10  B: 4
```

Plain Text

## 题解

思路: 田忌赛马问题 : 首先将分数进行从小到大排序, 分别使用双指针指向两人最大值和最小值下标, 从两个最大值处开始比较, 指针移动规律分为以下三种情况:

- $a[a_{right}] > b[b_{right}]$ : a可以直接获胜, `res += 1, a_{right} -= 1, b_{right} -= 1`
- $a[a_{right}] < b[b_{right}]$ : a肯定会输一场, 选择将最小值损失掉, `a_{left} += 1, b_{right} -= 1, res -= 1`
- $a[a_{right}] == b[b_{right}]$ : 这时候需要考虑最小值的情况: 举两个例子a [3 4] b [2 4] / a [2 5] b[3 5]
  - $a[a_{left}] > b[b_{left}]$  情况下, 此时应该移动先考虑左指针, `a_{left} += 1, b_{left} += 1, res += 1`
  - $a[a_{left}] \leq b[b_{left}]$  情况下, a应该使用最小值去对抗b的最大值。

C++

```

1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 using namespace std;
8
9 int main() {
10     int n;
11     cin >> n;
12     vector<int> aScores(n), bScores(n);
13     for (int i = 0; i < n; i++) {
14         cin >> aScores[i];
15     }
16     for (int i = 0; i < n; i++) {
17         cin >> bScores[i];
18     }
19     sort(aScores.begin(), aScores.end());
20     sort(bScores.begin(), bScores.end());
21
22     int leftA = 0, rightA = n - 1;
23     int leftB = 0, rightB = n - 1;
24     int res = 0;
25
26     while (leftB <= rightB) {
27         // 最大值相比, a > b 则直接比
28         if (aScores[rightA] > bScores[rightB]) {
29             rightA--;
30             rightB--;
31             res += 1;
32         // 最大值相比, a < b, 无论如何都要输, 选择损失最小数
33         } else if (aScores[rightA] < bScores[rightB]) {
34             rightB--;
35             leftA++;
36             res -=1;
37         } else {
38             // 相等情况下 , a最小值大于b最小值, 先考虑左指针移动
39             if (aScores[leftA] > bScores[leftB]) {
40                 res+=1;
41                 leftA++;
42                 leftB++;
43             // 相等情况下, a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
44             }else {
45                 // a [1 1 1] b[1 1 1] 考虑这个情况

```

```
46             if (bScores[rightB] > aScores[leftA]) {
47                 res -=1;
48             }
49             leftA++;
50             rightB--;
51         }
52     }
53 }
54 cout << res;
55 return 0;
56 }
```

## JAVA

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int n = Integer.parseInt(sc.nextLine());
9         int[] a = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::pa
rseInt).toArray();
10        int[] b = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::pa
rseInt).toArray();
11
12        int result = handle(n, a, b);
13        System.out.println(result);
14    }
15
16    public static int handle(int n, int[] a, int[] b) {
17        Arrays.sort(a);
18        Arrays.sort(b);
19
20        int la = 0; // 指向a的最小值index
21        int ra = n - 1; // 指向a的最大值index
22
23        int lb = 0; // 指向b的最小值index
24        int rb = n - 1; // 指向b的最大值index
25
26        int ans = 0; // 记录a可以获得的分数
27
28        while (la <= ra) {
29            if (a[ra] > b[rb]) {
30                // a最大值 > b的最大值 直接比
31                ans += 1;
32                ra--;
33                rb--;
34            } else if (a[ra] < b[rb]) {
35                // a最大值 < b的最大值, 损失掉a的最小值, 保留最大值
36                ans -= 1;
37                la++;
38                rb--;
39            } else {
40                // 相等情况下 , a最小值大于b最小值, 先考虑左指针移动
41                if (a[la] > b[lb]) {
42                    ans += 1;
43                    la++;
```

```
44     lb++;
45     // 相等情况下，a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
46 } else {
47     // a [1 1 1] b[1 1 1] 考虑这个情况
48     if (b[rb] > a[la]) ans -= 1;
49     la++;
50     rb--;
51 }
52 }
53 }
54 return ans;
55 }
56 }
57 }
```

## Python

```
1 # 输入获取
2 n = int(input())
3 a = list(map(int, input().split())) # A的数组
4 b = list(map(int, input().split())) # B的数组
5
6
7 # 算法入口
8 def handle():
9     a.sort()
10    b.sort()
11
12    la = 0 # 指向a的最小值index
13    ra = n - 1 # 指向a的最大值index
14
15    lb = 0 # 指向b的最小值index
16    rb = n - 1 # 指向b的最大值index
17
18    ans = 0 # 记录a可以获得的分数
19
20    while la <= ra:
21        if a[ra] > b[rb]:
22            # a最大值 > b的最大值 直接比
23            ans += 1
24            ra -= 1
25            rb -= 1
26        elif a[ra] < b[rb]:
27            # a最大值 < b的最大值，损失掉a的最小值，保留最大值
28            ans -= 1
29            la += 1
30            rb -= 1
31        else:
32            # 相等情况下，a最小值大于b最小值，先考虑左指针移动
33            if a[la] > b[lb]:
34                # 优先保证a可以赢的
35                ans += 1
36                la += 1
37                lb += 1
38            # 相等情况下，a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
39        else:
40            # a [1 1 1] b[1 1 1] 考虑这个情况
41            if b[rb] > a[la]:
42                ans -= 1
43                la += 1
44                rb -= 1
45
```

```
46     return ans
47
48
49 # 算法调用
50 print(handle())
```

## JavaScript

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11     lines.push(line);
12
13     if (lines.length == 3) {
14         const n = parseInt(lines[0]);
15         const a = lines[1].split(" ").map(Number);
16         const b = lines[2].split(" ").map(Number);
17
18         console.log(handle(n, a, b));
19
20         lines.length = 0;
21     }
22 });
23 });
24
25 function handle(n, a, b) {
26     a.sort((a, b) => a - b);
27     b.sort((a, b) => a - b);
28
29     let la = 0; // 指向a的最小值index
30     let ra = n - 1; // 指向a的最大值index
31
32     let lb = 0; // 指向b的最小值index
33     let rb = n - 1; // 指向b的最大值index
34
35     let ans = 0; // 记录a可以获得的分数
36
37     while (la <= ra) {
38         if (a[ra] > b[rb]) {
39             // a最大值 > b的最大值 直接比
40             ans += 1;
41             ra--;
42             rb--;
43         } else if (a[ra] < b[rb]) {
44             // 最小消耗最大
45             ans -= 1;
```

```
46     la++;
47     rb--;
48 } else {
49     // 相等情况下，a最小值大于b最小值，先考虑左指针移动
50     if (a[la] > b[lb]) {
51         ans += 1;
52         la++;
53         lb++;
54     // 相等情况下，a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
55     } else {
56         // a [1 1 1] b[1 1 1] 考虑这个情况
57         if (b[rb] > a[la]) ans -= 1;
58         la++;
59         rb--;
60     }
61 }
62 }
63
64     return ans;
65 }
```

## Go

```
1 package main
2
3 import (
4     "fmt"
5     "sort"
6 )
7
8 func main() {
9     var n int
10    fmt.Scan(&n)
11
12    aScores := make([]int, n)
13    bScores := make([]int, n)
14
15    for i := 0; i < n; i++ {
16        fmt.Scan(&aScores[i])
17    }
18    for i := 0; i < n; i++ {
19        fmt.Scan(&bScores[i])
20    }
21
22    sort.Ints(aScores)
23    sort.Ints(bScores)
24    // 双指针分别最大值和最小值下
25    leftA, rightA := 0, n-1
26    leftB, rightB := 0, n-1
27    res := 0
28
29    for leftB <= rightB {
30        // a最大值 > b最大值 直接比
31        if aScores[rightA] > bScores[rightB] {
32            rightA--
33            rightB--
34            res++
35            // 肯定会输，选择最小值来输
36        } else if aScores[rightA] < bScores[rightB] {
37            rightB--
38            leftA++
39            res--
40        } else {
41            // 相等情况下，a最小值大于b最小值，先考虑左指针移动
42            if aScores[leftA] > bScores[leftB] {
43                res++
44                leftA++
45                leftB++
```

```
46          // 相等情况下，a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
47      } else {
48          // a [1 1 1] b[1 1 1] 考虑这个情况 最小值和最大值相等情况了
49          if bScores[rightB] > aScores[leftA] {
50              res--
51          }
52          leftA++
53          rightB--
54      }
55  }
56 }
57 fmt.Println(res)
58 }
```

| 来自: 华为OD 机考 2025C卷 – 数字序列比大小 (C++ & Python & JAVA & JS & GO)–CSDN博客

# 华为OD机考 2025C卷 - 内存资源分配-CSDN博客

## 内存资源分配

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

华为OD机试2025C卷 100分题型

### 题目描述

有一个简易内存池，内存按照大小粒度分类，每个粒度有若干个可用内存资源，用户会进行一系列内存申请，需要按需分配内存池中的资源返回申请结果成功失败列表。

分配规则如下：

- 分配的内存要大于等于内存的申请量，存在满足需求的内存就必须分配，优先分配粒度小的，但内存不能拆分使用；
- 需要按申请顺序分配，先申请的先分配，有可用内存分配则申请结果为true；
- 没有可用则返回false。

注意：不考虑内存释放

### 输入描述

输入为两行字符串：

第一行为内存池资源列表，包含内存粒度数据信息，粒度数据间用逗号分割

- 一个粒度信息内用冒号分割，冒号前为内存粒度大小，冒号后为数量
- 资源列表不大于1024
- 每个粒度的数量不大于4096

第二行为申请列表，申请的内存大小间用逗号分割

- 申请列表不大于100000

如：

64:2,128:1,32:4,1:128

50,36,64,128,127

### 输出描述

输出为内存池分配结果

如true,true,true,false,false

### 示例1

#### 输入

```
Plain Text |  
1 64:2,128:1,32:4,1:128  
2 50,36,64,128,127
```

## 输出

```
Plain Text |  
1 true,true,true,false,false
```

## 说明

内存池资源包含：64K共2个、128K共1个、32K共4个、1K共128个的内存资源；

针对50,36,64,128,127的[内存申请]序列，分配的内存依次是：64,64,128,NULL,NULL,

第三次申请内存时已经将128分配出去，因此输出结果是：

true,true,true,false,false

## 题解

思路： 模拟

- 解析输入的内存池资源，使用 哈希 数据结构统计出每个大小内存的数量。
- 对于每个申请内存的请求，尝试从哈希表中从小到大找到第一个满足要求的内存资源大小 key
  - 如果不存在，直接输出false。
  - 如果存在，输出true，并更新哈希表中 key 的数量，如果数量变为0，则从哈希表中移除该key。

## C++

```
1 #include <iostream>
2 #include <map>
3 #include <sstream>
4 #include <vector>
5
6 using namespace std;
7
8 int main() {
9     // 处理输入
10    string memoryInfo, applyList;
11    cin >> memoryInfo >> applyList;
12
13    // 使用 map 记录内存大小及对应数量
14    map<int, int> memoryMap;
15    stringstream ss(memoryInfo);
16    string info;
17    while (getline(ss, info, ',')) {
18        int colonIndex = info.find(":");
19        int size = stoi(info.substr(0, colonIndex));
20        int count = stoi(info.substr(colonIndex + 1));
21        memoryMap[size] += count; // 累加相同大小的内存块
22    }
23
24    // 申请信息
25    vector<int> applyMemoryList;
26    stringstream ss2(applyList);
27    string apply;
28    while (getline(ss2, apply, ',')) {
29        applyMemoryList.push_back(stoi(apply));
30    }
31
32    // 分配内存
33    vector<bool> resultList;
34    for (int applyMemory : applyMemoryList) {
35        // 在 map 中查找第一个大于等于 applyMemory 的内存块
36        auto it = memoryMap.lower_bound(applyMemory);
37        if (it != memoryMap.end()) {
38            // 找到了合适的内存块，减少该内存块的数量
39            it->second--;
40            if (it->second == 0) {
41                memoryMap.erase(it); // 如果数量为 0，则移除该大小的内存块
42            }
43            resultList.push_back(true);
44        } else {
45            resultList.push_back(false);
46        }
47    }
48 }
```

```
46         }
47     }
48
49     // 输出结果
50     for (size_t i = 0; i < resultList.size(); i++) {
51         cout << (resultList[i] ? "true" : "false");
52         if (i != resultList.size() - 1) {
53             cout << ",";
54         }
55     }
56
57     return 0;
58 }
```

## Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6         String memoryInfo = scanner.next();
7         String applyList = scanner.next();
8         scanner.close();
9
10        // 使用 TreeMap 记录内存大小及其数量（默认按 key 升序排序）
11        TreeMap<Integer, Integer> memoryMap = new TreeMap<>();
12        for (String info : memoryInfo.split(",")) {
13            String[] parts = info.split(":");
14            int size = Integer.parseInt(parts[0]);
15            int count = Integer.parseInt(parts[1]);
16            memoryMap.put(size, memoryMap.getOrDefault(size, 0) + count);
17        }
18
19        // 读取申请信息
20        List<Integer> applyMemoryList = new ArrayList<>();
21        for (String apply : applyList.split(",")) {
22            applyMemoryList.add(Integer.parseInt(apply));
23        }
24
25        // 分配内存
26        List<String> resultList = new ArrayList<>();
27        for (int applyMemory : applyMemoryList) {
28            Map.Entry<Integer, Integer> entry = memoryMap.ceilingEntry(app
lyMemory);
29            if (entry != null) {
30                int key = entry.getKey();
31                if (entry.getValue() == 1) {
32                    memoryMap.remove(key);
33                } else {
34                    memoryMap.put(key, entry.getValue() - 1);
35                }
36                resultList.add("true");
37            } else {
38                resultList.add("false");
39            }
40        }
41
42        // 输出结果
43        System.out.println(String.join(",", resultList));
44    }
```

```
45 }
```

## Python

```
Plain Text |
```

```
1 import sys
2 from collections import defaultdict
3
4 # 读取输入
5 memory_info, apply_list = sys.stdin.read().split()
6
7 # 解析内存信息，使用字典模拟有序 map
8 memory_map = defaultdict(int)
9 for info in memory_info.split(","):
10     size, count = map(int, info.split(":"))
11     memory_map[size] += count
12
13 # 申请信息
14 apply_memory_list = list(map(int, apply_list.split(",")))
15
16 # 分配内存
17 result_list = []
18 sorted_keys = sorted(memory_map.keys()) # 确保 key 按升序排列
19
20 for apply_memory in apply_memory_list:
21     for key in sorted_keys:
22         if key >= apply_memory and memory_map[key] > 0:
23             memory_map[key] -= 1
24             if memory_map[key] == 0:
25                 sorted_keys.remove(key) # 内存用完，从列表中移除
26                 result_list.append("true")
27                 break
28     else:
29         result_list.append("false")
30
31 # 输出结果
32 print(",".join(result_list))
```

## JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  const inputLines = [];
9
10 rl.on("line", (line) => {
11     inputLines.push(line.trim()); // 去掉首尾空格
12     if (inputLines.length === 2) {
13         rl.close();
14     }
15 });
16
17 rl.on("close", () => {
18     const [memoryInfo, applyList] = inputLines;
19
20     // 解析内存信息
21     const memoryMap = new Map();
22     memoryInfo.split(",").forEach(info => {
23         let [size, count] = info.split(":").map(Number);
24         memoryMap.set(size, (memoryMap.get(size) || 0) + count);
25     });
26
27     // 按 key 升序排列
28     const sortedKeys = [...memoryMap.keys()].sort((a, b) => a - b);
29
30     // 解析申请的内存列表
31     const applyMemoryList = applyList.split(",").map(Number);
32     const resultList = [];
33
34     for (let applyMemory of applyMemoryList) {
35         let allocated = false;
36         for (let key of sortedKeys) {
37             if (key >= applyMemory && memoryMap.get(key) > 0) {
38                 memoryMap.set(key, memoryMap.get(key) - 1);
39                 if (memoryMap.get(key) === 0) {
40                     sortedKeys.splice(sortedKeys.indexOf(key), 1);
41                 }
42                 resultList.push("true");
43                 allocated = true;
44                 break;
45             }
46         }
47     }
48 }
```

```
46      }
47      if (!allocated) {
48          resultList.push("false");
49      }
50  }
51
52  console.log(resultList.join(","));
53});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10    )
11
12 func main() {
13     scanner := bufio.NewScanner(os.Stdin)
14     lines := []string{}
15
16     // 读取两行输入
17     for scanner.Scan() {
18         lines = append(lines, strings.TrimSpace(scanner.Text()))
19         if len(lines) == 2 {
20             break
21         }
22     }
23
24
25     memoryInfo := lines[0]
26     applyList := lines[1]
27
28     // 解析内存信息
29     memoryMap := make(map[int]int)
30     for _, info := range strings.Split(memoryInfo, ",") {
31         parts := strings.Split(info, ":")
32         if len(parts) != 2 {
33             return
34         }
35         size, _ := strconv.Atoi(parts[0])
36         count, _ := strconv.Atoi(parts[1])
37         memoryMap[size] += count
38     }
39
40     // 获取内存大小的升序排列
41     keys := make([]int, 0, len(memoryMap))
42     for k := range memoryMap {
43         keys = append(keys, k)
44     }
45     sort.Ints(keys)
```

