t0729

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 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 <=T<= 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

```
9=9
9=4+5
9=2+3+4
Result:3
```

说明

```
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，
第2个表达式有2个自然数，第2次序输出，
第3个表达式有3个自然数，最后输出。
每个表达式中的自然数都是按递增次序输出的。
数字与符号之间无空格
```

用例2

输入

输出

```
10=10
10=1+2+3+4
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++

Plain Text

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

int main() {
    int t;
    cin >> t;
    vector<vector<int>> res;
    res.push_back({t}); // 单个数本身就是一种表达方式

    // 枚举向数
    for (int num = 2; num * (num - 1) / 2 < t; ++num) {
        // 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
        int diffSum = num * (num - 1) / 2;
        int remain = t - diffSum;
        // 不能整除说明无法拆解为num项
        if (remain % num != 0) continue;
        int begin = remain / num;
        // 大于0的整数才是自然数
        if (begin < 1){
            break;
        }
        vector<int> tmp;
        for (int i = 0; i < num; ++i) {
            tmp.push_back(begin + i);
        }
        res.push_back(tmp);
    }

    // 输出结果
    for (const auto& seq : res) {
        cout << t << "=";
        for (int i = 0; i < seq.size(); ++i) {
            cout << seq[i];
            if (i < seq.size() - 1) cout << "+";
        }
        cout << "\n";
    }
    cout << "Result:" << res.size() << endl;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        List<List<Integer>> res = new ArrayList<>();
        res.add(Collections.singletonList(t)); // 单个数本身就是一种表达方式

        // 枚举项数
        for (int num = 2; num * (num - 1) / 2 < t; ++num) {
            // 等差数列求和公式：与首项差值前num项之和为 num*(num-1)/2
            int diffSum = num * (num - 1) / 2;
            int remain = t - diffSum;

            // 不能整除说明无法拆解为num项
            if (remain % num != 0) continue;

            int begin = remain / num;
            // 大于0的整数才是自然数
            if (begin < 1) break;

            List<Integer> tmp = new ArrayList<>();
            for (int i = 0; i < num; ++i) {
                tmp.add(begin + i);
            }
            res.add(tmp);
        }

        // 输出结果
        for (List<Integer> seq : res) {
            System.out.print(t + "=");
            for (int i = 0; i < seq.size(); ++i) {
                System.out.print(seq.get(i));
                if (i < seq.size() - 1) System.out.print("+");
            }
            System.out.println();
        }
        System.out.println("Result:" + res.size());
    }
}
```

Python

```
Plain Text
```

```
t = int(input())
res = []
res.append([t])


num = 2
while num * (num - 1) // 2 < t:

    diff_sum = num * (num - 1) // 2
    remain = t - diff_sum


    if remain % num != 0:
        num += 1
        continue

    begin = remain // num
    if begin < 1:
        break

    tmp = [begin + i for i in range(num)]
    res.append(tmp)
    num += 1


for seq in res:
    print(f"{t}=" + "+".join(map(str, seq)))
print(f"Result:{len(res)}")
```

JavaScript

```
Plain Text
```

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

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

let input = [];
rl.on('line', (line) => {
    input.push(line);
}).on('close', () => {
    let t = parseInt(input[0]);
    let res = [];
    res.push([t]);


    for (let num = 2; num * (num - 1) / 2 < t; ++num) {

        let diffSum = num * (num - 1) / 2;
        let remain = t - diffSum;

        if (remain % num !== 0) continue;

        let begin = remain / num;
        if (begin < 1) break;

        let tmp = [];
        for (let i = 0; i < num; ++i) {
            tmp.push(begin + i);
        }
        res.push(tmp);
    }


    for (let seq of res) {
        console.log(`${t}=${seq.join('+')}`);
    }
    console.log(`Result:${res.length}`);
});
```

Go

Plain Text

```
package main

import (
    "fmt"
)

func main() {
    var t int
    fmt.Scan(&t)
    res := [][]int{{t}}


    for num := 2; num*(num-1)/2 < t; num++ {

        diffSum := num * (num - 1) / 2
        remain := t - diffSum

        if remain%num != 0 {
            continue
        }

        begin := remain / num
        if begin < 1 {
            break
        }

        tmp := make([]int, num)
        for i := 0; i < num; i++ {
            tmp[i] = begin + i
        }
        res = append(res, tmp)
    }


    for _, seq := range res {
        fmt.Printf("%d=", t)
        for i, val := range seq {
            if i != 0 {
                fmt.Print("+")
            }
            fmt.Print(val)
        }
        fmt.Println()
    }
```

```
45        fmt.Printf("Result:%d\n", len(res))
46    }
```

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， `sim += right, right++`
   c. `sum == t` : 找到一个合法序列 `[left, rihgt)` .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++

Plain Text

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

int main() {
    int t;
    cin >> t;
    vector<vector<int>> res;
    vector<int> sequence(t + 1);
    for (int i = 0; i <= t; i++) {
        sequence[i] = i + 1;
    }
    // 使用双指针 [left, right)属于区域中的值
    int left = 0, right = 1;
    int sum = sequence[left];
    while (right <= t) {
        if (sum < t) {
            sum += sequence[right];
            right++;
        } else if (sum > t) {
            sum -= sequence[left];
            left++;
        } else {
            vector<int> tmp(sequence.begin() + left, sequence.begin() + ri
ght);
            res.push_back(tmp);
            //移动左指针
            sum -= sequence[left];
            left++;
        }
    }
    // 根据长度进行排序
    sort(res.begin(), res.end(), [](const vector<int> &a, const vector<int
> &b) {
        return a.size() < b.size();
    });

    // 输出结果
    for (const auto& seq : res) {
        cout << t << "=";
        for (int i = 0; i < seq.size(); ++i) {
            cout << seq[i];
            if (i < seq.size() - 1) cout << "+";
```

```
            }
            cout << "\n";
        }
        cout << "Result:" << res.size() << endl;

    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int t = sc.nextInt();
        List<List<Integer>> res = new ArrayList<>();
        int[] sequence = new int[t + 1];
        for (int i = 0; i <= t; i++) {
            sequence[i] = i + 1;
        }

        // 使用双指针 [left, right) 属于区域中的值
        int left = 0, right = 1;
        int sum = sequence[left];
        while (right <= t) {
            if (sum < t) {
                sum += sequence[right];
                right++;
            } else if (sum > t) {
                sum -= sequence[left];
                left++;
            } else {
                List<Integer> tmp = new ArrayList<>();
                for (int i = left; i < right; i++) {
                    tmp.add(sequence[i]);
                }
                res.add(tmp);
                // 移动左指针
                sum -= sequence[left];
                left++;
            }
        }

        // 根据长度排序
        res.sort(Comparator.comparingInt(List::size));

        // 输出结果
        for (List<Integer> seq : res) {
            System.out.print(t + "=");
            for (int i = 0; i < seq.size(); i++) {
                System.out.print(seq.get(i));
                if (i < seq.size() - 1) System.out.print("+");
            }
```

```
            System.out.println();
        }
        System.out.println("Result:" + res.size());
    }
}
```