```
46
47 // 解析申请内存列表
48 applyRequests := strings.Split(applyList, ",")
49 results := make([]string, len(applyRequests))
50
51 for i, req := range applyRequests {
52     applyMemory, _ := strconv.Atoi(req)
53     allocated := false
54
55     for _, size := range keys {
56         if size >= applyMemory && memoryMap[size] > 0 {
57             memoryMap[size]--
58             if memoryMap[size] == 0 {
59                 delete(memoryMap, size) // 移除已分配完的块
60             }
61             results[i] = "true"
62             allocated = true
63             break
64         }
65     }
66
67     if !allocated {
68         results[i] = "false"
69     }
70 }
71
72 // 输出结果
73 fmt.Println(strings.Join(results, ","))
74 }
```

| 来自: 华为OD机考 2025C卷 – 内存资源分配-CSDN博客

# 华为OD机考 2025C卷 - 二维伞的雨滴效应 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 二维伞的雨滴效应

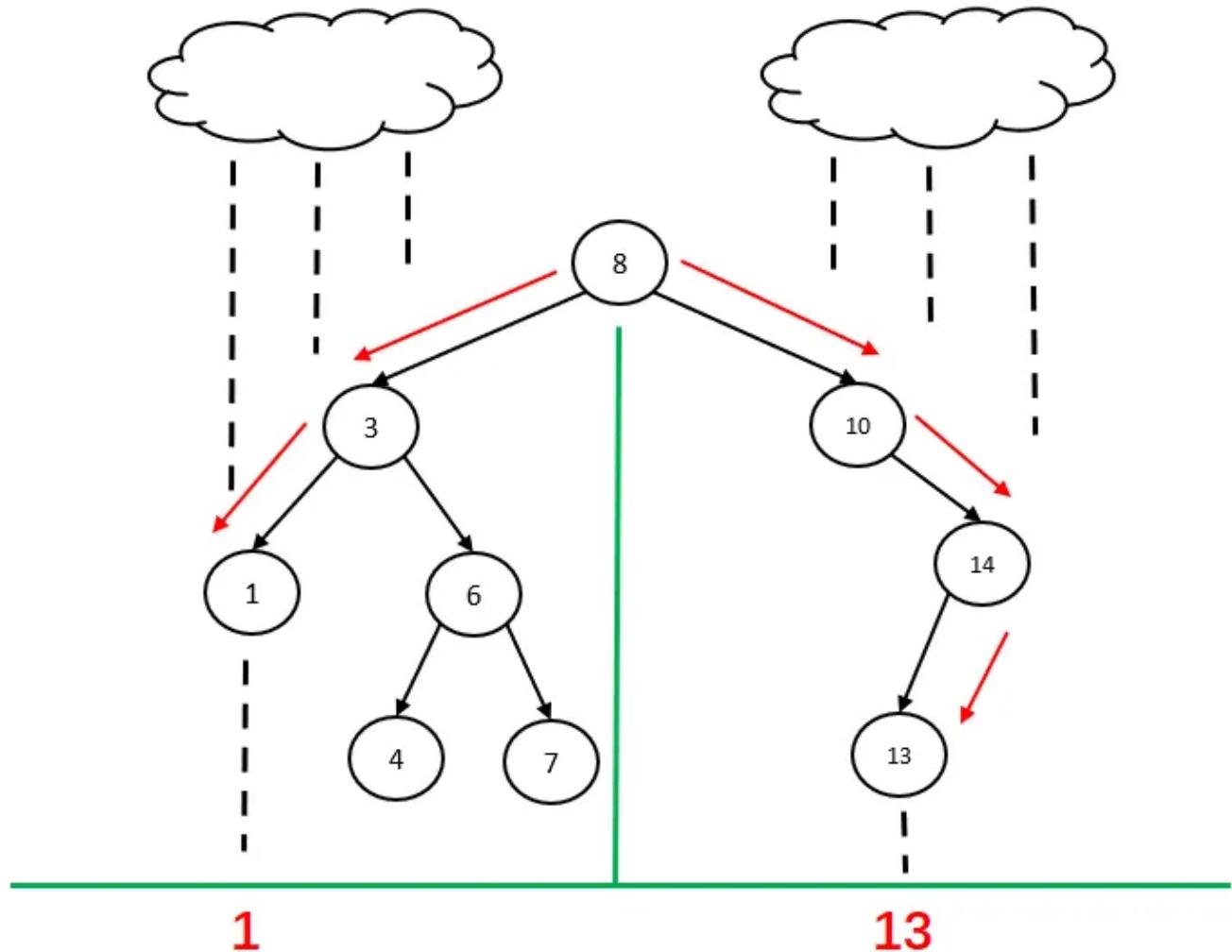
华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 200分题型

### 题目描述

普通的伞在二维平面世界中，左右两侧均有一条边，而两侧伞边最下面各有一个伞坠子，雨滴落到伞面，逐步流到伞坠处，会将伞坠的信息携带并落到地面，随着日积月累，地面会呈现伞坠的信息。

- 1、为了模拟伞状雨滴效应，用[二叉树](#)来模拟二维平面伞（如下图所示），现在输入一串正整数数组序列（不含0，数组成员至少是1个），若此数组序列是二叉搜索树的前序遍历的结果，那么请输出一个返回值1，否则输出0。
- 2、同时请将此序列构成的伞状效应携带到地面的数字信息输出来(左边伞坠信息，右边伞坠信息，详细参考示例图地面上数字)，若此树不存在左或右扇坠，则对应位置返回0。同时若非二叉排序树那么左右伞坠信息也返回0。



## 输入描述

一个通过空格分割的整数序列字符串，数组不含0，数组成员至少1个，输入的数组的任意两个数字都互不相同，最多1000个正整数，正整数值范围1~65535

## 输出描述

输出如下三个值，以空格分隔：是否二叉排序树，左侧地面呈现的伞坠数字值，右侧地面呈现的伞坠数字值。

若是二叉排序树，则输出1，否则输出0（其左右伞坠值也直接赋值0）。

若不存在左侧或者右侧伞坠值，那么对应伞坠值直接赋值0。

## 用例1

### 输入

▼

Plain Text |

```
1 8 3 1 6 4 7 10 14 13
```

## 输出

▼

Plain Text |

```
1 1 1 13
```

## 说明

| 1表示是[二叉搜索树](#)前序遍历结果，1表示左侧地面呈现的伞坠数字值，13表示右侧地面呈现的伞坠数字值

## 题解

思路：搜索二叉树问题。

- 搜索二叉树的性质：利用这个性质可以判断是否为搜索二叉树
  - a. 左子树的值肯定严格小于父节点。
  - b. 右子树的值肯定严格大于父节点。
- 找左右吊坠的规律：
  - a. 如果根节点没有左子树或者右子树，对应方向吊坠的值为0
  - b. 说一下找左吊坠值的规律，右边类似
    - i. 从根节点向下搜索不断找寻左节点，找到最左边的节点(node)。
    - ii. 如果node存在右节点，把指针调换至 `root->value` 重复1的操作。
    - iii. 重复1和2的操作，直到找到一个叶子节点，这个叶子节点的值就是结果。

## C++

```
1 #include <cstdio>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 #include<list>
9 #include<queue>
10 #include<map>
11 using namespace std;
12
13 struct Node {
14     int value;
15     Node* left;
16     Node* right;
17     Node():left(nullptr), right(nullptr){}
18     Node(int value, Node* left, Node* right):value(value),left(left),right(right) {}
19     Node(int value):value(value),left(nullptr), right(nullptr){}
20 };
21
22
23 // 通用 split 函数
24 vector<string> split(const string& str, const string& delimiter) {
25     vector<string> result;
26     size_t start = 0;
27     size_t end = str.find(delimiter);
28     while (end != string::npos) {
29         result.push_back(str.substr(start, end - start));
30         start = end + delimiter.length();
31         end = str.find(delimiter, start);
32     }
33     // 添加最后一个部分
34     result.push_back(str.substr(start));
35     return result;
36 }
37
38 bool isValid(Node* root, int start, int end, vector<int>& preOrder) {
39     if (start == end) {
40         return true;
41     }
42
43     int i = start + 1;
44     // 不断迭代找到父节点的右子树节点为止
```

```

45     while (i <= end && preOrder[i] < root->value) {
46         i++;
47     }
48
49     int j = i;
50     // 按照搜索树规则 找到右子树的终点位置
51     while (j <= end && preOrder[j] > root->value) {
52         j++;
53     }
54     // 异常 j 必须等于 end才是正常的
55     if (j <= end) {
56         return false;
57     }
58
59     // i > start + 1说明才存在左子树
60     if (i > start + 1) {
61         root->left = new Node(preOrder[start+1]);
62         //递归构建左节点的结构
63         if (!isValid(root->left, start + 1, i - 1, preOrder)) {
64             return false;
65         }
66     }
67
68     // i <= end说明才存在右子树
69     if (i <= end) {
70         root->right = new Node(preOrder[i]);
71         return isValid(root->right, i, end, preOrder);
72     }
73
74     return true;
75 }
76
77 // 获取左伞坠的值
78 int leftBottom(Node* root, int level) {
79     if (root->left != nullptr) {
80         return leftBottom(root->left, level + 1);
81     }
82     // 说明根节点没有子节点, 没有坠子, 直接返回0
83     if (level == 0) {
84         return 0;
85     }
86
87     // 存在
88     if (root->right != nullptr) {
89         return leftBottom(root->right, level + 1);
90     }
91     return root->value;
92 }

```

```
93
94 // 获取右伞坠的值
95 int rightBottom(Node* root, int level) {
96     if (root -> right != nullptr) {
97         return rightBottom(root->right, level + 1);
98     }
99     // 说明根节点没有子节点，没有坠子，直接返回0
100    if (level == 0) {
101        return 0;
102    }
103
104    // 存在右节点
105    if (root->left != nullptr) {
106        return rightBottom(root->left, level + 1);
107    }
108    return root->value;
109 }
110
111 void dfs(Node* root, vector<int>& res) {
112     if (root == nullptr) {
113         return;
114     }
115     dfs(root->left, res);
116     res.push_back(root->value);
117     dfs(root->right, res);
118 }
119
120
121 int main() {
122     string line;
123     getline(cin, line);
124     vector<string> tmp = split(line, " ");
125     int n = tmp.size();
126     vector<int> ans(n);
127     for (int i = 0; i < n; i++) {
128         ans[i] = stoi(tmp[i]);
129     }
130     Node* root = new Node(ans[0]);
131     // 不合法
132     if (!isValid(root, 0, ans.size() - 1, ans)) {
133         cout << "0 0 0";
134         return 0;
135     }
136     vector<int> res;
137     dfs(root, res);
138
139     int leftValue = leftBottom(root, 0);
140     int rightValue = rightBottom(root, 0);
```

```
141
142     cout << "1 " << leftValue << " " << rightValue;
143     return 0;
144 }
```

## JAVA

```
1 import java.util.*;
2
3 class Node {
4     int value;
5     Node left, right;
6
7     Node(int value) {
8         this.value = value;
9         this.left = null;
10        this.right = null;
11    }
12 }
13
14 public class Main {
15
16     // 判断是否是有效的二叉搜索树的前序遍历
17     static boolean isValid(Node root, int start, int end, List<Integer> preOrder) {
18         if (start == end) {
19             return true;
20         }
21
22         int i = start + 1;
23         // 找到右子树的起点
24         while (i <= end && preOrder.get(i) < root.value) {
25             i++;
26         }
27
28         int j = i;
29         // 验证右子树的所有值是否大于根节点
30         while (j <= end && preOrder.get(j) > root.value) {
31             j++;
32         }
33
34         // 若 j 未能完全遍历完，则说明不符合BST规则
35         if (j <= end) {
36             return false;
37         }
38
39         // 递归构建左子树
40         if (i > start + 1) {
41             root.left = new Node(preOrder.get(start + 1));
42             if (!isValid(root.left, start + 1, i - 1, preOrder)) {
43                 return false;
44             }
45         }
46     }
47 }
```

```
45     }
46
47     // 递归构建右子树
48     if (i <= end) {
49         root.right = new Node(preOrder.get(i));
50         return isValid(root.right, i, end, preOrder);
51     }
52
53     return true;
54 }
55
56 // 获取左伞坠
57 static int leftBottom(Node root, int level) {
58     if (root.left != null) {
59         return leftBottom(root.left, level + 1);
60     }
61     if (level == 0) {
62         return 0;
63     }
64     if (root.right != null) {
65         return leftBottom(root.right, level + 1);
66     }
67     return root.value;
68 }
69
70 // 获取右伞坠
71 static int rightBottom(Node root, int level) {
72     if (root.right != null) {
73         return rightBottom(root.right, level + 1);
74     }
75     if (level == 0) {
76         return 0;
77     }
78     if (root.left != null) {
79         return rightBottom(root.left, level + 1);
80     }
81     return root.value;
82 }
83
84 // 中序遍历
85 static void dfs(Node root, List<Integer> res) {
86     if (root == null) {
87         return;
88     }
89     dfs(root.left, res);
90     res.add(root.value);
91     dfs(root.right, res);
92 }
```

```
93
94     public static void main(String[] args) {
95         Scanner scanner = new Scanner(System.in);
96         String[] inputs = scanner.nextLine().split(" ");
97         List<Integer> ans = new ArrayList<>();
98         for (String num : inputs) {
99             ans.add(Integer.parseInt(num));
100        }
101        scanner.close();
102
103        Node root = new Node(ans.get(0));
104
105        if (!isValid(root, 0, ans.size() - 1, ans)) {
106            System.out.println("0 0 0");
107            return;
108        }
109
110        int leftValue = leftBottom(root, 0);
111        int rightValue = rightBottom(root, 0);
112
113        System.out.println("1 " + leftValue + " " + rightValue);
114    }
115 }
```

## Python

```
1 import sys
2
3 # 定义树节点
4 class Node:
5     def __init__(self, value):
6         self.value = value
7         self.left = None
8         self.right = None
9
10 # 判断是否是有效的BST前序遍历
11 def is_valid(root, start, end, pre_order):
12     if start == end:
13         return True
14
15     i = start + 1
16     # 找到右子树的起点
17     while i <= end and pre_order[i] < root.value:
18         i += 1
19
20     j = i
21     # 验证右子树的所有值是否都大于根节点
22     while j <= end and pre_order[j] > root.value:
23         j += 1
24
25     # 如果 j 没有遍历完, 说明不是合法的 BST
26     if j <= end:
27         return False
28
29     # 递归构建左子树
30     if i > start + 1:
31         root.left = Node(pre_order[start + 1])
32         if not is_valid(root.left, start + 1, i - 1, pre_order):
33             return False
34
35     # 递归构建右子树
36     if i <= end:
37         root.right = Node(pre_order[i])
38         return is_valid(root.right, i, end, pre_order)
39
40     return True
41
42 # 获取左伞坠的值 (最左端的叶子节点)
43 def left_bottom(root, level):
44     if root.left:
45         return left_bottom(root.left, level + 1)
```

```
46     if level == 0:
47         return 0
48     if root.right:
49         return left_bottom(root.right, level + 1)
50     return root.value
51
52 # 获取右伞坠的值（最右端的叶子节点）
53 def right_bottom(root, level):
54     if root.right:
55         return right_bottom(root.right, level + 1)
56     if level == 0:
57         return 0
58     if root.left:
59         return right_bottom(root.left, level + 1)
60     return root.value
61
62 # 读取输入
63 pre_order = list(map(int, sys.stdin.readline().strip().split()))
64 root = Node(pre_order[0])
65
66 # 验证是否为合法 BST
67 if not is_valid(root, 0, len(pre_order) - 1, pre_order):
68     print("0 0 0")
69 else:
70     print(f"1 {left_bottom(root, 0)} {right_bottom(root, 0)}")
```

## JavaScript

```
1 // 定义树节点
2 class Node {
3     constructor(value) {
4         this.value = value;
5         this.left = null;
6         this.right = null;
7     }
8 }
9
10 // 判断是否是有效的BST前序遍历
11 function isValid(root, start, end, preOrder) {
12     if (start === end) return true;
13
14     let i = start + 1;
15     // 找到右子树的起点
16     while (i <= end && preOrder[i] < root.value) i++;
17
18     let j = i;
19     // 验证右子树的所有值是否大于根节点
20     while (j <= end && preOrder[j] > root.value) j++;
21
22     // 若 j 未能完全遍历完，则说明不符合BST规则
23     if (j <= end) return false;
24
25     // 递归构建左子树
26     if (i > start + 1) {
27         root.left = new Node(preOrder[start + 1]);
28         if (!isValid(root.left, start + 1, i - 1, preOrder)) return false;
29     }
30
31     // 递归构建右子树
32     if (i <= end) {
33         root.right = new Node(preOrder[i]);
34         return isValid(root.right, i, end, preOrder);
35     }
36
37     return true;
38 }
39
40 // 获取左伞坠
41 function leftBottom(root, level) {
42     if (root.left) return leftBottom(root.left, level + 1);
43     if (level === 0) return 0;
44     if (root.right) return leftBottom(root.right, level + 1);
45     return root.value;
}
```