Python

Plain Text

```
t = int(input())
res = []
sequence = [i + 1 for i in range(t + 1)]

left, right = 0, 1
s = sequence[left]
while right <= t:
    if s < t:
        s += sequence[right]
        right += 1
    elif s > t:
        s -= sequence[left]
        left += 1
    else:
        tmp = sequence[left:right]
        res.append(tmp)

        s -= sequence[left]
        left += 1


res.sort(key=len)


for seq in res:
    print(f"{t}=" + "+".join(map(str, seq)))
print("Result:" + str(len(res)))
```

JavaScript

Plain Text

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

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

let inputLines = [];

rl.on('line', function (line) {
    inputLines.push(line.trim());
    if (inputLines.length === 1) {
        main(parseInt(inputLines[0]));
        rl.close();
    }
});

function main(t) {
    let res = [];
    let sequence = Array.from({ length: t + 1 }, (_, i) => i + 1);


    let left = 0, right = 1;
    let sum = sequence[left];

    while (right <= t) {
        if (sum < t) {
            sum += sequence[right++];
        } else if (sum > t) {
            sum -= sequence[left++];
        } else {
            let tmp = sequence.slice(left, right);
            res.push(tmp);
            sum -= sequence[left++];
        }
    }


    res.sort((a, b) => a.length - b.length);


    for (let seq of res) {
        console.log(`${t}=` + seq.join('+'));
    }
```

```
45        console.log("Result:" + res.length);
46    }
```

Go

Plain Text

```
package main

import (
    "fmt"
    "sort"
)

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

    res := [][]int{}
    sequence := make([]int, t+1)
    for i := 0; i <= t; i++ {
        sequence[i] = i + 1
    }


    left, right := 0, 1
    sum := sequence[left]
    for right <= t {
        if sum < t {
            sum += sequence[right]
            right++
        } else if sum > t {
            sum -= sequence[left]
            left++
        } else {
            tmp := make([]int, right-left)
            copy(tmp, sequence[left:right])
            res = append(res, tmp)
            sum -= sequence[left]
            left++
        }
    }


    sort.Slice(res, func(i, j int) bool {
        return len(res[i]) < len(res[j])
    })


    for _, seq := range res {
        fmt.Printf("%d=", t)
```

```
45          for i := 0; i < len(seq); i++ {
46              fmt.Print(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 ⩽ R，C ⩽ 20
- 0 ⩽ S ⩽ 65535

用例1

输入

输出

说明

用例2

输入

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

输出

说明

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

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} -> {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++

Plain Text

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

int main() {
   int row,col;
   cin >> row >> col;
   vector<vector<int>> cov(row, vector<int>(col,0));

   for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            cin >> cov[i][j];
        }
   }


   vector<int> dp(row * col, 0);
   dp[0] = cov[0][0];

   // 上下左右四个方向
   int direct[4][2] = {
{1,0},{-1,0},{0,1},{0,-1}};

   // 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）
   auto cmp = [&](int a, int b) {
    return dp[a] < dp[b];  // 大顶堆，值越大，优先级越高
   };

   priority_queue<int, vector<int>, decltype(cmp)> pq(cmp);

   pq.push(0);
   while (pq.size() > 0) {
      int index = pq.top();
      pq.pop();
      int x = index / col;
      int y = index % col;
      // 说明[row-1][col-1]已经是最大值，其它的不能使它变得更大
      if (x == row - 1 && y == col -1) {
        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

Plain Text

```
import java.util.*;

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

        int row = scanner.nextInt();
        int col = scanner.nextInt();
        int[][] cov = new int[row][col];

        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                cov[i][j] = scanner.nextInt();
            }
        }

        int[] dp = new int[row * col];
        Arrays.fill(dp, 0);
        dp[0] = cov[0][0];

        // 上下左右四个方向
        int[][] direct = {
{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

        // 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）
        PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> Integer.
compare(dp[b], dp[a]));

        pq.offer(0);

        while (!pq.isEmpty()) {
            int index = pq.poll();
            int x = index / col;
            int y = index % col;

            // 说明 (row-1, col-1) 已经是最大值，其他的不能使它变得更大
            if (x == row - 1 && y == col - 1) {
                break;
            }

            // 遍历四个方向
            for (int[] d : direct) {
                int nextX = x + d[0];
                int nextY = y + d[1];
```

```
44
45                  if (nextX < 0 || nextX >= row || nextY < 0 || nextY >= co
    l) {
46                      continue;
47                  }
48
49                  // 二维索引转换为一维
50                  int pos = nextX * col + nextY;
51                  int w = Math.min(dp[index], cov[nextX][nextY]);
52
53                  // 更新 dp[pos] 的值，并重新加入队列
54                  if (dp[pos] < w) {
55                      dp[pos] = w;
56                      pq.offer(pos);
57                  }
58              }
59          }
60
61          System.out.println(dp[row * col - 1]);
62          scanner.close();
63      }
64  }
```