```
46     }
47
48     // 获取右伞坠
49     function rightBottom(root, level) {
50         if (root.right) return rightBottom(root.right, level + 1);
51         if (level === 0) return 0;
52         if (root.left) return rightBottom(root.left, level + 1);
53         return root.value;
54     }
55
56     // 读取输入
57     const readline = require("readline");
58     const rl = readline.createInterface({ input: process.stdin });
59
60     rl.on("line", (line) => {
61         const preOrder = line.split(" ").map(Number);
62         const root = new Node(preOrder[0]);
63
64         if (!isValid(root, 0, preOrder.length - 1, preOrder)) {
65             console.log("0 0 0");
66         } else {
67             console.log(`1 ${leftBottom(root, 0)} ${rightBottom(root, 0)}`);
68         }
69         rl.close();
70     });
});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 // 定义树节点
12 type Node struct {
13     value int
14     left  *Node
15     right *Node
16 }
17
18 // 判断是否是有效的BST前序遍历
19 func isValid(root *Node, start, end int, preOrder []int) bool {
20     if start == end {
21         return true
22     }
23
24     i := start + 1
25     // 找到右子树的起点
26     for i <= end && preOrder[i] < root.value {
27         i++
28     }
29
30     j := i
31     // 验证右子树的所有值是否大于根节点
32     for j <= end && preOrder[j] > root.value {
33         j++
34     }
35
36     // 若 j 未能完全遍历完，则说明不符合BST规则
37     if j <= end {
38         return false
39     }
40
41     // 递归构建左子树
42     if i > start+1 {
43         root.left = &Node{value: preOrder[start+1]}
44         if !isValid(root.left, start+1, i-1, preOrder) {
45             return false
46         }
47     }
48 }
```

```
46     }
47 }
48
49 // 递归构建右子树
50 if i <= end {
51     root.right = &Node{value: preOrder[i]}
52     return isValid(root.right, i, end, preOrder)
53 }
54
55     return true
56 }
57
58 // 获取左伞坠
59 func leftBottom(root *Node, level int) int {
60     if root.left != nil {
61         return leftBottom(root.left, level+1)
62     }
63     if level == 0 {
64         return 0
65     }
66     if root.right != nil {
67         return leftBottom(root.right, level+1)
68     }
69     return root.value
70 }
71
72 // 获取右伞坠
73 func rightBottom(root *Node, level int) int {
74     if root.right != nil {
75         return rightBottom(root.right, level+1)
76     }
77     if level == 0 {
78         return 0
79     }
80     if root.left != nil {
81         return rightBottom(root.left, level+1)
82     }
83     return root.value
84 }
85
86 func main() {
87     // 读取输入
88     reader := bufio.NewReader(os.Stdin)
89     line, _ := reader.ReadString('\n')
90     line = strings.TrimSpace(line)
91
92     // 解析输入为整数数组
93     strNums := strings.Split(line, " ")
```

```
94     preOrder := make([]int, len(strNums))
95     for i, str := range strNums {
96         preOrder[i], _ = strconv.Atoi(str)
97     }
98
99     root := &Node{value: preOrder[0]}
100
101    // 验证是否为合法 BST
102    if !isValid(root, 0, len(preOrder)-1, preOrder) {
103        fmt.Println("0 0 0")
104    } else {
105        fmt.Printf("1 %d %d\n", leftBottom(root, 0), rightBottom(root, 0))
106    }
107 }
```

| 来自: 华为OD机考 2025C卷 – 二维伞的雨滴效应 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机试2025B卷 - 字符串加密(C++ & Python & Java & JS & Go)-CSDN博客

## 字符串加密

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

华为OD机试2025C卷 100分题型

## 题目描述

给你一串未加密的字符串str，通过对字符串的每一个字母进行改变来实现加密，加密方式是在每一个字母str[i]偏移特定数组元素a[i]的量，数组a前三位已经赋值：a[0]=1,a[1]=2,a[2]=4。当*i>=3*时，数组元素a[i]=a[i-1]+a[i-2]+a[i-3]。例如：原文 abcde 加密后 bdgkr，其中偏移量分别是1,2,4,7,13

## 输入描述

第一行是一个整数n ( $1 \leq n \leq 1000$ )，表示有n组测试数据，每组数据包含一行，原文str（只含有小写字母， $0 < \text{长度} \leq 50$ ）。

## 输出描述

每组测试数据输出一行，表示字符串的密文

## 用例1

输入

## 输出

# 题解

思路： 模拟 实现

1. 题目说明 每一个字母`str[i]`偏移特定数组元素`a[i]`的量，说明`a[0]=1,a[1]=2,a[2]=4`，当`i>=3`时，数组元素`a[i]=a[i-1]+a[i-2]+a[i-3]`，又限制字符串的最长的长度为50.我们可以直接预处理`a[]`数组，计算出`0 - 49`的`a`的值。`a[i]=a[i-1]+a[i-2]+a[i-3]`可以简化为`2 * ans[i-1] - ans[i-4]`因为`ans[i-1] = ans[i-4] + ans[i-3] + ans[i-2]`.当然不进行简化也是可以的。注意需要存储的对应26的余数，不然int会溢出。
2. 接下来就是接收输入的多组字符串，每个字符串按照题目要求对各个字符进行偏移量转换，转换状态的方程为`'a' + ((s[i] - 'a' + ans[i]) % 26)`。并换行输出转换之后的字符串即可。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<list>
8 #include<queue>
9 using namespace std;
10
11 int main() {
12     string s;
13     vector<int> ans(50);
14     ans[0] = 1;
15     ans[1] = 2;
16     ans[2] = 4;
17     ans[3] = 7;
18     // 计算偏移量 预处理
19     for (int i = 4; i <= 49; i++) {
20         ans[i] = (2 * ans[i-1] - ans[i-4] + 26) % 26;
21     }
22     int n;
23     cin >> n;
24     while (n--) {
25         string s;
26         cin >> s;
27         string res = "";
28         for (int i = 0; i < s.size(); i++) {
29             // 计算差值
30             int diff = ((s[i] - 'a' + ans[i]) % 26);
31             char c = 'a' + diff;
32             res += c;
33         }
34         cout << res << endl;
35     }
36     return 0;
37 }
```

## JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 预计算偏移量
8         int[] ans = new int[50];
9         ans[0] = 1;
10        ans[1] = 2;
11        ans[2] = 4;
12        ans[3] = 7;
13        for (int i = 4; i < 50; i++) {
14            ans[i] = (2 * ans[i - 1] - ans[i - 4] + 26) % 26;
15        }
16
17        int n = scanner.nextInt();
18        scanner.nextLine(); // 读取换行符
19        while (n-- > 0) {
20            String s = scanner.nextLine();
21            StringBuilder res = new StringBuilder();
22            for (int i = 0; i < s.length(); i++) {
23                // 计算新字符
24                int diff = ((s.charAt(i) - 'a' + ans[i]) % 26);
25                char c = (char) ('a' + diff);
26                res.append(c);
27            }
28            System.out.println(res);
29        }
30        scanner.close();
31    }
32 }
```

## Python

```
1 import sys
2
3 def main():
4     # 预计算偏移量
5     ans = [0] * 50
6     ans[0] = 1
7     ans[1] = 2
8     ans[2] = 4
9     ans[3] = 7
10    for i in range(4, 50):
11        ans[i] = (2 * ans[i - 1] - ans[i - 4] + 26) % 26
12
13    # 读取输入
14    n = int(sys.stdin.readline().strip())
15    for _ in range(n):
16        s = sys.stdin.readline().strip()
17        res = []
18        for i in range(len(s)):
19            # 计算新字符
20            diff = (ord(s[i]) - ord('a') + ans[i]) % 26
21            res.append(chr(ord('a') + diff))
22        print("".join(res))
23
24 if __name__ == "__main__":
25     main()
```

## JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let ans = new Array(50);
9 ans[0] = 1;
10 ans[1] = 2;
11 ans[2] = 4;
12 ans[3] = 7;
13 // 预计算偏移量
14 for (let i = 4; i < 50; i++) {
15     ans[i] = (2 * ans[i - 1] - ans[i - 4] + 26) % 26;
16 }
17
18 let inputLines = [];
19 rl.on("line", (line) => {
20     inputLines.push(line.trim());
21 }).on("close", () => {
22     let n = parseInt(inputLines[0]);
23     for (let i = 1; i <= n; i++) {
24         let s = inputLines[i];
25         let res = "";
26         for (let j = 0; j < s.length; j++) {
27             let diff = ((s.charCodeAt(j) - "a".charCodeAt(0) + ans[j]) % 2
28 6);
29             let c = String.fromCharCode("a".charCodeAt(0) + diff);
30             res += c;
31         }
32         console.log(res);
33     }
34 });
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7 )
8
9 func main() {
10    // 预计算偏移量
11    ans := make([]int, 50)
12    ans[0] = 1
13    ans[1] = 2
14    ans[2] = 4
15    ans[3] = 7
16    for i := 4; i < 50; i++ {
17        ans[i] = (2*ans[i-1] - ans[i-4] + 26) % 26
18    }
19
20    // 读取输入
21    scanner := bufio.NewScanner(os.Stdin)
22    scanner.Scan()
23    var n int
24    fmt.Sscanf(scanner.Text(), "%d", &n)
25
26    for i := 0; i < n; i++ {
27        scanner.Scan()
28        s := scanner.Text()
29        res := make([]byte, len(s))
30
31        // 处理字符串
32        for j := 0; j < len(s); j++ {
33            diff := (int(s[j]-'a') + ans[j]) % 26
34            res[j] = byte('a' + diff)
35        }
36
37        fmt.Println(string(res))
38    }
39 }
```

# 华为OD机考 2025 C卷 - 自然数分解 / 用连续自然数之和来表达整数 (C++ & Pyth)

## 自然数分解

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

华为OD机试2025C卷 100分题型

## 题目描述

一个整数可以由连续的自然数之和来表示。

给定一个整数，计算该整数有几种连续自然数之和的表达式，且打印出每种表达式

## 输入描述

一个目标整数T ( $1 \leq T \leq 1000$ )

## 输出描述

该整数的所有表达式和表达式的个数。

如果有多种表达式，输出要求为：自然数个数最少的表达式优先输出，每个表达式中按自然数递增的顺序输出，具体的格式参见样例。

在每个测试数据结束时，输出一行”Result:X”，其中X是最终的表达式个数。

## 用例1

### 输入

### 输出

```
Plain Text |  
1  9=9  
2  9=4+5  
3  9=2+3+4  
4  Result:3
```

## 说明

Plain Text

- 1 整数 9 有三种表示方法，第1个表达式只有1个自然数，最先输出，
- 2 第2个表达式有2个自然数，第2次序输出，
- 3 第3个表达式有3个自然数，最后输出。
- 4 每个表达式中的自然数都是按递增次序输出的。
- 5 数字与符号之间无空格

## 用例2

### 输入

### 输出

Plain Text

- 1  $10=10$
- 2  $10=1+2+3+4$
- 3 Result:2

## 题解一

思路：使用 等差数列 相关的数学原理解答

1. 连续自然数之和等于t，可以从枚举 自然数项数 来解决。具体处理逻辑如下：
  - a. 假设当前枚举项数为 n 项，可以假设首项为 x，所有项和首项的差值构成一个等差数列，使用 等差数列求和 公式计算出  $diffSum = n * (n - 1) / 2$  的值，然后判断是否可拆分为n项
    - $(t - diffSum) \% n == 0$  :说明可拆分为n项连续和，首项就为  $(t - diffSum) / n$  差值为1，构造出对应序列即可。
    - $(t - diffSum) \% n != 0$  :不能拆分，跳过。
  - b. 枚举结束条件有两个  $(n * (n - 1) / 2) >= t$  或者  $begin < 1$  结束，都是自然数的限制。
2. 根据1的逻辑，处理之后，按题目要求输出结果即可。

额外注释，枚举项数要从2开始，因为当n=1时，  $n * (n - 1) / 2$  会出现除0的情况。

## C++

```
1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 int main() {
6     int t;
7     cin >> t;
8     vector<vector<int>> res;
9     res.push_back({t}); // 单个数本身就是一种表达方式
10
11    // 枚举向数
12    for (int num = 2; num * (num - 1) / 2 < t; ++num) {
13        // 等差数列求和公式 假设连续首项为x 这里计算的所有数与首项差值的等差数列
14        int diffSum = num * (num - 1) / 2;
15        int remain = t - diffSum;
16        // 不能整除说明无法拆解为num项
17        if (remain % num != 0) continue;
18        int begin = remain / num;
19        // 大于0的整数才是自然数
20        if (begin < 1){
21            break;
22        }
23        vector<int> tmp;
24        for (int i = 0; i < num; ++i) {
25            tmp.push_back(begin + i);
26        }
27        res.push_back(tmp);
28    }
29
30    // 输出结果
31    for (const auto& seq : res) {
32        cout << t << "=";
33        for (int i = 0; i < seq.size(); ++i) {
34            cout << seq[i];
35            if (i < seq.size() - 1) cout << "+";
36        }
37        cout << "\n";
38    }
39    cout << "Result:" << res.size() << endl;
40    return 0;
41 }
```

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int t = sc.nextInt();
7         List<List<Integer>> res = new ArrayList<>();
8         res.add(Collections.singletonList(t)); // 单个数本身就是一种表达方式
9
10        // 枚举项数
11        for (int num = 2; num * (num - 1) / 2 < t; ++num) {
12            // 等差数列求和公式: 与首项差值前num项之和为 num*(num-1)/2
13            int diffSum = num * (num - 1) / 2;
14            int remain = t - diffSum;
15
16            // 不能整除说明无法拆解为num项
17            if (remain % num != 0) continue;
18
19            int begin = remain / num;
20            // 大于0的整数才是自然数
21            if (begin < 1) break;
22
23            List<Integer> tmp = new ArrayList<>();
24            for (int i = 0; i < num; ++i) {
25                tmp.add(begin + i);
26            }
27            res.add(tmp);
28        }
29
30        // 输出结果
31        for (List<Integer> seq : res) {
32            System.out.print(t + "=");
33            for (int i = 0; i < seq.size(); ++i) {
34                System.out.print(seq.get(i));
35                if (i < seq.size() - 1) System.out.print("+");
36            }
37            System.out.println();
38        }
39        System.out.println("Result:" + res.size());
40    }
41 }
```

## Python

```
1 t = int(input())
2 res = []
3 res.append([t])
4
5
6 num = 2
7 while num * (num - 1) // 2 < t:
8
9     diff_sum = num * (num - 1) // 2
10    remain = t - diff_sum
11
12    if remain % num != 0:
13        num += 1
14        continue
15
16    begin = remain // num
17    if begin < 1:
18        break
19
20    tmp = [begin + i for i in range(num)]
21    res.append(tmp)
22    num += 1
23
24
25
26 for seq in res:
27     print(f"{t}={"+''.join(map(str, seq))})
28 print(f"Result:{len(res)}")
```

## JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = [];
9 rl.on('line', (line) => {
10     input.push(line);
11 }).on('close', () => {
12     let t = parseInt(input[0]);
13     let res = [];
14     res.push([t]);
15
16
17     for (let num = 2; num * (num - 1) / 2 < t; ++num) {
18
19         let diffSum = num * (num - 1) / 2;
20         let remain = t - diffSum;
21
22         if (remain % num !== 0) continue;
23
24         let begin = remain / num;
25         if (begin < 1) break;
26
27         let tmp = [];
28         for (let i = 0; i < num; ++i) {
29             tmp.push(begin + i);
30         }
31         res.push(tmp);
32     }
33
34
35     for (let seq of res) {
36         console.log(`\$${t}\$\$${seq.join('+')}`);
37     }
38     console.log(`Result:\$${res.length}`);
39 });
```

## Go

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 func main() {
8     var t int
9     fmt.Scan(&t)
10    res := [][]int{{t}}
11
12
13    for num := 2; num*(num-1)/2 < t; num++ {
14
15        diffSum := num * (num - 1) / 2
16        remain := t - diffSum
17
18        if remain%num != 0 {
19            continue
20        }
21
22        begin := remain / num
23        if begin < 1 {
24            break
25        }
26
27        tmp := make([]int, num)
28        for i := 0; i < num; i++ {
29            tmp[i] = begin + i
30        }
31        res = append(res, tmp)
32    }
33
34
35    for _, seq := range res {
36        fmt.Printf("%d=", t)
37        for i, val := range seq {
38            if i != 0 {
39                fmt.Print("+")
40            }
41            fmt.Print(val)
42        }
43        fmt.Println()
44    }
45    fmt.Printf("Result:%d\n", len(res))
```

## 题解二

思路： 双指针

1. 双指针本身就特别符合连续特性，定义双边界 `left = 1, right = 2` , 定义规则 `[left, right]` 为选择区域，使用 `sum` 记录选中区域的和，指针移动规则为
  - a. `sum > t` : 左指针右移, `sum -= left, left++`
  - b. `sum < t` : 右指针右移, `sum += right, right++`
  - c. `sum == t` : 找到一个合法序列 `[left, right)` . 记录这个序列到结果数组中，并移动任一指针即可。
2. 根据1的逻辑，结束条件设置为 `right > t+1` 。然后对结果数组进行排序之后，按照格式输出即可。

C++