Python

Plain Text

```
import sys
import heapq


row = int(sys.stdin.readline().strip())
col = int(sys.stdin.readline().strip())


cov = [list(map(int, sys.stdin.readline().strip().split())) for _ in range(row)]


dp = [0] * (row * col)
dp[0] = cov[0][0]


directions = [(1, 0), (-1, 0), (0, 1), (0, -1)]


pq = [(-dp[0], 0)]
heapq.heapify(pq)

while pq:

    neg_val, index = heapq.heappop(pq)
    x, y = divmod(index, col)


    if x == row - 1 and y == col - 1:
        break


    for dx, dy in directions:
        nextX, nextY = x + dx, y + dy


        if 0 <= nextX < row and 0 <= nextY < col:
            pos = nextX * col + nextY
            w = min(-neg_val, cov[nextX][nextY])


            if dp[pos] < w:
                dp[pos] = w
                heapq.heappush(pq, (-dp[pos], pos))
```

```
44
45
46  print(dp[row * col - 1])
```

JavaScript

Plain Text

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

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

let input = [];
rl.on("line", (line) => {
    input.push(line.trim());
}).on("close", () => {
    if (input.length < 2) {

        process.exit(1);
    }


    let row = parseInt(input[0], 10);
    let col = parseInt(input[1], 10);


    if (isNaN(row) || isNaN(col) || row <= 0 || col <= 0) {

        process.exit(1);
    }


    if (input.length - 2 < row) {

        process.exit(1);
    }


    let cov = [];
    for (let i = 2; i < 2 + row; i++) {
        let rowData = input[i].split(/\s+/).map(Number);
        if (rowData.length !== col) {

            process.exit(1);
        }
        cov.push(rowData);
    }


```

```
    let dp = Array(row * col).fill(0);
    dp[0] = cov[0][0];


    let directions = [[1, 0], [-1, 0], [0, 1], [0, -1]];


    class PriorityQueue {
        constructor(compare) {
            this.data = [];
            this.compare = compare;
        }
        push(item) {
            this.data.push(item);
            this.data.sort(this.compare);
        }
        pop() {
            return this.data.shift();
        }
        size() {
            return this.data.length;
        }
    }


    let pq = new PriorityQueue((a, b) => dp[b] - dp[a]);


    pq.push(0);

    while (pq.size() > 0) {
        let index = pq.pop();
        let x = Math.floor(index / col);
        let y = index % col;


        if (x === row - 1 && y === col - 1) break;


        for (let [dx, dy] of directions) {
            let nextX = x + dx;
            let nextY = y + dy;


            if (nextX < 0 || nextX >= row || nextY < 0 || nextY >= col) c
ontinue;

```

```
            let pos = nextX * col + nextY;
            let w = Math.min(dp[index], cov[nextX][nextY]);


            if (dp[pos] < w) {
                dp[pos] = w;
                pq.push(pos);
            }
        }
    }


    console.log(dp[row * col - 1]);
    process.exit(0);
});
```

Go

Plain Text

```
package main

import (
  "container/heap"
  "fmt"
)


type Point struct {
  signal int
  index  int
}


type MaxHeap []Point

func (h MaxHeap) Len() int           { return len(h) }
func (h MaxHeap) Less(i, j int) bool { return h[i].signal > h[j].signal }
func (h MaxHeap) Swap(i, j int)      { h[i], h[j] = h[j], h[i] }

func (h *MaxHeap) Push(x interface{}) {
  *h = append(*h, x.(Point))
}

func (h *MaxHeap) Pop() interface{} {
  old := *h
  n := len(old)
  x := old[n-1]
  *h = old[:n-1]
  return x
}

func main() {
  var row, col int
  fmt.Scan(&row, &col)


  cov := make([][]int, row)
  for i := 0; i < row; i++ {
    cov[i] = make([]int, col)
    for j := 0; j < col; j++ {
      fmt.Scan(&cov[i][j])
    }
  }
```

```
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
```

> 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

```
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。
序列A：4 8 10
序列B：3 6 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：
1）A：4   B：3
2）A：8   B：6
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[aright] > b[bright]: a可以直接获胜， `res+= 1, aright -= 1, bright-=1`
- a[aright] < b[bright]: a肯定会输一场，选择将最小值损失掉，`aleft += 1, bright -= 1, res -= 1`
- a[aright] == b[bright]: 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[aleft] > b[bleft]` 情况下,此时应该移动先考虑左指针, `aleft += 1, bleft += 1, res +=1`
  - `a[aleft] <= b[bleft]` 情况下，a应该使用最小值去对抗b的最大值。

C++

Plain Text

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

int main() {
    int n;
    cin >> n;
    vector<int> aScores(n),bScores(n);
    for (int i = 0; i < n; i++) {
        cin >> aScores[i];
    }
    for (int i = 0; i < n; i++) {
        cin >> bScores[i];
    }
    sort(aScores.begin(), aScores.end());
    sort(bScores.begin(), bScores.end());

    int leftA = 0, rightA = n - 1;
    int leftB = 0, rightB = n - 1;
    int res = 0;

    while (leftB <= rightB) {
        // 最大值相比，a > b 则直接比
        if (aScores[rightA] > bScores[rightB]) {
            rightA--;
            rightB--;
            res += 1;
        // 最大值相比， a < b，无论如何都要输，选择损失最小数
        } else if (aScores[rightA] < bScores[rightB]) {
            rightB--;
            leftA++;
            res -=1;
        } else {
            // 相等情况下 ，a最小值大于b最小值，先考虑左指针移动
            if (aScores[leftA] > bScores[leftB]) {
                res+=1;
                leftA++;
                leftB++;
            // 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
            }else {
```