```
1 #include <iostream>
2 #include <vector>
3 #include<algorithm>
4 using namespace std;
5
6 int main() {
7     int t;
8     cin >> t;
9     vector<vector<int>> res;
10    vector<int> sequence(t + 1);
11    for (int i = 0; i <= t; i++) {
12        sequence[i] = i + 1;
13    }
14    // 使用双指针 [left, right)属于区域中的值
15    int left = 0, right = 1;
16    int sum = sequence[left];
17    while (right <= t) {
18        if (sum < t) {
19            sum += sequence[right];
20            right++;
21        } else if (sum > t) {
22            sum -= sequence[left];
23            left++;
24        } else {
25            vector<int> tmp(sequence.begin() + left, sequence.begin() + right);
26            res.push_back(tmp);
27            //移动左指针
28            sum -= sequence[left];
29            left++;
30        }
31    }
32    // 根据长度进行排序
33    sort(res.begin(), res.end(), [] (const vector<int> &a, const vector<int> &b) {
34        return a.size() < b.size();
35    });
36
37    // 输出结果
38    for (const auto& seq : res) {
39        cout << t << "=";
40        for (int i = 0; i < seq.size(); ++i) {
41            cout << seq[i];
42            if (i < seq.size() - 1) cout << "+";
43        }
44    }
45}
```

```
44         cout << "\n";
45     }
46     cout << "Result:" << res.size() << endl;
47
48 }
```

## JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner sc = new Scanner(System.in);
6         int t = sc.nextInt();
7         List<List<Integer>> res = new ArrayList<>();
8         int[] sequence = new int[t + 1];
9         for (int i = 0; i <= t; i++) {
10             sequence[i] = i + 1;
11         }
12
13         // 使用双指针 [left, right) 属于区域中的值
14         int left = 0, right = 1;
15         int sum = sequence[left];
16         while (right <= t) {
17             if (sum < t) {
18                 sum += sequence[right];
19                 right++;
20             } else if (sum > t) {
21                 sum -= sequence[left];
22                 left++;
23             } else {
24                 List<Integer> tmp = new ArrayList<>();
25                 for (int i = left; i < right; i++) {
26                     tmp.add(sequence[i]);
27                 }
28                 res.add(tmp);
29                 // 移动左指针
30                 sum -= sequence[left];
31                 left++;
32             }
33         }
34
35         // 根据长度排序
36         res.sort(Comparator.comparingInt(List::size));
37
38         // 输出结果
39         for (List<Integer> seq : res) {
40             System.out.print(t + "=");
41             for (int i = 0; i < seq.size(); i++) {
42                 System.out.print(seq.get(i));
43                 if (i < seq.size() - 1) System.out.print("+");
44             }
45             System.out.println();
46         }
47     }
48 }
```

```
46         }
47         System.out.println("Result:" + res.size());
48     }
49 }
```

## Python

```
1  t = int(input())
2  res = []
3  sequence = [i + 1 for i in range(t + 1)]
4
5  left, right = 0, 1
6  s = sequence[left]
7  while right <= t:
8      if s < t:
9          s += sequence[right]
10     right += 1
11     elif s > t:
12         s -= sequence[left]
13         left += 1
14     else:
15         tmp = sequence[left:right]
16         res.append(tmp)
17
18         s -= sequence[left]
19         left += 1
20
21
22 res.sort(key=len)
23
24
25 for seq in res:
26     print(f"\{t}\=" + "+".join(map(str, seq)))
27 print("Result:" + str(len(res)))
```

Plain Text |

## JavaScript

```
1 const readline = require('readline');
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let inputLines = [];
9
10 rl.on('line', function (line) {
11     inputLines.push(line.trim());
12     if (inputLines.length === 1) {
13         main(parseInt(inputLines[0]));
14         rl.close();
15     }
16 });
17
18 function main(t) {
19     let res = [];
20     let sequence = Array.from({ length: t + 1 }, (_, i) => i + 1);
21
22     let left = 0, right = 1;
23     let sum = sequence[left];
24
25     while (right <= t) {
26         if (sum < t) {
27             sum += sequence[right++];
28         } else if (sum > t) {
29             sum -= sequence[left++];
30         } else {
31             let tmp = sequence.slice(left, right);
32             res.push(tmp);
33             sum -= sequence[left++];
34         }
35     }
36 }
37
38
39 res.sort((a, b) => a.length - b.length);
40
41 for (let seq of res) {
42     console.log(`\$t\` = ` + seq.join('+'));
43 }
44
45 console.log("Result:" + res.length);
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "sort"
6 )
7
8 func main() {
9     var t int
10    fmt.Scan(&t)
11
12    res := [][]int{}
13    sequence := make([]int, t+1)
14    for i := 0; i <= t; i++ {
15        sequence[i] = i + 1
16    }
17
18    left, right := 0, 1
19    sum := sequence[left]
20    for right <= t {
21        if sum < t {
22            sum += sequence[right]
23            right++
24        } else if sum > t {
25            sum -= sequence[left]
26            left++
27        } else {
28            tmp := make([]int, right-left)
29            copy(tmp, sequence[left:right])
30            res = append(res, tmp)
31            sum -= sequence[left]
32            left++
33        }
34    }
35 }
36
37
38 sort.Slice(res, func(i, j int) bool {
39     return len(res[i]) < len(res[j])
40 })
41
42
43 for _, seq := range res {
44     fmt.Printf("%d=", t)
45     for i := 0; i < len(seq); i++ {
```

```
46         fmt.Println(seq[i])
47         if i < len(seq)-1 {
48             fmt.Print("+")
49         }
50     }
51     fmt.Println()
52 }
53 fmt.Printf("Result:%d\n", len(res))
54 }
```

来自: 华为OD机考 2025 C卷 – 自然数分解 / 用连续自然数之和来表达整数 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机试 2025c卷 - 寻找最优的路测线路

## (C++ & Python & JAVA & J

### 寻找最优的路测线路

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

华为OD机试2025C卷 200分题型

#### 题目描述

评估一个网络的信号质量，其中一个做法是将网络划分为栅格，然后对每个栅格的信号质量计算。

路测的时候，希望选择一条信号最好的路线（彼此相连的栅格集合）进行演示。

现给出 R 行 C 列的整数数组 Cov，每个单元格的数值 S 即为该栅格的信号质量（已归一化，无单位，值越大信号越好）。

要求从 [0, 0] 到 [R-1, C-1] 设计一条最优路测路线。返回该路线得分。

规则：

1. 路测路线可以上下左右四个方向，不能对角
2. 路线的评分是以路线上信号最差的栅格为准的，例如路径 8→4→5→9 的值为4，该线路评分为4。线路最优表示该条线路的评分最高。

#### 输入描述

第一行表示栅格的行数 R

第二行表示栅格的列数 C

第三行开始，每一行表示栅格地图一行的信号值，如5 4 5

#### 输出描述

最优路线的得分

#### 备注

- $1 \leq R, C \leq 20$
- $0 \leq S \leq 65535$

#### 用例1

##### 输入

##### 输出

## 说明

## 用例2

### 输入

```
1 6
2 5
3 3 4 6 3 4
4 0 2 1 1 7
5 8 8 3 2 7
6 3 2 4 9 8
7 4 1 2 0 0
8 4 6 5 4 3
```

Plain Text |

### 输出

## 说明

```
1 路线为: 3→4→6→3→4→7→7→8→9→4→3→8→8→3→4→4→6→5→4→3
```

Plain Text |

## 题解

思路：单点最短路径的变形题，使用 Dijkstra 算法求解。

- 定义 `dp` 数组，初始化所有元素为0。`dp[x][y]` 的值代表  $\{0, 0\} \rightarrow \{x, y\}$  的所有路径

中"最大的"最小节点权重。设置 `dp[0][0] = cov[0][0]`

- 之后定义优先队列 `pq` (大顶堆), `dist[x][y]` 越大在队列的权重中优先级越高。初始时将(0,0)加入优先队列。
- 之后不断从优先队列中获取优先级最高的节点, 更新与上下左右节点的dp数组的值。
  - 假设优先队列中取出的节点坐标为{x, y}, 值为`dp[x][y]`, 邻接节点的坐标为{nx, ny}, `dp[nx][ny] = min(cov[nx][ny], dp[x][y])`, 如果 `dp[nx][ny]` 值变大, 将该节点加入到优先队列中。
  - 按上述逻辑一直进行, 直到pq为空。
- 输出 `dp[row-1][col-1]` 的值

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<queue>
8 using namespace std;
9
10 int main() {
11     int row,col;
12     cin >> row >> col;
13     vector<vector<int>> cov(row, vector<int>(col,0));
14
15     for (int i = 0; i < row; i++) {
16         for (int j = 0; j < col; j++) {
17             cin >> cov[i][j];
18         }
19     }
20
21
22     vector<int> dp(row * col, 0);
23     dp[0] = cov[0][0];
24
25     // 上下左右四个方向
26     int direct[4][2] = {
27     {1,0}, {-1,0}, {0,1}, {0,-1}};
28
29     // Lambda 比较函数: 按照数组值的大小决定优先级 (值越大优先级越高)
30     auto cmp = [&](int a, int b) {
31         return dp[a] < dp[b]; // 大顶堆, 值越大, 优先级越高
32     };
33
34     priority_queue<int, vector<int>, decltype(cmp)> pq(cmp);
35
36     pq.push(0);
37     while (pq.size() > 0) {
38         int index = pq.top();
39         pq.pop();
40         int x = index / col;
41         int y = index % col;
42         // 说明[row-1][col-1]已经是最大值, 其它的不能使它变得更大
43         if (x == row - 1 && y == col - 1) {
44             break;
45         }
```

```
46     // 遍历四个方向
47     for (int i = 0; i < 4; i++) {
48         int nextX = x + direct[i][0];
49         int nextY = y + direct[i][1];
50
51         if (nextX < 0 || nextX >= row || nextY < 0 || nextY >= col) {
52             continue;
53         }
54         // 二维转移一维
55         int pos = nextX * col + nextY;
56
57         int w = min(dp[index], cov[nextX][nextY]);
58
59         // 更新pos的值, 重新加入到队列去更新它邻接的网格信号质量
60         if (dp[pos] < w) {
61             dp[pos] = w;
62
63             pq.push(pos);
64         }
65     }
66 }
67
68 cout << dp[row * col - 1];
69 return 0;
70 }
```

## JAVA

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         int row = scanner.nextInt();
8         int col = scanner.nextInt();
9         int[][] cov = new int[row][col];
10
11        for (int i = 0; i < row; i++) {
12            for (int j = 0; j < col; j++) {
13                cov[i][j] = scanner.nextInt();
14            }
15        }
16
17        int[] dp = new int[row * col];
18        Arrays.fill(dp, 0);
19        dp[0] = cov[0][0];
20
21        // 上下左右四个方向
22        int[][] direct = {
23            {1, 0}, {-1, 0}, {0, 1}, {0, -1}};
24
25        // Lambda 比较函数: 按照 dp 值的大小决定优先级 (值越大优先级越高)
26        PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> Integer.
compare(dp[b], dp[a]));
27
28        pq.offer(0);
29
30        while (!pq.isEmpty()) {
31            int index = pq.poll();
32            int x = index / col;
33            int y = index % col;
34
35            // 说明 (row-1, col-1) 已经是最大值, 其他的不能使它变得更大
36            if (x == row - 1 && y == col - 1) {
37                break;
38            }
39
40            // 遍历四个方向
41            for (int[] d : direct) {
42                int nextX = x + d[0];
43                int nextY = y + d[1];
```

```
45             if (nextX < 0 || nextX >= row || nextY < 0 || nextY >= co
46             l) {
47                 continue;
48             }
49
50             // 二维索引转换为一维
51             int pos = nextX * col + nextY;
52             int w = Math.min(dp[index], cov[nextX][nextY]);
53
54             // 更新 dp[pos] 的值，并重新加入队列
55             if (dp[pos] < w) {
56                 dp[pos] = w;
57                 pq.offer(pos);
58             }
59         }
60
61         System.out.println(dp[row * col - 1]);
62         scanner.close();
63     }
64 }
```

## Python

```
1 import sys
2 import heapq
3
4
5 row = int(sys.stdin.readline().strip())
6 col = int(sys.stdin.readline().strip())
7
8
9 cov = [list(map(int, sys.stdin.readline().strip().split())) for _ in range
10 (row)]
11
12 dp = [0] * (row * col)
13 dp[0] = cov[0][0]
14
15
16 directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]
17
18
19 pq = [(-dp[0], 0)]
20 heapq.heapify(pq)
21
22 while pq:
23
24     neg_val, index = heapq.heappop(pq)
25     x, y = divmod(index, col)
26
27
28     if x == row - 1 and y == col - 1:
29         break
30
31
32     for dx, dy in directions:
33         nextX, nextY = x + dx, y + dy
34
35
36     if 0 <= nextX < row and 0 <= nextY < col:
37         pos = nextX * col + nextY
38         w = min(-neg_val, cov[nextX][nextY])
39
40
41         if dp[pos] < w:
42             dp[pos] = w
43             heapq.heappush(pq, (-dp[pos], pos))
44
```

```
45  
46    print(dp[row * col - 1])
```

## JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout
6 });
7
8 let input = [];
9 rl.on("line", (line) => {
10     input.push(line.trim());
11 }).on("close", () => {
12     if (input.length < 2) {
13
14         process.exit(1);
15     }
16
17
18     let row = parseInt(input[0], 10);
19     let col = parseInt(input[1], 10);
20
21
22     if (isNaN(row) || isNaN(col) || row <= 0 || col <= 0) {
23
24         process.exit(1);
25     }
26
27
28     if (input.length - 2 < row) {
29
30         process.exit(1);
31     }
32
33
34     let cov = [];
35     for (let i = 2; i < 2 + row; i++) {
36         let rowData = input[i].split(/\s+/).map(Number);
37         if (rowData.length !== col) {
38
39             process.exit(1);
40         }
41         cov.push(rowData);
42     }
43
44
45     let dp = Array(row * col).fill(0);
```

```

46     dp[0] = cov[0][0];
47
48
49     let directions = [[1, 0], [-1, 0], [0, 1], [0, -1]];
50
51
52     class PriorityQueue {
53         constructor(compare) {
54             this.data = [];
55             this.compare = compare;
56         }
57         push(item) {
58             this.data.push(item);
59             this.data.sort(this.compare);
60         }
61         pop() {
62             return this.data.shift();
63         }
64         size() {
65             return this.data.length;
66         }
67     }
68
69
70     let pq = new PriorityQueue((a, b) => dp[b] - dp[a]);
71
72
73     pq.push(0);
74
75     while (pq.size() > 0) {
76         let index = pq.pop();
77         let x = Math.floor(index / col);
78         let y = index % col;
79
80
81         if (x === row - 1 && y === col - 1) break;
82
83
84         for (let [dx, dy] of directions) {
85             let nextX = x + dx;
86             let nextY = y + dy;
87
88
89             if (nextX < 0 || nextX >= row || nextY < 0 || nextY >= col) c
90             ontinue;
91
92             let pos = nextX * col + nextY;
93             let w = Math.min(dp[index], cov[nextX][nextY]);

```

```
93
94
95      if (dp[pos] < w) {
96          dp[pos] = w;
97          pq.push(pos);
98      }
99  }
100 }
101
102
103     console.log(dp[row * col - 1]);
104     process.exit(0);
105 });

});
```

## Go

```
1 package main
2
3 import (
4     "container/heap"
5     "fmt"
6 )
7
8
9 type Point struct {
10     signal int
11     index   int
12 }
13
14
15 type MaxHeap []Point
16
17 func (h MaxHeap) Len() int           { return len(h) }
18 func (h MaxHeap) Less(i, j int) bool { return h[i].signal > h[j].signal }
19 func (h MaxHeap) Swap(i, j int)      { h[i], h[j] = h[j], h[i] }
20
21 func (h *MaxHeap) Push(x interface{}) {
22     *h = append(*h, x.(Point))
23 }
24
25 func (h *MaxHeap) Pop() interface{} {
26     old := *h
27     n := len(old)
28     x := old[n-1]
29     *h = old[:n-1]
30     return x
31 }
32
33 func main() {
34     var row, col int
35     fmt.Scan(&row, &col)
36
37
38     cov := make([][]int, row)
39     for i := 0; i < row; i++ {
40         cov[i] = make([]int, col)
41         for j := 0; j < col; j++ {
42             fmt.Scan(&cov[i][j])
43         }
44     }
45 }
```