```
                // a [1 1 1] b[1 1 1] 考虑这个情况
                if (bScores[rightB] > aScores[leftA]) {
                    res -=1;
                }
                leftA++;
                rightB--;
            }
        }
    }

    cout << res;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.Arrays;
import java.util.Scanner;

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

    int n = Integer.parseInt(sc.nextLine());
    int[] a = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
    int[] b = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();

    int result = handle(n, a, b);
    System.out.println(result);
  }

  public static int handle(int n, int[] a, int[] b) {
    Arrays.sort(a);
    Arrays.sort(b);

    int la = 0; // 指向a的最小值index
    int ra = n - 1; // 指向a的最大值index

    int lb = 0; // 指向b的最小值index
    int rb = n - 1; // 指向b的最大值index

    int ans = 0; // 记录a可以获得的分数

    while (la <= ra) {
      if (a[ra] > b[rb]) {
        //  a最大值 > b的最大值 直接比
        ans += 1;
        ra--;
        rb--;
      } else if (a[ra] < b[rb]) {
        // a最大值 < b的最大值, 损失掉a的最小值, 保留最大值
        ans -= 1;
        la++;
        rb--;
      } else {
        // 相等情况下 , a最小值大于b最小值, 先考虑左指针移动
        if (a[la] > b[lb]) {
          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
55        return ans;
56      }
57    }
```

Python

Plain Text

```
n = int(input())
a = list(map(int, input().split()))
b = list(map(int, input().split()))



def handle():
    a.sort()
    b.sort()

    la = 0
    ra = n - 1

    lb = 0
    rb = n - 1

    ans = 0

    while la <= ra:
        if a[ra] > b[rb]:

            ans += 1
            ra -= 1
            rb -= 1
        elif a[ra] < b[rb]:

            ans -= 1
            la += 1
            rb -= 1
        else:

            if a[la] > b[lb]:

                ans += 1
                la += 1
                lb += 1

            else:

                if b[rb] > a[la]:
                    ans -= 1
                la += 1
                rb -= 1

```

```
45        return ans
46
47
48
49  print(handle())
```

JavaScript

Plain Text

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

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

const lines = [];
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length == 3) {
    const n = parseInt(lines[0]);
    const a = lines[1].split(" ").map(Number);
    const b = lines[2].split(" ").map(Number);


    console.log(handle(n, a, b));

    lines.length = 0;
  }
});

function handle(n, a, b) {
  a.sort((a, b) => a - b);
  b.sort((a, b) => a - b);

  let la = 0;
  let ra = n - 1;

  let lb = 0;
  let rb = n - 1;

  let ans = 0;

  while (la <= ra) {
    if (a[ra] > b[rb]) {

      ans += 1;
      ra--;
      rb--;
    } else if (a[ra] < b[rb]) {

      ans -= 1;
```

```
45          la++;
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

Plain Text

```
package main

import (
  "fmt"
  "sort"
)

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

  aScores := make([]int, n)
  bScores := make([]int, n)

  for i := 0; i < n; i++ {
    fmt.Scan(&aScores[i])
  }
  for i := 0; i < n; i++ {
    fmt.Scan(&bScores[i])
  }

  sort.Ints(aScores)
  sort.Ints(bScores)

  leftA, rightA := 0, n-1
  leftB, rightB := 0, n-1
  res := 0

  for leftB <= rightB {

    if aScores[rightA] > bScores[rightB] {
      rightA--
      rightB--
      res++

    } else if aScores[rightA] < bScores[rightB] {
      rightB--
      leftA++
      res--
    } else {

      if aScores[leftA] > bScores[leftB] {
        res++
        leftA++
```

```
            leftB++

        } else {

            if bScores[rightB] > aScores[leftA] {
              res--
            }
            leftA++
            rightB--
        }
      }
    }

    fmt.Println(res)
}
```

> 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 ⩽ 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 Max1 ，单人分到第二多月饼个数是 Max2 ，Max1 – Max2 ⩽ 3 ，
- 单人分到第 n – 1 多月饼个数是 Max(n–1)，单人分到第n多月饼个数是 Max(n) ，Max(n–1) – Max(n) ⩽ 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 ⩽ 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 `number 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++

Plain Text

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

int res = 0;
int m, n;

// number当前员工序号 last 上一个员工分配数量  remain 剩余月饼数量
void dfs(int number, int last, int remain) {
    // 剪枝 剩余数量不满足后续分配 少的情况
    if ((m - number) * last > remain) {
        return;
    }

    // 剪枝 剩余数量不满足后续分配 多的情况
    if ((2 *(last + 3)  + (m - number -1) * 3) * (m - number) < remain * 2) {
        return;
    }


    if (number == m-1)  {
        if (remain - last <= 3 && remain - last >= 0) {
            res++;
        }
        return;
    }
    // 下一项和当前项 0 <= 差值 <= 3
    for (int i = last; i <= last + 3; i++) {
        dfs(number + 1, i, remain - i);
    }
}


int main() {
    cin >> m >> n;
    // 特殊情况
    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

Plain Text

```
import java.util.Scanner;

public class Main {
    static int res = 0;
    static int m, n;

    // number 当前员工序号，last 上一个员工分配数量，remain 剩余月饼数量
    static void dfs(int number, int last, int remain) {
        // 剪枝：剩余数量不满足后续分配（少）
        if ((m - number) * last > remain) return;

        // 剪枝：剩余数量不满足后续分配（多）
        if ((2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remai
n * 2) return;

        if (number == m - 1) {
            if (remain - last <= 3 && remain - last >= 0) res++;
            return;
        }

        // 下一项和当前项差值在 0 到 3
        for (int i = last; i <= last + 3; i++) {
            dfs(number + 1, i, remain - i);
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        m = sc.nextInt();
        n = sc.nextInt();
        if (m == 1) {
            System.out.println(1);
            return;
        }

        for (int i = 1; i <= n / m; i++) {
            dfs(1, i, n - i);
        }
        System.out.println(res);
    }
}
```

Python

Plain Text