```

46
47     dp := make([]int, row*col)
48     dp[0] = cov[0][0]
49
50
51     directions := [4][2]int{{1, 0}, {-1, 0}, {0, 1}, {0, -1}}
52
53
54     pq := &MaxHeap{Point{cov[0][0], 0}}
55     heap.Init(pq)
56
57     for pq.Len() > 0 {
58         cur := heap.Pop(pq).(Point)
59         x, y := cur.index/col, cur.index%col
60
61
62         if x == row-1 && y == col-1 {
63             break
64         }
65
66
67         for _, d := range directions {
68             nx, ny := x+d[0], y+d[1]
69
70
71             if nx < 0 || nx >= row || ny < 0 || ny >= col {
72                 continue
73             }
74
75             pos := nx*col + ny
76             w := min(cur.signal, cov[nx][ny])
77
78
79             if dp[pos] < w {
80                 dp[pos] = w
81                 heap.Push(pq, Point{w, pos})
82             }
83         }
84     }
85
86
87     fmt.Println(dp[row*col-1])
88 }
89
90
91 func min(a, b int) int {
92     if a < b {
93         return a

```

```
94     }
95     return b
96 }
```

| 来自: 华为OD机试 2025c卷 – 寻找最优的路测线路 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机考2025C卷 - 数字序列比大小(C++ & Python & JAVA & JS)

## 数字序列比大小

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

华为OD机试2025C卷 200分题型

### 题目描述

A, B两个人玩一个数字比大小的游戏，在游戏前，两个人会拿到相同长度的两个数字序列，两个数字序列不相同的，且其中的数字是随机的。

A, B各自从数字序列中挑选出一个数字进行大小比较，赢的人得1分，输的人扣1分，相等则各自的分数不变。用过的数字需要丢弃。

求A可能赢B的最大分数。

### 输入描述

输入数据的第1个数字表示数字序列的长度N，后面紧跟着两个长度为N的数字序列。

### 输出描述

A可能赢B的最大分数

### 备注

1. 这里要求计算A可能赢B的最大分数，不妨假设，A知道B的数字序列，且总是B先挑选数字并明示。
2. 可以采用贪心策略，能赢的一定要赢，要输的尽量减少损失。

### 用例1

#### 输入

#### 输出

#### 说明

```

1 输入数据第1个数字表示数字序列长度为3，后面紧跟着两个长度为3的数字序列。
2 序列A: 4 8 10
3 序列B: 3 6 4
4 A可以赢的最大分数是3。获得该分数的比大小过程可以是：
5 1) A: 4    B: 3
6 2) A: 8    B: 6
7 3) A: 10   B: 4

```

## 题解

思路: 田忌赛马问题 : 首先将分数进行从小到达排序, 分别使用双指针指向两人最大值和最小值下标, 从两个最大值处开始比较, 指针移动规律分为以下三种情况:

- $a[a_{right}] > b[b_{right}]$ : a可以直接获胜, `res+= 1, a_{right} -= 1, b_{right}-=1`
- $a[a_{right}] < b[b_{right}]$ : a肯定会输一场, 选择将最小值损失掉, `a_{left} += 1, b_{right} -= 1, res -= 1`
- $a[a_{right}] == b[b_{right}]$ : 这时候需要考虑最小值的情况: 举两个例子a [3 4] b [2 4] / a [2 5] b[3 5]
  - $a[a_{left}] > b[b_{left}]$  情况下, 此时应该移动先考虑左指针, `a_{left} += 1, b_{left} += 1, res +=1`
  - $a[a_{left}] <= b[b_{left}]$  情况下, a应该使用最小值去对抗b的最大值。

## C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 using namespace std;
8
9 int main() {
10     int n;
11     cin >> n;
12     vector<int> aScores(n), bScores(n);
13     for (int i = 0; i < n; i++) {
14         cin >> aScores[i];
15     }
16     for (int i = 0; i < n; i++) {
17         cin >> bScores[i];
18     }
19     sort(aScores.begin(), aScores.end());
20     sort(bScores.begin(), bScores.end());
21
22     int leftA = 0, rightA = n - 1;
23     int leftB = 0, rightB = n - 1;
24     int res = 0;
25
26     while (leftB <= rightB) {
27         // 最大值相比, a > b 则直接比
28         if (aScores[rightA] > bScores[rightB]) {
29             rightA--;
30             rightB--;
31             res += 1;
32         // 最大值相比, a < b, 无论如何都要输, 选择损失最小数
33         } else if (aScores[rightA] < bScores[rightB]) {
34             rightB--;
35             leftA++;
36             res -=1;
37         } else {
38             // 相等情况下 , a最小值大于b最小值, 先考虑左指针移动
39             if (aScores[leftA] > bScores[leftB]) {
40                 res+=1;
41                 leftA++;
42                 leftB++;
43             // 相等情况下, a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
44             }else {
45                 // a [1 1 1] b[1 1 1] 考虑这个情况
46             }
47         }
48     }
49 }
```

```
46             if (bScores[rightB] > aScores[leftA]) {
47                 res -=1;
48             }
49             leftA++;
50             rightB--;
51         }
52     }
53 }
54 cout << res;
55 return 0;
56 }
```

## JAVA

```
1 import java.util.Arrays;
2 import java.util.Scanner;
3
4 public class Main {
5     public static void main(String[] args) {
6         Scanner sc = new Scanner(System.in);
7
8         int n = Integer.parseInt(sc.nextLine());
9         int[] a = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::pa
rseInt).toArray();
10        int[] b = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::pa
rseInt).toArray();
11
12        int result = handle(n, a, b);
13        System.out.println(result);
14    }
15
16    public static int handle(int n, int[] a, int[] b) {
17        Arrays.sort(a);
18        Arrays.sort(b);
19
20        int la = 0; // 指向a的最小值index
21        int ra = n - 1; // 指向a的最大值index
22
23        int lb = 0; // 指向b的最小值index
24        int rb = n - 1; // 指向b的最大值index
25
26        int ans = 0; // 记录a可以获得的分数
27
28        while (la <= ra) {
29            if (a[ra] > b[rb]) {
30                // a最大值 > b的最大值 直接比
31                ans += 1;
32                ra--;
33                rb--;
34            } else if (a[ra] < b[rb]) {
35                // a最大值 < b的最大值, 损失掉a的最小值, 保留最大值
36                ans -= 1;
37                la++;
38                rb--;
39            } else {
40                // 相等情况下 , a最小值大于b最小值, 先考虑左指针移动
41                if (a[la] > b[lb]) {
42                    ans += 1;
43                    la++;
```

```
44     lb++;
45     // 相等情况下，a的最小值小于b的最小值。 使用a的最小值和b的最大值对抗
46 } else {
47     // a [1 1 1] b[1 1 1] 考虑这个情况
48     if (b[rb] > a[la]) ans -= 1;
49     la++;
50     rb--;
51 }
52 }
53 }
54 return ans;
55 }
56 }
57 }
```

## Python

```
1  n = int(input())
2  a = list(map(int, input().split()))
3  b = list(map(int, input().split()))
4
5
6
7  def handle():
8      a.sort()
9      b.sort()
10
11     la = 0
12     ra = n - 1
13
14     lb = 0
15     rb = n - 1
16
17     ans = 0
18
19     while la <= ra:
20         if a[ra] > b[rb]:
21
22             ans += 1
23             ra -= 1
24             rb -= 1
25         elif a[ra] < b[rb]:
26
27             ans -= 1
28             la += 1
29             rb -= 1
30         else:
31
32             if a[la] > b[lb]:
33
34                 ans += 1
35                 la += 1
36                 lb += 1
37
38         else:
39
40             if b[rb] > a[la]:
41                 ans -= 1
42                 la += 1
43                 rb -= 1
44
45     return ans
```

```
46  
47  
48  
49  print(handle())
```

## JavaScript

```
1 const readline = require("readline");
2
3 const rl = readline.createInterface({
4     input: process.stdin,
5     output: process.stdout,
6 });
7
8 const lines = [];
9 rl.on("line", (line) => {
10     lines.push(line);
11
12     if (lines.length == 3) {
13         const n = parseInt(lines[0]);
14         const a = lines[1].split(" ").map(Number);
15         const b = lines[2].split(" ").map(Number);
16
17         console.log(handle(n, a, b));
18
19         lines.length = 0;
20     }
21 });
22 );
23
24 function handle(n, a, b) {
25     a.sort((a, b) => a - b);
26     b.sort((a, b) => a - b);
27
28     let la = 0;
29     let ra = n - 1;
30
31     let lb = 0;
32     let rb = n - 1;
33
34     let ans = 0;
35
36     while (la <= ra) {
37         if (a[ra] > b[rb]) {
38
39             ans += 1;
40             ra--;
41             rb--;
42         } else if (a[ra] < b[rb]) {
43
44             ans -= 1;
45             la++;
46         }
47     }
48
49     return ans;
50 }
```

```
46         rb--;
47     } else {
48
49         if (a[la] > b[lb]) {
50             ans += 1;
51             la++;
52             lb++;
53
54         } else {
55
56             if (b[rb] > a[la]) ans -= 1;
57             la++;
58             rb--;
59         }
60     }
61 }
62
63     return ans;
64 }
```

## Go

```
1 package main
2
3 import (
4     "fmt"
5     "sort"
6 )
7
8 func main() {
9     var n int
10    fmt.Scan(&n)
11
12    aScores := make([]int, n)
13    bScores := make([]int, n)
14
15    for i := 0; i < n; i++ {
16        fmt.Scan(&aScores[i])
17    }
18    for i := 0; i < n; i++ {
19        fmt.Scan(&bScores[i])
20    }
21
22    sort.Ints(aScores)
23    sort.Ints(bScores)
24
25    leftA, rightA := 0, n-1
26    leftB, rightB := 0, n-1
27    res := 0
28
29    for leftB <= rightB {
30
31        if aScores[rightA] > bScores[rightB] {
32            rightA--
33            rightB--
34            res++
35
36        } else if aScores[rightA] < bScores[rightB] {
37            rightB--
38            leftA++
39            res--
40        } else {
41
42            if aScores[leftA] > bScores[leftB] {
43                res++
44                leftA++
45                leftB++
```

```
46
47     } else {
48
49         if bScores[rightB] > aScores[leftA] {
50             res--
51         }
52         leftA++
53         rightB--
54     }
55 }
56 }
57
58 fmt.Println(res)
59 }
```

| 来自: 华为OD 机考 2025C卷 – 数字序列比大小 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机考 2025C卷 - 分月饼 (C++ & Python & JAVA & JS & GO)

## 分月饼

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

华为OD机试 2025C卷 200分题型

### 题目描述

中秋节，公司分月饼， $m$  个员工，买了  $n$  个月饼， $m \leq n$ ，每个员工至少分 1 个月饼，但可以分多个，

- 单人分到最多月饼的个数是  $\text{Max1}$ ，单人分到第二多月饼个数是  $\text{Max2}$ ， $\text{Max1} - \text{Max2} \leq 3$ ，
- 单人分到第  $n - 1$  多月饼个数是  $\text{Max}(n-1)$ ，单人分到第  $n$  多月饼个数是  $\text{Max}(n)$ ， $\text{Max}(n-1) - \text{Max}(n) \leq 3$ ，

问有多少种分月饼的方法？

### 输入描述

每一行输入  $m\ n$ ，表示  $m$  个员工， $n$  个月饼， $m \leq n$

### 输出描述

输出有多少种月饼分法

### 用例1

#### 输入

#### 输出

### 说明

分法有2种:  $4 = 1 + 3$   $4 = 2 + 2$  注意: 1+3和3+1算一种分法

### 用例2

#### 输入

## 输出

## 说明

```
5 = 1 + 1 + 3  
5 = 1 + 2 + 2
```

## 用例3

## 输入

## 输出

## 说明

满足要求的有6种分法：

```
12 = 1 + 1 + 10 (Max1 = 10, Max2 = 1, 不满足Max1 - Max2 <= 3要求)  
12 = 1 + 2 + 9 (Max1 = 9, Max2 = 2, 不满足Max1 - Max2 <= 3要求)  
12 = 1 + 3 + 8 (Max1 = 8, Max2 = 3, 不满足Max1 - Max2 <= 3要求)  
12 = 1 + 4 + 7 (Max1 = 7, Max2 = 4, Max3 = 1, 满足要求)  
12 = 1 + 5 + 6 (Max1 = 6, Max2 = 5, Max3 = 1, 不满足要求)  
12 = 2 + 2 + 8 (Max1 = 8, Max2 = 2, 不满足要求)  
12 = 2 + 3 + 7 (Max1 = 7, Max2 = 3, 不满足要求)  
12 = 2 + 4 + 6 (Max1 = 6, Max2 = 4, Max3 = 2, 满足要求)  
12 = 2 + 5 + 5 (Max1 = 5, Max2 = 2, 满足要求)  
12 = 3 + 3 + 6 (Max1 = 6, Max2 = 3, 满足要求)  
12 = 3 + 4 + 5 (Max1 = 5, Max2 = 4, Max3 = 3, 满足要求)  
12 = 4 + 4 + 4 (Max1 = 4, 满足要求)
```

## 题解

思路： `DFS` 求解

1. 使用 `dfs` 算法拆分整数为 `a0 a2 ... am-1`，保持这个整数序列递增(可以通过这个性质进行去重)，并且 `aj - ai <= 3`，其中 `j - i = 1`。
2. 由序列保持递增性质，确定 `a1` 的取值范围为 `[1, m / n]`。定义函数处理拆分，函数传递参数 `num`

`ber` 当前分配员工序号 `last` 上一个员工分配数列 `remain` 剩余数列，当前员工可分配数量为 `[last, last + 3]` 由 `Max(n-1) - Max(n) ≤ 3` 决定，如果 `number == m-1 && remain - last <= 3 && remian - last >= 0` 则可行解 + 1. 可以添加两个剪枝处理：

- 剩余数量不满足后续分配少的情况：假设当前分配序号为  $x$ , 全部分配为 `last` 个月饼，还是大于剩余数量时可提前剪枝，剩余待分配员工数量为  $(m - 1 - x + 1) = (m - x)$  , 可以当前需要最少月饼数为 `(m - x) * last` , 如果 `(m - x) * last > remain` 则说明 `remain` 不能满足后续分配，可提前剪枝。
- 剩余数量不满足后续分配，多的情况：假设当前分配序号为  $x$ , 每后一个员工比前一个员工分配数量多3个，还是小于剩余数量时可提前剪枝。这里利用 等差数列求和 进行剪枝，剩余分配员工数量为 `(m - x)` , 等差数列首项为 `last + 3` , 等差数列尾项为 `last + 3 + (m - x - 1) * 3` , 如果 `(2 * (last + 3) + (m - x - 1) * 3) * (m - x) < remain * 2` 则说明 `remain` 不能满足后续分配，可提前剪枝。

3. 输出拆解的可行方案即可。

## C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 using namespace std;
10
11 int res = 0;
12 int m, n;
13
14 // number当前员工序号 last 上一个员工分配数量 remain 剩余月饼数量
15 void dfs(int number, int last, int remain) {
16     // 剪枝 剩余数量不满足后续分配 少的情况
17     if ((m - number) * last > remain) {
18         return;
19     }
20
21     // 剪枝 剩余数量不满足后续分配 多的情况
22     if ((2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain *
23 2) {
24         return;
25     }
26
27     if (number == m-1) {
28         if (remain - last <= 3 && remain - last >= 0) {
29             res++;
30         }
31         return;
32     }
33     // 下一项和当前项 0 <= 差值 <= 3
34     for (int i = last; i <= last + 3; i++) {
35         dfs(number + 1, i, remain - i);
36     }
37 }
38
39
40 int main() {
41     cin >> m >> n;
42     // 特殊情况
43     if (m == 1) {
44         cout << 1;
```

```
45         return 0;
46     }
47     // 枚举第一个员工能够获得最大月饼数
48     for (int i = 1; i <= n / m; i++) {
49         dfs(1, i, n - i);
50     }
51     cout << res;
52     return 0;
53 }
```

## JAVA

```
1 import java.util.Scanner;
2
3 public class Main {
4     static int res = 0;
5     static int m, n;
6
7     // number 当前员工序号, last 上一个员工分配数量, remain 剩余月饼数量
8     static void dfs(int number, int last, int remain) {
9         // 剪枝: 剩余数量不满足后续分配 (少)
10        if ((m - number) * last > remain) return;
11
12        // 剪枝: 剩余数量不满足后续分配 (多)
13        if ((2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2) return;
14
15        if (number == m - 1) {
16            if (remain - last <= 3 && remain - last >= 0) res++;
17            return;
18        }
19
20        // 下一项和当前项差值在 0 到 3
21        for (int i = last; i <= last + 3; i++) {
22            dfs(number + 1, i, remain - i);
23        }
24    }
25
26    public static void main(String[] args) {
27        Scanner sc = new Scanner(System.in);
28        m = sc.nextInt();
29        n = sc.nextInt();
30        if (m == 1) {
31            System.out.println(1);
32            return;
33        }
34
35        for (int i = 1; i <= n / m; i++) {
36            dfs(1, i, n - i);
37        }
38        System.out.println(res);
39    }
40 }
```

## Python