```
res = 0
m, n = map(int, input().split())


def dfs(number, last, remain):
    global res

    if (m - number) * last > remain:
        return


    if (2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2:
        return

    if number == m - 1:
        if remain - last <= 3:
            res += 1
        return


    for i in range(last, last + 4):
        dfs(number + 1, i, remain - i)


if m == 1:
    print(1)
else:
    for i in range(1, n // m + 1):
        dfs(1, i, n - i)
    print(res)
```

JavaScript

Plain Text

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

let m = 0, n = 0, res = 0;

function dfs(number, last, remain) {

    if ((m - number) * last > remain) return;


    if ((2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2) return;

    if (number === m - 1) {
        if (remain - last <= 3 && remain - last >= 0) res++;
        return;
    }


    for (let i = last; i <= last + 3; i++) {
        dfs(number + 1, i, remain - i);
    }
}

rl.on('line', line => {
    [m, n] = line.trim().split(' ').map(Number);

    if (m === 1) {
        console.log(1);
    } else {
        for (let i = 1; i <= Math.floor(n / m); i++) {
            dfs(1, i, n - i);
        }
        console.log(res);
    }
});
```

Go

Plain Text

```
package main

import (
  "fmt"
)

var res int
var m, n int


func dfs(number, last, remain int) {

  if (m - number) * last > remain {
    return
  }


  if (2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2 {
    return
  }

  if number == m - 1 {
    if remain - last <= 3 &&  remain - last >= 0{
      res++
    }
    return
  }


  for i := last; i <= last + 3; i++ {
    dfs(number + 1, i, remain - i)
  }
}

func main() {
  fmt.Scan(&m, &n)

  if m == 1 {
    fmt.Println(1)
    return
  }

  for i := 1; i <= n / m; i++ {
    dfs(1, i, n - i)
```

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

> 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

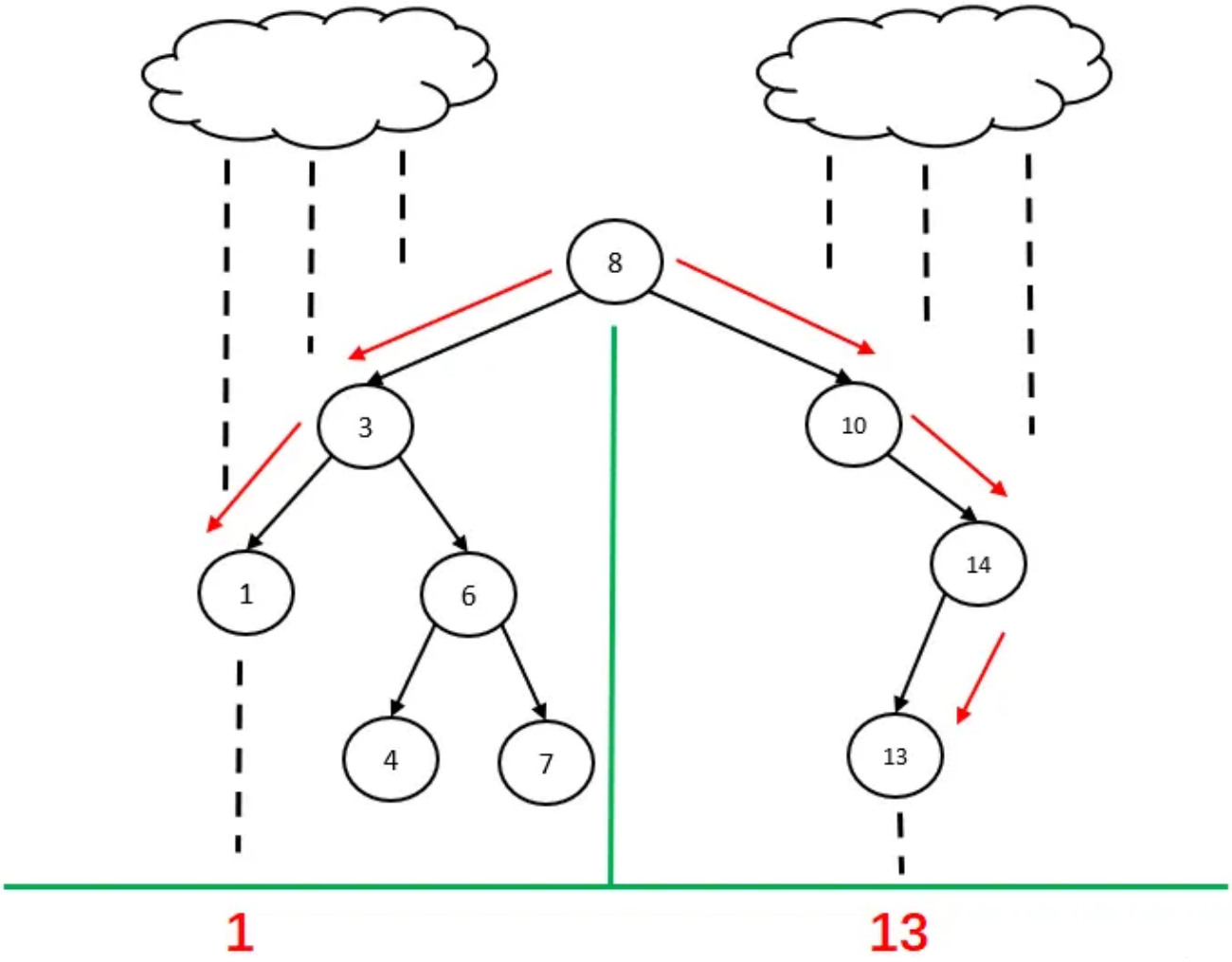
> 华为OD机试2025C卷 200分题型

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++

Plain Text

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

struct Node {
    int value;
    Node* left;
    Node* right;
    Node():left(nullptr), right(nullptr){}
    Node(int value, Node* left, Node* right):value(value),left(left),right(right) {}
    Node(int value):value(value),left(nullptr), right(nullptr){}
};


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

bool isValid(Node* root, int start, int end, vector<int>& preOrder) {
    if (start == end) {
        return true;
    }

    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;
```

JAVA

Plain Text

```
import java.util.*;

class Node {
    int value;
    Node left, right;

    Node(int value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}

public class Main {

    // 判断是否是有效的二叉搜索树的前序遍历
    static boolean isValid(Node root, int start, int end, List<Integer> p
reOrder) {
        if (start == end) {
            return true;
        }

        int i = start + 1;
        // 找到右子树的起点
        while (i <= end && preOrder.get(i) < root.value) {
            i++;
        }

        int j = i;
        // 验证右子树的所有值是否大于根节点
        while (j <= end && preOrder.get(j) > root.value) {
            j++;
        }

        // 若 j 未能完全遍历完，则说明不符合BST规则
        if (j <= end) {
            return false;
        }

        // 递归构建左子树
        if (i > start + 1) {
            root.left = new Node(preOrder.get(start + 1));
            if (!isValid(root.left, start + 1, i - 1, preOrder)) {
                return false;
```

```
            }
        }

        // 递归构建右子树
        if (i <= end) {
            root.right = new Node(preOrder.get(i));
            return isValid(root.right, i, end, preOrder);
        }

        return true;
    }

    // 获取左伞坠
    static int leftBottom(Node root, int level) {
        if (root.left != null) {
            return leftBottom(root.left, level + 1);
        }
        if (level == 0) {
            return 0;
        }
        if (root.right != null) {
            return leftBottom(root.right, level + 1);
        }
        return root.value;
    }

    // 获取右伞坠
    static int rightBottom(Node root, int level) {
        if (root.right != null) {
            return rightBottom(root.right, level + 1);
        }
        if (level == 0) {
            return 0;
        }
        if (root.left != null) {
            return rightBottom(root.left, level + 1);
        }
        return root.value;
    }

    // 中序遍历
    static void dfs(Node root, List<Integer> res) {
        if (root == null) {
            return;
        }
        dfs(root.left, res);
        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     }
```

Python

Plain Text

```
import sys


class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


def is_valid(root, start, end, pre_order):
    if start == end:
        return True

    i = start + 1

    while i <= end and pre_order[i] < root.value:
        i += 1

    j = i

    while j <= end and pre_order[j] > root.value:
        j += 1


    if j <= end:
        return False


    if i > start + 1:
        root.left = Node(pre_order[start + 1])
        if not is_valid(root.left, start + 1, i - 1, pre_order):
            return False


    if i <= end:
        root.right = Node(pre_order[i])
        return is_valid(root.right, i, end, pre_order)

    return True


def left_bottom(root, level):
    if root.left:
```

```
            return left_bottom(root.left, level + 1)
        if level == 0:
            return 0
        if root.right:
            return left_bottom(root.right, level + 1)
        return root.value


    def right_bottom(root, level):
        if root.right:
            return right_bottom(root.right, level + 1)
        if level == 0:
            return 0
        if root.left:
            return right_bottom(root.left, level + 1)
        return root.value


    pre_order = list(map(int, sys.stdin.readline().strip().split()))
    root = Node(pre_order[0])


    if not is_valid(root, 0, len(pre_order) - 1, pre_order):
        print("0 0 0")
    else:
        print(f"1 {left_bottom(root, 0)} {right_bottom(root, 0)}")
```

JavaScript

Plain Text

```
class Node {
    constructor(value) {
        this.value = value;
        this.left = null;
        this.right = null;
    }
}


function isValid(root, start, end, preOrder) {
    if (start === end) return true;

    let i = start + 1;

    while (i <= end && preOrder[i] < root.value) i++;

    let j = i;

    while (j <= end && preOrder[j] > root.value) j++;


    if (j <= end) return false;


    if (i > start + 1) {
        root.left = new Node(preOrder[start + 1]);
        if (!isValid(root.left, start + 1, i - 1, preOrder)) return false;
    }


    if (i <= end) {
        root.right = new Node(preOrder[i]);
        return isValid(root.right, i, end, preOrder);
    }

    return true;
}


function leftBottom(root, level) {
    if (root.left) return leftBottom(root.left, level + 1);
    if (level === 0) return 0;
    if (root.right) return leftBottom(root.right, level + 1);
    return root.value;
```

```
}


function rightBottom(root, level) {
    if (root.right) return rightBottom(root.right, level + 1);
    if (level === 0) return 0;
    if (root.left) return rightBottom(root.left, level + 1);
    return root.value;
}


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

rl.on("line", (line) => {
    const preOrder = line.split(" ").map(Number);
    const root = new Node(preOrder[0]);

    if (!isValid(root, 0, preOrder.length - 1, preOrder)) {
        console.log("0 0 0");
    } else {
        console.log(`1 ${leftBottom(root, 0)} ${rightBottom(root, 0)}`);
    }
    rl.close();
});
```

Go

Plain Text

```
package main

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


type Node struct {
  value int
  left  *Node
  right *Node
}


func isValid(root *Node, start, end int, preOrder []int) bool {
  if start == end {
    return true
  }

  i := start + 1

  for i <= end && preOrder[i] < root.value {
    i++
  }

  j := i

  for j <= end && preOrder[j] > root.value {
    j++
  }


  if j <= end {
    return false
  }


  if i > start+1 {
    root.left = &Node{value: preOrder[start+1]}
    if !isValid(root.left, start+1, i-1, preOrder) {
```

```
        return false
      }
    }


    if i <= end {
      root.right = &Node{value: preOrder[i]}
      return isValid(root.right, i, end, preOrder)
    }

    return true
  }


  func leftBottom(root *Node, level int) int {
    if root.left != nil {
      return leftBottom(root.left, level+1)
    }
    if level == 0 {
      return 0
    }
    if root.right != nil {
      return leftBottom(root.right, level+1)
    }
    return root.value
  }


  func rightBottom(root *Node, level int) int {
    if root.right != nil {
      return rightBottom(root.right, level+1)
    }
    if level == 0 {
      return 0
    }
    if root.left != nil {
      return rightBottom(root.left, level+1)
    }
    return root.value
  }

  func main() {

    reader := bufio.NewReader(os.Stdin)
    line, _ := reader.ReadString('\n')
    line = strings.TrimSpace(line)

```