```
1 res = 0
2 m, n = map(int, input().split())
3
4
5 def dfs(number, last, remain):
6     global res
7
8     if (m - number) * last > remain:
9         return
10
11
12     if (2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2:
13         return
14
15     if number == m - 1:
16         if remain - last <= 3:
17             res += 1
18         return
19
20
21     for i in range(last, last + 4):
22         dfs(number + 1, i, remain - i)
23
24
25 if m == 1:
26     print(1)
27 else:
28     for i in range(1, n // m + 1):
29         dfs(1, i, n - i)
30     print(res)
```

## JavaScript

```
1 const readline = require('readline');
2 const rl = readline.createInterface({ input: process.stdin, output: process.stdout });
3
4 let m = 0, n = 0, res = 0;
5
6 function dfs(number, last, remain) {
7
8     if ((m - number) * last > remain) return;
9
10    if ((2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2) return;
11
12    if (number === m - 1) {
13        if (remain - last <= 3 && remain - last >= 0) res++;
14        return;
15    }
16
17
18    for (let i = last; i <= last + 3; i++) {
19        dfs(number + 1, i, remain - i);
20    }
21
22 }
23
24 rl.on('line', line => {
25     [m, n] = line.trim().split(' ').map(Number);
26
27     if (m === 1) {
28         console.log(1);
29     } else {
30         for (let i = 1; i <= Math.floor(n / m); i++) {
31             dfs(1, i, n - i);
32         }
33         console.log(res);
34     }
35 });
36
```

## Go

```
1 package main
2
3 import (
4     "fmt"
5 )
6
7 var res int
8 var m, n int
9
10
11 func dfs(number, last, remain int) {
12
13     if (m - number) * last > remain {
14         return
15     }
16
17     if (2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2 {
18         return
19     }
20
21     if number == m - 1 {
22         if remain - last <= 3 && remain - last >= 0{
23             res++
24         }
25         return
26     }
27 }
28
29
30     for i := last; i <= last + 3; i++ {
31         dfs(number + 1, i, remain - i)
32     }
33 }
34
35 func main() {
36     fmt.Scan(&m, &n)
37
38     if m == 1 {
39         fmt.Println(1)
40         return
41     }
42
43     for i := 1; i <= n / m; i++ {
44         dfs(1, i, n - i)
45     }
}
```

```
46     fmt.Println(res)
47 }
```

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

# [来自浏览器插件]无标题

## 二维伞的雨滴效应

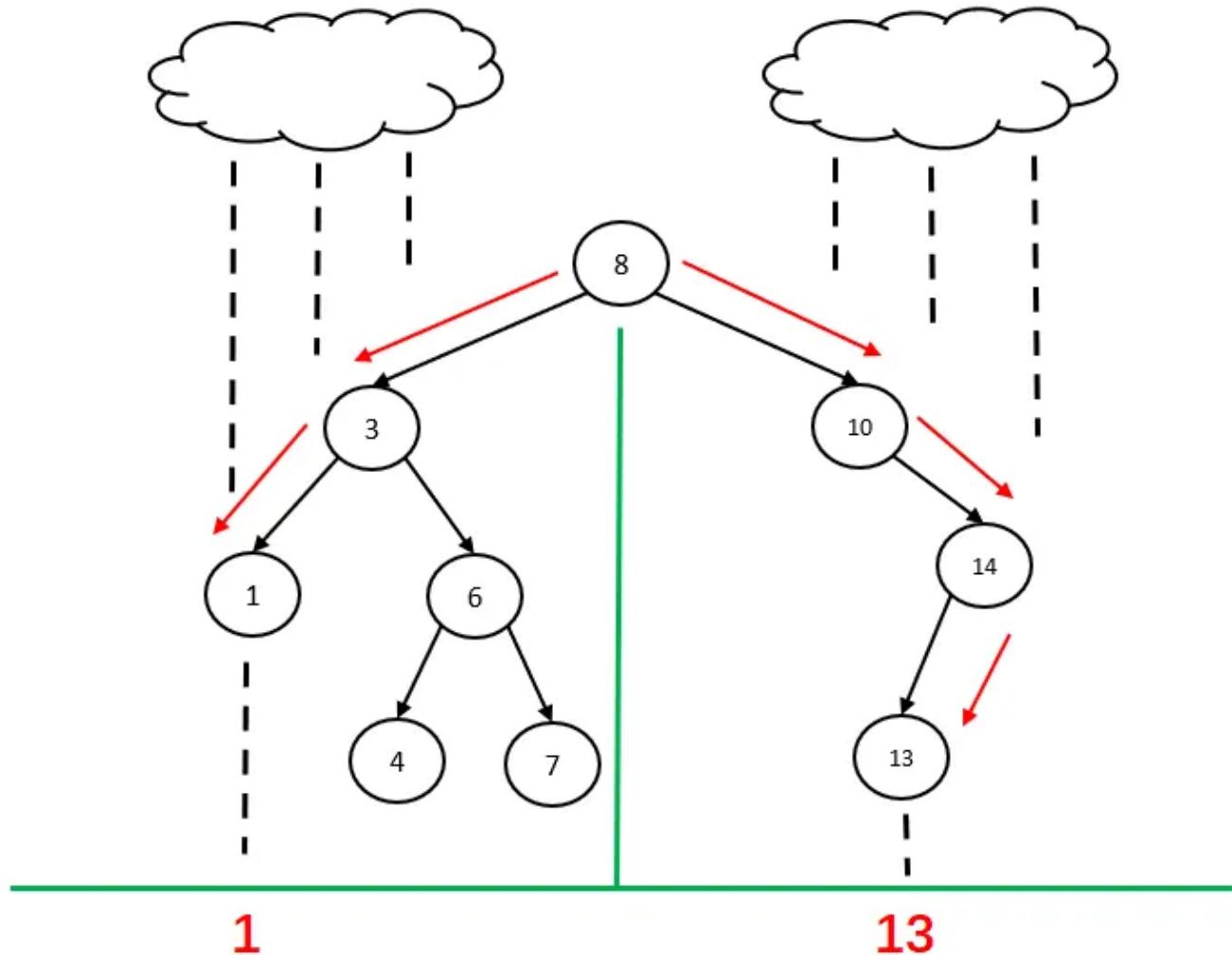
华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录 | 机考题库 + 算法考点详解](#)

华为OD机试2025C卷 200分题型

### 题目描述

普通的伞在二维平面世界中，左右两侧均有一条边，而两侧伞边最下面各有一个伞坠子，雨滴落到伞面，逐步流到伞坠处，会将伞坠的信息携带并落到地面，随着日积月累，地面会呈现伞坠的信息。

- 1、为了模拟伞状雨滴效应，用[二叉树](#)来模拟二维平面伞（如下图所示），现在输入一串正整数数组序列（不含0，数组成员至少是1个），若此数组序列是二叉搜索树的前序遍历的结果，那么请输出一个返回值1，否则输出0。
- 2、同时请将此序列构成的伞状效应携带到地面的数字信息输出来(左边伞坠信息，右边伞坠信息，详细参考示例图地面上数字)，若此树不存在左或右扇坠，则对应位置返回0。同时若非二叉排序树那么左右伞坠信息也返回0。



## 输入描述

一个通过空格分割的整数序列字符串，数组不含0，数组成员至少1个，输入的数组的任意两个数字都互不相同，最多1000个正整数，正整数值范围1~65535

## 输出描述

输出如下三个值，以空格分隔：是否二叉排序树，左侧地面呈现的伞坠数字值，右侧地面呈现的伞坠数字值。

若是二叉排序树，则输出1，否则输出0（其左右伞坠值也直接赋值0）。

若不存在左侧或者右侧伞坠值，那么对应伞坠值直接赋值0。

## 用例1

### 输入

# 输出

## 说明

1表示是[二叉搜索树](#)前序遍历结果，1表示左侧地面呈现的伞坠数字值，13表示右侧地面呈现的伞坠数字值

## 题解

思路：搜索二叉树问题。

- 搜索二叉树的性质：利用这个性质可以判断是否为搜索二叉树
  - a. 左子树的值肯定严格小于父节点。
  - b. 右子树的值肯定严格大于父节点。
- 找左右吊坠的规律：
  - a. 如果根节点没有左子树或者右子树，对应方向吊坠的值为0
  - b. 说一下找左吊坠值的规律，右边类似
    - i. 从根节点向下搜索不断找寻左节点，找到最左边的节点(node)。
    - ii. 如果node存在右节点，把指针调换至 `root->value` 重复1的操作。
    - iii. 重复1和2的操作，直到找到一个叶子节点，这个叶子节点的值就是结果。

## C++

```
1 #include <cstdio>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 #include<list>
9 #include<queue>
10 #include<map>
11 using namespace std;
12
13 struct Node {
14     int value;
15     Node* left;
16     Node* right;
17     Node():left(nullptr), right(nullptr){}
18     Node(int value, Node* left, Node* right):value(value),left(left),right(right) {}
19     Node(int value):value(value),left(nullptr), right(nullptr){}
20 };
21
22
23 // 通用 split 函数
24 vector<string> split(const string& str, const string& delimiter) {
25     vector<string> result;
26     size_t start = 0;
27     size_t end = str.find(delimiter);
28     while (end != string::npos) {
29         result.push_back(str.substr(start, end - start));
30         start = end + delimiter.length();
31         end = str.find(delimiter, start);
32     }
33     // 添加最后一个部分
34     result.push_back(str.substr(start));
35     return result;
36 }
37
38 bool isValid(Node* root, int start, int end, vector<int>& preOrder) {
39     if (start == end) {
40         return true;
41     }
42
43     int i = start + 1;
44     // 不断迭代找到父节点的右子树节点为止
```

```

45     while (i <= end && preOrder[i] < root->value) {
46         i++;
47     }
48
49     int j = i;
50     // 按照搜索树规则 找到右子树的终点位置
51     while (j <= end && preOrder[j] > root->value) {
52         j++;
53     }
54     // 异常 j 必须等于 end才是正常的
55     if (j <= end) {
56         return false;
57     }
58
59     // i > start + 1说明才存在左子树
60     if (i > start + 1) {
61         root->left = new Node(preOrder[start+1]);
62         //递归构建左节点的结构
63         if (!isValid(root->left, start + 1, i - 1, preOrder)) {
64             return false;
65         }
66     }
67
68     // i <= end说明才存在右子树
69     if (i <= end) {
70         root->right = new Node(preOrder[i]);
71         return isValid(root->right, i, end, preOrder);
72     }
73
74     return true;
75 }
76
77 // 获取左伞坠的值
78 int leftBottom(Node* root, int level) {
79     if (root->left != nullptr) {
80         return leftBottom(root->left, level + 1);
81     }
82     // 说明根节点没有子节点, 没有坠子, 直接返回0
83     if (level == 0) {
84         return 0;
85     }
86
87     // 存在
88     if (root->right != nullptr) {
89         return leftBottom(root->right, level + 1);
90     }
91     return root->value;
92 }

```

```
93
94 // 获取右伞坠的值
95 int rightBottom(Node* root, int level) {
96     if (root -> right != nullptr) {
97         return rightBottom(root->right, level + 1);
98     }
99     // 说明根节点没有子节点，没有坠子，直接返回0
100    if (level == 0) {
101        return 0;
102    }
103
104    // 存在右节点
105    if (root->left != nullptr) {
106        return rightBottom(root->left, level + 1);
107    }
108    return root->value;
109 }
110
111 void dfs(Node* root, vector<int>& res) {
112     if (root == nullptr) {
113         return;
114     }
115     dfs(root->left, res);
116     res.push_back(root->value);
117     dfs(root->right, res);
118 }
119
120
121 int main() {
122     string line;
123     getline(cin, line);
124     vector<string> tmp = split(line, " ");
125     int n = tmp.size();
126     vector<int> ans(n);
127     for (int i = 0; i < n; i++) {
128         ans[i] = stoi(tmp[i]);
129     }
130     Node* root = new Node(ans[0]);
131     // 不合法
132     if (!isValid(root, 0, ans.size() - 1, ans)) {
133         cout << "0 0 0";
134         return 0;
135     }
136     vector<int> res;
137     dfs(root, res);
138
139     int leftValue = leftBottom(root, 0);
140     int rightValue = rightBottom(root, 0);
```

```
141
142     cout << "1 " << leftValue << " " << rightValue;
143     return 0;
144 }
```

## JAVA

```
1 import java.util.*;
2
3 class Node {
4     int value;
5     Node left, right;
6
7     Node(int value) {
8         this.value = value;
9         this.left = null;
10        this.right = null;
11    }
12 }
13
14 public class Main {
15
16     // 判断是否是有效的二叉搜索树的前序遍历
17     static boolean isValid(Node root, int start, int end, List<Integer> preOrder) {
18         if (start == end) {
19             return true;
20         }
21
22         int i = start + 1;
23         // 找到右子树的起点
24         while (i <= end && preOrder.get(i) < root.value) {
25             i++;
26         }
27
28         int j = i;
29         // 验证右子树的所有值是否大于根节点
30         while (j <= end && preOrder.get(j) > root.value) {
31             j++;
32         }
33
34         // 若 j 未能完全遍历完，则说明不符合BST规则
35         if (j <= end) {
36             return false;
37         }
38
39         // 递归构建左子树
40         if (i > start + 1) {
41             root.left = new Node(preOrder.get(start + 1));
42             if (!isValid(root.left, start + 1, i - 1, preOrder)) {
43                 return false;
44             }
45         }
46     }
47 }
```

```
45     }
46
47     // 递归构建右子树
48     if (i <= end) {
49         root.right = new Node(preOrder.get(i));
50         return isValid(root.right, i, end, preOrder);
51     }
52
53     return true;
54 }
55
56 // 获取左伞坠
57 static int leftBottom(Node root, int level) {
58     if (root.left != null) {
59         return leftBottom(root.left, level + 1);
60     }
61     if (level == 0) {
62         return 0;
63     }
64     if (root.right != null) {
65         return leftBottom(root.right, level + 1);
66     }
67     return root.value;
68 }
69
70 // 获取右伞坠
71 static int rightBottom(Node root, int level) {
72     if (root.right != null) {
73         return rightBottom(root.right, level + 1);
74     }
75     if (level == 0) {
76         return 0;
77     }
78     if (root.left != null) {
79         return rightBottom(root.left, level + 1);
80     }
81     return root.value;
82 }
83
84 // 中序遍历
85 static void dfs(Node root, List<Integer> res) {
86     if (root == null) {
87         return;
88     }
89     dfs(root.left, res);
90     res.add(root.value);
91     dfs(root.right, res);
92 }
```

```
93
94     public static void main(String[] args) {
95         Scanner scanner = new Scanner(System.in);
96         String[] inputs = scanner.nextLine().split(" ");
97         List<Integer> ans = new ArrayList<>();
98         for (String num : inputs) {
99             ans.add(Integer.parseInt(num));
100        }
101        scanner.close();
102
103        Node root = new Node(ans.get(0));
104
105        if (!isValid(root, 0, ans.size() - 1, ans)) {
106            System.out.println("0 0 0");
107            return;
108        }
109
110        int leftValue = leftBottom(root, 0);
111        int rightValue = rightBottom(root, 0);
112
113        System.out.println("1 " + leftValue + " " + rightValue);
114    }
115 }
```

## Python

```
1 import sys
2
3
4 class Node:
5     def __init__(self, value):
6         self.value = value
7         self.left = None
8         self.right = None
9
10
11 def is_valid(root, start, end, pre_order):
12     if start == end:
13         return True
14
15     i = start + 1
16
17     while i <= end and pre_order[i] < root.value:
18         i += 1
19
20     j = i
21
22     while j <= end and pre_order[j] > root.value:
23         j += 1
24
25
26     if j <= end:
27         return False
28
29
30     if i > start + 1:
31         root.left = Node(pre_order[start + 1])
32         if not is_valid(root.left, start + 1, i - 1, pre_order):
33             return False
34
35
36     if i <= end:
37         root.right = Node(pre_order[i])
38         return is_valid(root.right, i, end, pre_order)
39
40
41
42
43 def left_bottom(root, level):
44     if root.left:
45         return left_bottom(root.left, level + 1)
```

```

46     if level == 0:
47         return 0
48     if root.right:
49         return left_bottom(root.right, level + 1)
50     return root.value
51
52
53 def right_bottom(root, level):
54     if root.right:
55         return right_bottom(root.right, level + 1)
56     if level == 0:
57         return 0
58     if root.left:
59         return right_bottom(root.left, level + 1)
60     return root.value
61
62
63 pre_order = list(map(int, sys.stdin.readline().strip().split()))
64 root = Node(pre_order[0])
65
66
67 if not is_valid(root, 0, len(pre_order) - 1, pre_order):
68     print("0 0 0")
69 else:
70     print(f"1 {left_bottom(root, 0)} {right_bottom(root, 0)}")

```

## JavaScript