```
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     }
```

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

输入描述

输入第一行是苹果数量：3

输入第二行是每个苹果重量：3 5 6

输出描述

输出第一行是B获取的苹果总重量：11

示例1

输入

Plain Text

```
3
3 5 6
```

输出

```
11
```

示例2

输入

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

输出

```
35165
```

题解

思路： A进行运算实际上是 `异或运算` 。

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

C++

Plain Text

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


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

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

```

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

Java

Plain Text

```
import java.util.*;

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

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

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

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

        scanner.close();
    }
}
```

Python

Plain Text

```
import sys

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

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

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

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

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

JavaScript

Plain Text

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

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

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

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

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

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

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

Go

Plain Text

```
package main

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

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

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

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

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

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

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

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

输入描述

输入第一行是苹果数量：3

输入第二行是每个苹果重量：3 5 6

输出描述

输出第一行是B获取的苹果总重量：11

示例1

输入

Plain Text

```
3
3 5 6
```

输出

Plain Text

```
11
```

示例2

输入

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

输出

```
35165
```

题解

思路： A进行运算实际上是 `异或运算` 。

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

C++

Plain Text

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


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

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

```

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

Java

```
Plain Text
```

```
import java.util.*;

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

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

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

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

        scanner.close();
    }
}
```

Python

Plain Text

```
import sys

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

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

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

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

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

JavaScript

Plain Text

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

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

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

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

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

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

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

Go

Plain Text

```
package main

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

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

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

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

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

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

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

> 来自: 华为OD机考 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 <= 总苹果数量 <= 20000
- 1 <= 每个苹果重量 <= 10000

输入描述

输入第一行是苹果数量：3

输入第二行是每个苹果重量：3 5 6

输出描述

输出第一行是B获取的苹果总重量：11

示例1

输入

Plain Text

```
3
3 5 6
```

输出

Plain Text

```
11
```

示例2

输入

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

输出

```
35165
```

题解

思路： A进行运算实际上是 `异或运算` 。

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

C++

Plain Text

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


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

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

```

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

Java

Plain Text

```
import java.util.*;

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

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

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

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

        scanner.close();
    }
}
```

Python

Plain Text

```
import sys

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

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

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

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

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

JavaScript

Plain Text

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

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

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

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

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

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

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

Go

Plain Text

```
package main

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

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

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

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

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

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

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

> 来自: 华为OD机考 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 ⩽ 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 ⩽ N
- 0 < Y ⩽ N
- X ≠ 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
3
3
1 2 3 0
1 3 1 0
2 3 5 0
```

输出

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

输出

```Plain Text
-1
```

说明

> 3基站无法与其他基站连接，输出-1

用例3

输入

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

输出

```Plain Text
1
```

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 `和树种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 `不属于树种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++

Plain Text

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

// prim最小生成树算法
int  prim(vector<vector<int>>& grid, int n) {
    // 记录最小成本
    int minCost = 0;
    // inTreeFlag[i] 表示该节点是否在最小生成树中
    vector<bool> inTreeFlag(n + 1, false);

    // 初始任意选择一个节点作为树的初始节点,这里选择1
    inTreeFlag[1] = true;
    // 记录最小生成树中节点数量
    int inTreeCount = 1;

    // 记录最小生成树中所有节点到其它节点的最小值
    vector<int> dist(n + 1);
    for (int i = 1; i <= n; i++) {
        dist[i] = grid[1][i];
    }

    while (inTreeCount < n) {
        // 初始化最小距离 以及 对应节点索引
        int minDist = INT_MAX;
        int nodeIndex = 0;

        // 找出和最小生成树相邻成本消耗最少节点
        for (int i = 1; i <= n; i++) {
            if (inTreeFlag[i]) {
                continue;
            }
            if (dist[i] < minDist) {
                minDist = dist[i];
                nodeIndex = i;
            }
        }
```

```
45
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      return minCost;
68  }
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

Plain Text

```
import java.util.*;

public class Main {
    // prim最小生成树算法
    public static int prim(int[][] grid, int n) {
        int minCost = 0;
        boolean[] inTreeFlag = new boolean[n + 1]; // 表示该节点是否在最小生成树中
        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
        int inTreeCount = 1; // 最小生成树中节点数量

        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
        System.arraycopy(grid[1], 0, dist, 0, n + 1);

        while (inTreeCount < n) {
            int minDist = Integer.MAX_VALUE;
            int nodeIndex = 0;

            // 找出和最小生成树相邻成本消耗最少节点
            for (int i = 1; i <= n; i++) {
                if (!inTreeFlag[i] && dist[i] < minDist) {
                    minDist = dist[i];
                    nodeIndex = i;
                }
            }

            // 无法再接入新的节点 => 不能全部相连
            if (nodeIndex == 0) return -1;

            inTreeFlag[nodeIndex] = true;
            inTreeCount++;
            minCost += minDist;

            // 最小生成树节点发生改变 更新dist
            for (int i = 1; i <= n; i++) {
                if (!inTreeFlag[i] && grid[nodeIndex][i] < dist[i]) {
                    dist[i] = grid[nodeIndex][i];
                }
            }
        }

        return minCost;
    }

```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt(); // 节点数量
        int m = sc.nextInt(); // 边数量

        int[][] grid = new int[n + 1][n + 1];
        for (int[] row : grid) Arrays.fill(row, Integer.MAX_VALUE);

        for (int i = 0; i < m; i++) {
            int x = sc.nextInt();
            int y = sc.nextInt();
            int z = sc.nextInt();
            int p = sc.nextInt();

            // 转换已连接将边权转换为0
            grid[x][y] = grid[y][x] = (p == 1 ? 0 : z);
        }

        int res = prim(grid, n);
        System.out.println(res);
    }
}
```

Python

Plain Text

```
import sys


# prim最小生成树算法
def prim(grid, n):
    min_cost = 0
    in_tree_flag = [False] * (n + 1)  # 表示该节点是否在最小生成树中
    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
    in_tree_count = 1  # 最小生成树中节点数量

    dist = grid[1][:]  # 最小生成树中所有节点到其它节点的最小值

    while in_tree_count < n:
        min_dist = float('inf')
        node_index = 0

        # 找出和最小生成树相邻成本消耗最少节点
        for i in range(1, n + 1):
            if not in_tree_flag[i] and dist[i] < min_dist:
                min_dist = dist[i]
                node_index = i

        # 无法再接入新的节点 => 不能全部相连
        if node_index == 0:
            return -1

        in_tree_flag[node_index] = True
        in_tree_count += 1
        min_cost += min_dist

        # 最小生成树节点发生改变 更新dist
        for i in range(1, n + 1):
            if not in_tree_flag[i] and grid[node_index][i] < dist[i]:
                dist[i] = grid[node_index][i]

    return min_cost



n = int(input())
m = int(input())
grid = [[float('inf')] * (n + 1) for _ in range(n + 1)]

for _ in range(m):
```

```
        x, y, z, p = map(int, input().split())
        # 转换已连接将边权转换为0
        grid[x][y] = grid[y][x] = 0 if p == 1 else z

    res = prim(grid, n)
    print(res)
```

JavaScript

Plain Text

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

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

const inputLines = [];
rl.on('line', line => {
    inputLines.push(line.trim());
    // 当读取的行数达到预期时处理数据
    const n = parseInt(inputLines[0]);
    const m = parseInt(inputLines[1]);
    if (inputLines.length === m + 2) {
        rl.close();
        solve(n, m, inputLines.slice(2));
    }
});

function solve(n, m, edgeLines) {
    const INT_MAX = Number.MAX_SAFE_INTEGER;

    // 初始化图，所有边默认设置为无穷大
    const grid = Array.from({ length: n + 1 }, () => Array(n + 1).fill(INT
_MAX));

    for (let i = 0; i < m; i++) {
        const [x, y, z, p] = edgeLines[i].split(' ').map(Number);
        if (p === 0) {
            grid[x][y] = z;
            grid[y][x] = z;
        } else {
            // 已连接边权转换为0
            grid[x][y] = 0;
            grid[y][x] = 0;
        }
    }

    const res = prim(grid, n);
    console.log(res);
}

// prim 最小生成树算法
function prim(grid, n) {
```

```
    // 记录最小成本
    let minCost = 0;
    // 表示该节点是否在最小生成树中
    const inTree = Array(n + 1).fill(false);
    //初始任意选择一个节点作为树的初始节点,这里选择1
    inTree[1] = true;
    // 最小生成树中节点数量
    let inTreeCount = 1;
    // 最小生成树中所有节点到其它节点的最小值
    const dist = Array(n + 1);
    for (let i = 1; i <= n; i++) {
        dist[i] = grid[1][i];
    }

    while (inTreeCount < n) {
        let minDist = Number.MAX_SAFE_INTEGER;
        let node = 0;
        // 找出和最小生成树相邻成本消耗最少节点
        for (let i = 1; i <= n; i++) {
            if (!inTree[i] && dist[i] < minDist) {
                minDist = dist[i];
                node = i;
            }
        }
        // 无法再接入新的节点 => 不能全部相连
        if (node === 0) return -1;

        inTree[node] = true;
        inTreeCount++;
        minCost += minDist;

        // 最小生成树节点发生改变 更新dist
        for (let i = 1; i <= n; i++) {
            // 只涉及新加入节点
            if (!inTree[i] && grid[node][i] < dist[i]) {
                dist[i] = grid[node][i];
            }
        }
    }

    return minCost;
}
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math"
)

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

  // 初始化图，边权初始为最大值
  grid := make([][]int, n+1)
  for i := range grid {
    grid[i] = make([]int, n+1)
    for j := range grid[i] {
      grid[i][j] = math.MaxInt32
    }
  }

  for i := 0; i < m; i++ {
    var x, y, z, p int
    fmt.Scan(&x, &y, &z, &p)
    // 直接相连设置为0
    if p == 0 {
      grid[x][y] = z
      grid[y][x] = z
    } else {
      grid[x][y] = 0
      grid[y][x] = 0
    }
  }

  res := prim(grid, n)
  fmt.Println(res)
}

// prim最小生成树算法
func prim(grid [][]int, n int) int {
  // 记录最小成本
  minCost := 0
  // 标记节点是否位于树种
  inTree := make([]bool, n+1)
  // 初始任意选择一个节点作为树的初始节点,这里选择1
```

```
    inTree[1] = true
    inTreeCount := 1

    // 最小生成树中所有节点到其它节点的最小值
    dist := make([]int, n+1)
    for i := 1; i <= n; i++ {
      dist[i] = grid[1][i]
    }

    for inTreeCount < n {
      minDist := math.MaxInt32
      nodeIndex := 0
          // 找出和最小生成树相邻成本消耗最少节点
      for i := 1; i <= n; i++ {
        if !inTree[i] && dist[i] < minDist {
          minDist = dist[i]
          nodeIndex = i
        }
      }

      // 无法再接入新的节点 => 不能全部相连
      if nodeIndex == 0 {
        return -1
      }

      inTree[nodeIndex] = true
      inTreeCount++
      minCost += minDist
      // 最小生成树节点发生改变 更新dist
      for i := 1; i <= n; i++ {
        //  只涉及新加入节点
        if !inTree[i] && grid[nodeIndex][i] < dist[i] {
          dist[i] = grid[nodeIndex][i]
        }
      }
    }
    return minCost
  }
```

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