```
1 class Node {
2     constructor(value) {
3         this.value = value;
4         this.left = null;
5         this.right = null;
6     }
7 }
8
9
10 function isValid(root, start, end, preOrder) {
11     if (start === end) return true;
12
13     let i = start + 1;
14
15     while (i <= end && preOrder[i] < root.value) i++;
16
17     let j = i;
18
19     while (j <= end && preOrder[j] > root.value) j++;
20
21
22     if (j <= end) return false;
23
24
25     if (i > start + 1) {
26         root.left = new Node(preOrder[start + 1]);
27         if (!isValid(root.left, start + 1, i - 1, preOrder)) return false;
28     }
29
30
31     if (i <= end) {
32         root.right = new Node(preOrder[i]);
33         return isValid(root.right, i, end, preOrder);
34     }
35
36     return true;
37 }
38
39
40 function leftBottom(root, level) {
41     if (root.left) return leftBottom(root.left, level + 1);
42     if (level === 0) return 0;
43     if (root.right) return leftBottom(root.right, level + 1);
44     return root.value;
45 }
```

```
46
47
48     function rightBottom(root, level) {
49         if (root.right) return rightBottom(root.right, level + 1);
50         if (level === 0) return 0;
51         if (root.left) return rightBottom(root.left, level + 1);
52         return root.value;
53     }
54
55
56     const readline = require("readline");
57     const rl = readline.createInterface({ input: process.stdin });
58
59     rl.on("line", (line) => {
60         const preOrder = line.split(" ").map(Number);
61         const root = new Node(preOrder[0]);
62
63         if (!isValid(root, 0, preOrder.length - 1, preOrder)) {
64             console.log("0 0 0");
65         } else {
66             console.log(`1 ${leftBottom(root, 0)} ${rightBottom(root, 0)}`);
67         }
68         rl.close();
69     });
});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11
12 type Node struct {
13     value int
14     left  *Node
15     right *Node
16 }
17
18
19 func isValid(root *Node, start, end int, preOrder []int) bool {
20     if start == end {
21         return true
22     }
23
24     i := start + 1
25
26     for i <= end && preOrder[i] < root.value {
27         i++
28     }
29
30     j := i
31
32     for j <= end && preOrder[j] > root.value {
33         j++
34     }
35
36
37     if j <= end {
38         return false
39     }
40
41
42     if i > start+1 {
43         root.left = &Node{value: preOrder[start+1]}
44         if !isValid(root.left, start+1, i-1, preOrder) {
45             return false
46         }
47     }
48 }
```

```

46     }
47 }
48
49
50 if i <= end {
51     root.right = &Node{value: preOrder[i]}
52     return isValid(root.right, i, end, preOrder)
53 }
54
55     return true
56 }
57
58
59 func leftBottom(root *Node, level int) int {
60     if root.left != nil {
61         return leftBottom(root.left, level+1)
62     }
63     if level == 0 {
64         return 0
65     }
66     if root.right != nil {
67         return leftBottom(root.right, level+1)
68     }
69     return root.value
70 }
71
72
73 func rightBottom(root *Node, level int) int {
74     if root.right != nil {
75         return rightBottom(root.right, level+1)
76     }
77     if level == 0 {
78         return 0
79     }
80     if root.left != nil {
81         return rightBottom(root.left, level+1)
82     }
83     return root.value
84 }
85
86 func main() {
87
88     reader := bufio.NewReader(os.Stdin)
89     line, _ := reader.ReadString('\n')
90     line = strings.TrimSpace(line)
91
92
93     strNums := strings.Split(line, " ")

```

```
94     preOrder := make([]int, len(strNums))
95     for i, str := range strNums {
96         preOrder[i], _ = strconv.Atoi(str)
97     }
98
99     root := &Node{value: preOrder[0]}
100
101
102    if !isValid(root, 0, len(preOrder)-1, preOrder) {
103        fmt.Println("0 0 0")
104    } else {
105        fmt.Printf("1 %d %d\n", leftBottom(root, 0), rightBottom(root, 0))
106    }
107 }
```

| 来自: 华为OD机考 2025C卷 – 二维伞的雨滴效应 (C++ & Python & JAVA & JS & GO)-CSDN博客

# 华为OD机考 2025 C卷 - 分苹果 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 分苹果

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

| 华为OD机试2025C卷 100分题型

### 题目描述

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:

- $1 \leq$  总苹果数量  $\leq 20000$
- $1 \leq$  每个苹果重量  $\leq 10000$

### 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 |  
1 3  
2 3 5 6
```

#### 输出

```
Plain Text |  
1 11
```

## 示例2

### 输入

```
Plain Text |  
1 8  
2 7258 6579 2602 6716 3050 3564 5396 1773
```

### 输出

```
Plain Text |  
1 35165
```

## 题解

思路： A进行运算实际上是 异或运算。

- 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,  $x \wedge x = 0$ ，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.
- 现在考虑什么时候B能收到最大重量苹果.考虑一个公式  $x \wedge 0 = x$ ，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++

```
1 #include <ctime>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 using namespace std;
9
10
11 // 通用 split 函数
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 int main() {
27     int n ;
28     cin >> n;
29     vector<int> ans(n, 0);
30     int sum = 0;
31     int totle = 0;
32     int minValue = 20000;
33     for (int i = 0; i < n; i++) {
34         int tmp;
35         cin >> tmp;
36         sum = sum ^ tmp;
37         ans[i] = tmp;
38         totle += ans[i];
39         minValue = min(minValue, ans[i]);
40     }
41     // 如果所有异或运算的结果不为0, 说明不会满足A的要求
42     if (sum != 0) {
43         cout << -1;
44     } else {
```

```
46         // a ^ a = 0 , 让A 取最小值
47         cout << totle - minValue;
48     }
49     return 0;
50 }
```

## Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取 n
8         int n = scanner.nextInt();
9         int[] ans = new int[n];
10        int sum = 0;
11        int total = 0;
12        int minValue = 20000;
13
14        for (int i = 0; i < n; i++) {
15            int tmp = scanner.nextInt();
16            sum ^= tmp; // 计算异或和
17            ans[i] = tmp;
18            total += tmp; // 计算总和
19            minValue = Math.min(minValue, tmp); // 取最小值
20        }
21
22        // 如果异或和不为 0, 说明无法满足 A 的要求
23        if (sum != 0) {
24            System.out.println(-1);
25        } else {
26            // 利用异或性质, 取出最小值
27            System.out.println(total - minValue);
28        }
29
30        scanner.close();
31    }
32 }
```

Plain Text

## Python

```
1 import sys
2
3 def main():
4     # 读取 n
5     n = int(sys.stdin.readline().strip())
6
7     ans = []
8     total = 0
9     xor_sum = 0
10    min_value = 20000
11
12    # 读取数据并计算异或值、总和和最小值
13    numbers = list(map(int, sys.stdin.readline().strip().split()))
14    for num in numbers:
15        xor_sum ^= num
16        total += num
17        min_value = min(min_value, num)
18
19    # 如果异或值不为 0, 输出 -1, 否则输出 total - min_value
20    if xor_sum != 0:
21        print(-1)
22    else:
23        print(total - min_value)
24
25 if __name__ == "__main__":
26     main()
```

## JavaScript

```
1 process.stdin.resume();
2 process.stdin.setEncoding("utf-8");
3
4 let input = "";
5 process.stdin.on("data", function (chunk) {
6     input += chunk;
7 });
8
9 process.stdin.on("end", function () {
10    let lines = input.trim().split("\n");
11
12    // 读取 n
13    let n = parseInt(lines[0]);
14    let numbers = lines[1].split(" ").map(Number);
15
16    let sum = 0;
17    let total = 0;
18    let minValue = 20000;
19
20    // 计算异或值、总和、最小值
21    for (let i = 0; i < n; i++) {
22        sum ^= numbers[i];
23        total += numbers[i];
24        minValue = Math.min(minValue, numbers[i]);
25    }
26
27    // 如果异或值不为 0，输出 -1，否则输出 total - minValue
28    if (sum !== 0) {
29        console.log(-1);
30    } else {
31        console.log(total - minValue);
32    }
33});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 使用 bufio 进行高效输入
13     reader := bufio.NewReader(os.Stdin)
14
15     // 读取 n
16     line, _ := reader.ReadString('\n')
17     n, _ := strconv.Atoi(strings.TrimSpace(line))
18
19     // 读取所有数字
20     line, _ = reader.ReadString('\n')
21     numsStr := strings.Fields(line)
22
23     nums := make([]int, n)
24     sum := 0
25     total := 0
26     minValue := 20000
27
28     // 计算异或值、总和、最小值
29     for i := 0; i < n; i++ {
30         nums[i], _ = strconv.Atoi(numsStr[i])
31         sum ^= nums[i]
32         total += nums[i]
33         if nums[i] < minValue {
34             minValue = nums[i]
35         }
36     }
37
38     // 如果异或值不为 0，输出 -1，否则输出 total - minValue
39     if sum != 0 {
40         fmt.Println(-1)
41     } else {
42         fmt.Println(total - minValue)
43     }
44 }
```

# 分苹果

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

华为OD机试2025C卷 100分题型

## 题目描述

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：

- $1 \leq$  总苹果数量  $\leq 20000$
- $1 \leq$  每个苹果重量  $\leq 10000$

## 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

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

### 输出

```
1 11
```

## 示例2

### 输入

Plain Text |

```
1 8
2 7258 6579 2602 6716 3050 3564 5396 1773
```

## 输出

Plain Text |

```
1 35165
```

## 题解

思路： A进行运算实际上是 异或运算。

- 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,  $x \wedge x = 0$ ，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.
- 现在考虑什么时候B能收到最大重量苹果.考虑一个公式  $x \wedge 0 = x$ ，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++

```
1 #include <ctime>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 using namespace std;
9
10
11 // 通用 split 函数
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 int main() {
27     int n ;
28     cin >> n;
29     vector<int> ans(n, 0);
30     int sum = 0;
31     int totle = 0;
32     int minValue = 20000;
33     for (int i = 0; i < n; i++) {
34         int tmp;
35         cin >> tmp;
36         sum = sum ^ tmp;
37         ans[i] = tmp;
38         totle += ans[i];
39         minValue = min(minValue, ans[i]);
40     }
41     // 如果所有异或运算的结果不为0, 说明不会满足A的要求
42     if (sum != 0) {
43         cout << -1;
44     } else {
```

```
46         // a ^ a = 0 , 让A 取最小值
47         cout << totle - minValue;
48     }
49     return 0;
50 }
```

## Java

```
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取 n
8         int n = scanner.nextInt();
9         int[] ans = new int[n];
10        int sum = 0;
11        int total = 0;
12        int minValue = 20000;
13
14        for (int i = 0; i < n; i++) {
15            int tmp = scanner.nextInt();
16            sum ^= tmp; // 计算异或和
17            ans[i] = tmp;
18            total += tmp; // 计算总和
19            minValue = Math.min(minValue, tmp); // 取最小值
20        }
21
22        // 如果异或和不为 0, 说明无法满足 A 的要求
23        if (sum != 0) {
24            System.out.println(-1);
25        } else {
26            // 利用异或性质, 取出最小值
27            System.out.println(total - minValue);
28        }
29
30        scanner.close();
31    }
32 }
```

Plain Text |

## Python

```
1 import sys
2
3 def main():
4     # 读取 n
5     n = int(sys.stdin.readline().strip())
6
7     ans = []
8     total = 0
9     xor_sum = 0
10    min_value = 20000
11
12    # 读取数据并计算异或值、总和和最小值
13    numbers = list(map(int, sys.stdin.readline().strip().split()))
14    for num in numbers:
15        xor_sum ^= num
16        total += num
17        min_value = min(min_value, num)
18
19    # 如果异或值不为 0, 输出 -1, 否则输出 total - min_value
20    if xor_sum != 0:
21        print(-1)
22    else:
23        print(total - min_value)
24
25 if __name__ == "__main__":
26     main()
```

## JavaScript

```
1 process.stdin.resume();
2 process.stdin.setEncoding("utf-8");
3
4 let input = "";
5 process.stdin.on("data", function (chunk) {
6     input += chunk;
7 });
8
9 process.stdin.on("end", function () {
10    let lines = input.trim().split("\n");
11
12    // 读取 n
13    let n = parseInt(lines[0]);
14    let numbers = lines[1].split(" ").map(Number);
15
16    let sum = 0;
17    let total = 0;
18    let minValue = 20000;
19
20    // 计算异或值、总和、最小值
21    for (let i = 0; i < n; i++) {
22        sum ^= numbers[i];
23        total += numbers[i];
24        minValue = Math.min(minValue, numbers[i]);
25    }
26
27    // 如果异或值不为 0，输出 -1，否则输出 total - minValue
28    if (sum !== 0) {
29        console.log(-1);
30    } else {
31        console.log(total - minValue);
32    }
33});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 使用 bufio 进行高效输入
13     reader := bufio.NewReader(os.Stdin)
14
15     // 读取 n
16     line, _ := reader.ReadString('\n')
17     n, _ := strconv.Atoi(strings.TrimSpace(line))
18
19     // 读取所有数字
20     line, _ = reader.ReadString('\n')
21     numsStr := strings.Fields(line)
22
23     nums := make([]int, n)
24     sum := 0
25     total := 0
26     minValue := 20000
27
28     // 计算异或值、总和、最小值
29     for i := 0; i < n; i++ {
30         nums[i], _ = strconv.Atoi(numsStr[i])
31         sum ^= nums[i]
32         total += nums[i]
33         if nums[i] < minValue {
34             minValue = nums[i]
35         }
36     }
37
38     // 如果异或值不为 0，输出 -1，否则输出 total - minValue
39     if sum != 0 {
40         fmt.Println(-1)
41     } else {
42         fmt.Println(total - minValue)
43     }
44 }
```

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

## 分苹果

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

| 华为OD机试2025C卷 100分题型

### 题目描述

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:

- $1 \leq$  总苹果数量  $\leq 20000$
- $1 \leq$  每个苹果重量  $\leq 10000$

### 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  
1 3  
2 3 5 6
```

#### 输出

```
▼ Plain Text  
1 11
```

## 示例2

### 输入

```
Plain Text |  
1 8  
2 7258 6579 2602 6716 3050 3564 5396 1773
```

### 输出

```
Plain Text |  
1 35165
```

## 题解

思路： A进行运算实际上是 **异或运算**。

- 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,  $x \wedge x = 0$ ，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.
- 现在考虑什么时候B能收到最大重量苹果.考虑一个公式  $x \wedge 0 = x$ ，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++

```
1 #include <ctime>
2 #include<iostream>
3 #include<vector>
4 #include<string>
5 #include <utility>
6 #include <sstream>
7 #include<algorithm>
8 using namespace std;
9
10
11 // 通用 split 函数
12 vector<string> split(const string& str, const string& delimiter) {
13     vector<string> result;
14     size_t start = 0;
15     size_t end = str.find(delimiter);
16     while (end != string::npos) {
17         result.push_back(str.substr(start, end - start));
18         start = end + delimiter.length();
19         end = str.find(delimiter, start);
20     }
21     // 添加最后一个部分
22     result.push_back(str.substr(start));
23     return result;
24 }
25
26 int main() {
27     int n ;
28     cin >> n;
29     vector<int> ans(n, 0);
30     int sum = 0;
31     int totle = 0;
32     int minValue = 20000;
33     for (int i = 0; i < n; i++) {
34         int tmp;
35         cin >> tmp;
36         sum = sum ^ tmp;
37         ans[i] = tmp;
38         totle += ans[i];
39         minValue = min(minValue, ans[i]);
40     }
41     // 如果所有异或运算的结果不为0, 说明不会满足A的要求
42     if (sum != 0) {
43         cout << -1;
44     } else {
```

```
46         // a ^ a = 0 , 让A 取最小值
47         cout << totle - minValue;
48     }
49     return 0;
50 }
```

## Java

```
Plain Text | ▾
1 import java.util.*;
2
3 public class Main {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         // 读取 n
8         int n = scanner.nextInt();
9         int[] ans = new int[n];
10        int sum = 0;
11        int total = 0;
12        int minValue = 20000;
13
14        for (int i = 0; i < n; i++) {
15            int tmp = scanner.nextInt();
16            sum ^= tmp; // 计算异或和
17            ans[i] = tmp;
18            total += tmp; // 计算总和
19            minValue = Math.min(minValue, tmp); // 取最小值
20        }
21
22        // 如果异或和不为 0, 说明无法满足 A 的要求
23        if (sum != 0) {
24            System.out.println(-1);
25        } else {
26            // 利用异或性质, 取出最小值
27            System.out.println(total - minValue);
28        }
29
30        scanner.close();
31    }
32 }
```

## Python

```
1 import sys
2
3 def main():
4     # 读取 n
5     n = int(sys.stdin.readline().strip())
6
7     ans = []
8     total = 0
9     xor_sum = 0
10    min_value = 20000
11
12    # 读取数据并计算异或值、总和和最小值
13    numbers = list(map(int, sys.stdin.readline().strip().split()))
14    for num in numbers:
15        xor_sum ^= num
16        total += num
17        min_value = min(min_value, num)
18
19    # 如果异或值不为 0, 输出 -1, 否则输出 total - min_value
20    if xor_sum != 0:
21        print(-1)
22    else:
23        print(total - min_value)
24
25 if __name__ == "__main__":
26     main()
```

## JavaScript

```
1 process.stdin.resume();
2 process.stdin.setEncoding("utf-8");
3
4 let input = "";
5 process.stdin.on("data", function (chunk) {
6     input += chunk;
7 });
8
9 process.stdin.on("end", function () {
10    let lines = input.trim().split("\n");
11
12    // 读取 n
13    let n = parseInt(lines[0]);
14    let numbers = lines[1].split(" ").map(Number);
15
16    let sum = 0;
17    let total = 0;
18    let minValue = 20000;
19
20    // 计算异或值、总和、最小值
21    for (let i = 0; i < n; i++) {
22        sum ^= numbers[i];
23        total += numbers[i];
24        minValue = Math.min(minValue, numbers[i]);
25    }
26
27    // 如果异或值不为 0，输出 -1，否则输出 total - minValue
28    if (sum !== 0) {
29        console.log(-1);
30    } else {
31        console.log(total - minValue);
32    }
33});
```

## Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9 )
10
11 func main() {
12     // 使用 bufio 进行高效输入
13     reader := bufio.NewReader(os.Stdin)
14
15     // 读取 n
16     line, _ := reader.ReadString('\n')
17     n, _ := strconv.Atoi(strings.TrimSpace(line))
18
19     // 读取所有数字
20     line, _ = reader.ReadString('\n')
21     numsStr := strings.Fields(line)
22
23     nums := make([]int, n)
24     sum := 0
25     total := 0
26     minValue := 20000
27
28     // 计算异或值、总和、最小值
29     for i := 0; i < n; i++ {
30         nums[i], _ = strconv.Atoi(numsStr[i])
31         sum ^= nums[i]
32         total += nums[i]
33         if nums[i] < minValue {
34             minValue = nums[i]
35         }
36     }
37
38     // 如果异或值不为 0，输出 -1，否则输出 total - minValue
39     if sum != 0 {
40         fmt.Println(-1)
41     } else {
42         fmt.Println(total - minValue)
43     }
44 }
```

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

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

# 华为OD机考 2025C卷 - 5G网络建设 (C++ & Python & JAVA & JS & GO)-CSDN博客

## 5G网络建设

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

| 华为OD机试2025C卷 200分题型

### 题目描述

现需要在某城市进行5G网络建设，已经选取N个地点设置5G基站，编号固定为1到N，接下来需要各个基站之间使用光纤进行连接以确保基站能互联互通，不同基站之间假设光纤的成本各不相同，且有些节点之间已经存在光纤相连。

请你设计算法，计算出能联通这些基站的最小成本是多少。

注意：基站的联通具有传递性，比如基站A与基站B架设了光纤，基站B与基站C也架设了光纤，则基站A与基站C视为可以互相联通。

### 输入描述

第一行输入表示基站的个数N，其中：

- $0 < N \leq 20$

第二行输入表示具备光纤直连条件的基站对的数目M，其中：

- $0 < M < N * (N - 1) / 2$

从第三行开始连续输入M行数据，格式为

| X Y Z P

其中：

X, Y 表示基站的编号

- $0 < X \leq N$
- $0 < Y \leq N$
- $X \neq Y$

Z 表示在 X、Y之间架设光纤的成本

- $0 < Z < 100$

P 表示是否已存在光纤连接，0 表示未连接，1表示已连接

### 输出描述

如果给定条件，可以建设成功互联互通的5G网络，则输出最小的建设成本

如果给定条件，无法建设成功互联互通的5G网络，则输出 -1

## 用例1

### 输入

```
Plain Text |  
1 3  
2 3  
3 1 2 3 0  
4 1 3 1 0  
5 2 3 5 0
```

### 输出

```
Plain Text |  
1 4
```

### 说明

| 只需要在1, 2以及1, 3基站之间铺设光纤，其成本为3+1=4

## 用例2

### 输入

```
Plain Text |  
1 3  
2 1  
3 1 2 5 0
```

### 输出

```
Plain Text |  
1 -1
```

### 说明

| 3基站无法与其他基站连接，输出-1

## 用例3

## 输入

```
1 3  
2 3  
3 1 2 3 0  
4 1 3 1 0  
5 2 3 5 1
```

Plain Text

## 输出

```
1 1
```

Plain Text

## 说明

2, 3基站已有光纤相连，只要在1, 3基站之间铺设光纤，其成本为1

## 题解

思路： **最小生成树** 问题

将初始直接相连两个基站的边权设置为0，就是经典的最小生成树问题，可以采用 **prim** 的算法来实现。

prim算法的本质就是 **贪心**，下面简单说说实现逻辑：

1. **prim** 最小生成树算法会把节点分成两类， **属于树中的节点** 和 **不属于树的节点**。整个算法的处理过程就是不断把不属于树的节点加入树中，每次选取的加入树中的节点就是 **和树中节点相连并且边权最小的节点**。代码执行到最后会分为两种情况：**所有节点都加入树中** 或者 **不属于树的节点不和树中任意节点相连，无法加入**。基本逻辑：
  - a. 初始化树节点： 开始可以选取任一节点作为初始节点，标记为**属于树中的节点**。
  - b. 合并节点： 找出与树中任意节点相邻并且距离最小的 **不属于树** 的节点，加入节点，更新合并成本。
  - c. 重复2的步骤，直到 **所有节点都加入树中** 或者 **不属于树的节点不和树中任意节点相连，无法加入** 这两种情况出现，结束执行。
2. 了解1的prim算法逻辑之后，这道题就比较简单，我们接收输入，将初始直接相连的节点边权设置为0，然后执行prim算法就行，如果执行prim出现 **不属于树的节点不和树中节点相连** 输出-1，否则输出合并的最小成本值。

C++

```
1 #include<iostream>
2 #include<vector>
3 #include<string>
4 #include <utility>
5 #include <sstream>
6 #include<algorithm>
7 #include<cmath>
8 #include<map>
9 #include<climits>
10 using namespace std;
11
12 // prim最小生成树算法
13 int prim(vector<vector<int>>& grid, int n) {
14     // 记录最小成本
15     int minCost = 0;
16     // inTreeFlag[i] 表示该节点是否在最小生成树中
17     vector<bool> inTreeFlag(n + 1, false);
18
19     // 初始任意选择一个节点作为树的初始节点,这里选择1
20     inTreeFlag[1] = true;
21     // 记录最小生成树中节点数量
22     int inTreeCount = 1;
23
24     // 记录最小生成树中所有节点到其它节点的最小值
25     vector<int> dist(n + 1);
26     for (int i = 1; i <= n; i++) {
27         dist[i] = grid[1][i];
28     }
29
30     while (inTreeCount < n) {
31         // 初始化最小距离 以及 对应节点索引
32         int minDist = INT_MAX;
33         int nodeIndex = 0;
34
35         // 找出和最小生成树相邻成本消耗最少节点
36         for (int i = 1; i <= n; i++) {
37             if (inTreeFlag[i]) {
38                 continue;
39             }
40             if (dist[i] < minDist) {
41                 minDist = dist[i];
42                 nodeIndex = i;
43             }
44         }
45     }
46 }
```

```

46     // 说明无法再接入新的节点 同时最小生成树节点数量 < n => 不能全部相连
47     if (nodeIndex == 0) {
48         return -1;
49     }
50
51     inTreeFlag[nodeIndex] = true;
52     inTreeCount++;
53     minCost += minDist;
54
55     // 最小生成树节点发生改变 更新dist
56     for (int i = 1; i <= n; i++) {
57         // 跳过
58         if (inTreeFlag[i]) {
59             continue;
60         }
61         // 只需要关注新加的node是否会影响距离即可
62         if (grid[nodeIndex][i] < dist[i]) {
63             dist[i] = grid[nodeIndex][i];
64         }
65     }
66 }
67
68     return minCost;
69 }
70
71 int main() {
72     // 节点数量 边数量
73     int n,m;
74     cin >> n;
75     cin >> m;
76
77     // 构建图 初始默认全不连接 设置为最大值
78     vector<vector<int>> grid(n+1, vector<int>(n+1, INT_MAX));
79     for (int i = 0; i < m; i++) {
80         int x,y,z,p;
81         cin >> x >> y >> z >> p;
82         if (p == 0) {
83             grid[x][y] = z;
84             grid[y][x] = z;
85             // 转换已连接将边权转换为0
86         } else {
87             grid[x][y] = 0;
88             grid[y][x] = 0;
89         }
90     }
91     int res = prim(grid, n);
92     cout << res;
93     return 0;

```

## JAVA

```
1 import java.util.*;
2
3 public class Main {
4     // prim最小生成树算法
5     public static int prim(int[][] grid, int n) {
6         int minCost = 0;
7         boolean[] inTreeFlag = new boolean[n + 1]; // 表示该节点是否在最小生成
树中
8         inTreeFlag[1] = true; // 初始任意选择一个节点作为树的初始节点，这里选择1
9         int inTreeCount = 1; // 最小生成树中节点数量
10
11        int[] dist = new int[n + 1]; // 最小生成树中所有节点到其它节点的最小值
12        System.arraycopy(grid[1], 0, dist, 0, n + 1);
13
14        while (inTreeCount < n) {
15            int minDist = Integer.MAX_VALUE;
16            int nodeIndex = 0;
17
18            // 找出和最小生成树相邻成本消耗最少节点
19            for (int i = 1; i <= n; i++) {
20                if (!inTreeFlag[i] && dist[i] < minDist) {
21                    minDist = dist[i];
22                    nodeIndex = i;
23                }
24            }
25
26            // 无法再接入新的节点 => 不能全部相连
27            if (nodeIndex == 0) return -1;
28
29            inTreeFlag[nodeIndex] = true;
30            inTreeCount++;
31            minCost += minDist;
32
33            // 最小生成树节点发生改变 更新dist
34            for (int i = 1; i <= n; i++) {
35                if (!inTreeFlag[i] && grid[nodeIndex][i] < dist[i]) {
36                    dist[i] = grid[nodeIndex][i];
37                }
38            }
39        }
40
41        return minCost;
42    }
43
44    public static void main(String[] args) {
```

```
45     Scanner sc = new Scanner(System.in);
46     int n = sc.nextInt(); // 节点数量
47     int m = sc.nextInt(); // 边数量
48
49     int[][] grid = new int[n + 1][n + 1];
50     for (int[] row : grid) Arrays.fill(row, Integer.MAX_VALUE);
51
52     for (int i = 0; i < m; i++) {
53         int x = sc.nextInt();
54         int y = sc.nextInt();
55         int z = sc.nextInt();
56         int p = sc.nextInt();
57
58         // 转换已连接将边权转换为0
59         grid[x][y] = grid[y][x] = (p == 1 ? 0 : z);
60     }
61
62     int res = prim(grid, n);
63     System.out.println(res);
64 }
65 }
```

## Python

```
1 import sys
2
3
4 # prim最小生成树算法
5 def prim(grid, n):
6     min_cost = 0
7     in_tree_flag = [False] * (n + 1) # 表示该节点是否在最小生成树中
8     in_tree_flag[1] = True # 初始任意选择一个节点作为树的初始节点,这里选择1
9     in_tree_count = 1 # 最小生成树中节点数量
10
11    dist = grid[1][:] # 最小生成树中所有节点到其它节点的最小值
12
13    while in_tree_count < n:
14        min_dist = float('inf')
15        node_index = 0
16
17        # 找出和最小生成树相邻成本消耗最少节点
18        for i in range(1, n + 1):
19            if not in_tree_flag[i] and dist[i] < min_dist:
20                min_dist = dist[i]
21                node_index = i
22
23        # 无法再接入新的节点 => 不能全部相连
24        if node_index == 0:
25            return -1
26
27        in_tree_flag[node_index] = True
28        in_tree_count += 1
29        min_cost += min_dist
30
31        # 最小生成树节点发生改变 更新dist
32        for i in range(1, n + 1):
33            if not in_tree_flag[i] and grid[node_index][i] < dist[i]:
34                dist[i] = grid[node_index][i]
35
36    return min_cost
37
38
39
40 n = int(input())
41 m = int(input())
42 grid = [[float('inf')]] * (n + 1) for _ in range(n + 1)]
43
44 for _ in range(m):
45     x, y, z, p = map(int, input().split())
```

```
46      # 转换已连接将边权转换为0
47      grid[x][y] = grid[y][x] = 0 if p == 1 else z
48
49  res = prim(grid, n)
50  print(res)
```

## JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4      input: process.stdin,
5      output: process.stdout
6  });
7
8  const inputLines = [];
9  rl.on('line', line => {
10     inputLines.push(line.trim());
11     // 当读取的行数达到预期时处理数据
12     const n = parseInt(inputLines[0]);
13     const m = parseInt(inputLines[1]);
14     if (inputLines.length === m + 2) {
15         rl.close();
16         solve(n, m, inputLines.slice(2));
17     }
18 });
19
20 function solve(n, m, edgeLines) {
21     const INT_MAX = Number.MAX_SAFE_INTEGER;
22
23     // 初始化图，所有边默认设置为无穷大
24     const grid = Array.from({ length: n + 1 }, () => Array(n + 1).fill(INT_MAX));
25
26     for (let i = 0; i < m; i++) {
27         const [x, y, z, p] = edgeLines[i].split(' ').map(Number);
28         if (p === 0) {
29             grid[x][y] = z;
30             grid[y][x] = z;
31         } else {
32             // 已连接边权转换为0
33             grid[x][y] = 0;
34             grid[y][x] = 0;
35         }
36     }
37
38     const res = prim(grid, n);
39     console.log(res);
40 }
41
42 // prim 最小生成树算法
43 function prim(grid, n) {
44     // 记录最小成本
```

```

45     let minCost = 0;
46     // 表示该节点是否在最小生成树中
47     const inTree = Array(n + 1).fill(false);
48     // 初始任意选择一个节点作为树的初始节点, 这里选择1
49     inTree[1] = true;
50     // 最小生成树中节点数量
51     let inTreeCount = 1;
52     // 最小生成树中所有节点到其它节点的最小值
53     const dist = Array(n + 1);
54     for (let i = 1; i <= n; i++) {
55         dist[i] = grid[1][i];
56     }
57
58     while (inTreeCount < n) {
59         let minDist = Number.MAX_SAFE_INTEGER;
60         let node = 0;
61         // 找出和最小生成树相邻成本消耗最少节点
62         for (let i = 1; i <= n; i++) {
63             if (!inTree[i] && dist[i] < minDist) {
64                 minDist = dist[i];
65                 node = i;
66             }
67         }
68         // 无法再接入新的节点 => 不能全部相连
69         if (node === 0) return -1;
70
71         inTree[node] = true;
72         inTreeCount++;
73         minCost += minDist;
74
75         // 最小生成树节点发生改变 更新dist
76         for (let i = 1; i <= n; i++) {
77             // 只涉及新加入节点
78             if (!inTree[i] && grid[node][i] < dist[i]) {
79                 dist[i] = grid[node][i];
80             }
81         }
82     }
83
84     return minCost;
85 }

```

## Go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 func main() {
9     var n, m int
10    fmt.Scan(&n, &m)
11
12    // 初始化图, 边权初始为最大值
13    grid := make([][]int, n+1)
14    for i := range grid {
15        grid[i] = make([]int, n+1)
16        for j := range grid[i] {
17            grid[i][j] = math.MaxInt32
18        }
19    }
20
21    for i := 0; i < m; i++ {
22        var x, y, z, p int
23        fmt.Scan(&x, &y, &z, &p)
24        // 直接相连设置为0
25        if p == 0 {
26            grid[x][y] = z
27            grid[y][x] = z
28        } else {
29            grid[x][y] = 0
30            grid[y][x] = 0
31        }
32    }
33
34    res := prim(grid, n)
35    fmt.Println(res)
36 }
37
38 // prim最小生成树算法
39 func prim(grid [][]int, n int) int {
40     // 记录最小成本
41     minCost := 0
42     // 标记节点是否位于树种
43     inTree := make([]bool, n+1)
44     // 初始任意选择一个节点作为树的初始节点, 这里选择1
45     inTree[1] = true
```

```
46     inTreeCount := 1
47
48     // 最小生成树中所有节点到其它节点的最小值
49     dist := make([]int, n+1)
50     for i := 1; i <= n; i++ {
51         dist[i] = grid[1][i]
52     }
53
54     for inTreeCount < n {
55         minDist := math.MaxInt32
56         nodeIndex := 0
57         // 找出和最小生成树相邻成本消耗最少节点
58         for i := 1; i <= n; i++ {
59             if !inTree[i] && dist[i] < minDist {
60                 minDist = dist[i]
61                 nodeIndex = i
62             }
63         }
64
65         // 无法再接入新的节点 => 不能全部相连
66         if nodeIndex == 0 {
67             return -1
68         }
69
70         inTree[nodeIndex] = true
71         inTreeCount++
72         minCost += minDist
73         // 最小生成树节点发生改变 更新dist
74         for i := 1; i <= n; i++ {
75             // 只涉及新加入节点
76             if !inTree[i] && grid[nodeIndex][i] < dist[i] {
77                 dist[i] = grid[nodeIndex][i]
78             }
79         }
80     }
81     return minCost
82 }
```

来自: 华为OD机考 2025C卷 – 5G网络建设 (C++ & Python & JAVA & JS & GO)-CSDN博客