

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

▼		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 $\text{diffSum} = n * (n - 1) / 2$ 的值，然后判断是否可拆分为n项
 - $(t - \text{diffSum}) \% n == 0$:说明可拆分为n项连续和，首项就为 $(t - \text{diffSum}) / n$ 差值为1，构造出对应序列即可。
 - $(t - \text{diffSum}) \% n != 0$:不能拆分，跳过。
 - b. 枚举结束条件有两个 $(n * (n - 1) / 2 >= t)$ 或者 $\text{begin} < 1$ 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。

```
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     // 枚举项数
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 }
```

JAVA

Plain Text

```
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

```
▼ Plain Text |
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
13     if remain % num != 0:
14         num += 1
15         continue
16
17     begin = remain // num
18     if begin < 1:
19         break
20
21     tmp = [begin + i for i in range(num)]
22     res.append(tmp)
23     num += 1
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))
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, `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         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     System.out.println("Result:" + res.size());
48 }
49 }
```

Python

▼ Plain Text

```
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("{t}=" + "+".join(map(str, seq)))
27  print("Result:" + str(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 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
23   let left = 0, right = 1;
24   let sum = sequence[left];
25
26   while (right <= t) {
27     if (sum < t) {
28       sum += sequence[right++];
29     } else if (sum > t) {
30       sum -= sequence[left++];
31     } else {
32       let tmp = sequence.slice(left, right);
33       res.push(tmp);
34       sum -= sequence[left++];
35     }
36   }
37
38
39   res.sort((a, b) => a.length - b.length);
40
41
42   for (let seq of res) {
43     console.log(`${t}=` + seq.join('+'));
44   }
```

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

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
19    left, right := 0, 1
20    sum := sequence[left]
21    for right <= t {
22        if sum < t {
23            sum += sequence[right]
24            right++
25        } else if sum > t {
26            sum -= sequence[left]
27            left++
28        } else {
29            tmp := make([]int, right-left)
30            copy(tmp, sequence[left:right])
31            res = append(res, tmp)
32            sum -= sequence[left]
33            left++
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.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 $8 \rightarrow 4 \rightarrow 5 \rightarrow 9$ 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

		Plain Text									
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						

输出

说明

		Plain Text									
1	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++

```
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];
```

```

44
45         if (nextX < 0 || nextX >= row || nextY < 0 || nextY >= co
1) {
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


```
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
(row)]
10
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    let row = parseInt(input[0], 10);
18    let col = parseInt(input[1], 10);
19
20
21    if (isNaN(row) || isNaN(col) || row <= 0 || col <= 0) {
22
23      process.exit(1);
24    }
25
26
27    if (input.length - 2 < row) {
28
29      process.exit(1);
30    }
31
32
33    let cov = [];
34    for (let i = 2; i < 2 + row; i++) {
35      let rowData = input[i].split(/\s+/).map(Number);
36      if (rowData.length !== col) {
37
38        process.exit(1);
39      }
40      cov.push(rowData);
41    }
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
ontinue;
90

```

```
91         let pos = nextX * col + nextY;
92         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：

- $a[\text{aright}] > b[\text{bright}]$: a可以直接获胜, `res += 1, aright -= 1, bright -= 1`
- $a[\text{aright}] < b[\text{bright}]$: a肯定会输一场，选择将最小值损失掉, `aleft += 1, bright -= 1, res -= 1`
- $a[\text{aright}] == b[\text{bright}]$: 这时候需要考虑最小值的情况：举两个例子 $a[3\ 4]\ b[2\ 4] / a[2\ 5]\ b[3\ 5]$
 - $a[\text{aleft}] > b[\text{bleft}]$ 情况下, 此时应该移动先考虑左指针, `aleft += 1, bleft += 1, res += 1`
 - $a[\text{aleft}] \leq b[\text{bleft}]$ 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
55  cout << res;
56  return 0;
57  }

```

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
55     return ans;
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         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 $Max1$ ，单人分到第二多月饼个数是 $Max2$ ， $Max1 - Max2 \leq 3$ ，
- 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 $Max(n)$ ， $Max(n-1) - Max(n) \leq 3$ ，

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：

$$12 = 1 + 1 + 10 \text{ (Max1 = 10, Max2 = 1, 不满足Max1 - Max2} \leq 3 \text{要求)}$$

$$12 = 1 + 2 + 9 \text{ (Max1 = 9, Max2 = 2, 不满足Max1 - Max2} \leq 3 \text{要求)}$$

$$12 = 1 + 3 + 8 \text{ (Max1 = 8, Max2 = 3, 不满足Max1 - Max2} \leq 3 \text{要求)}$$

$$12 = 1 + 4 + 7 \text{ (Max1 = 7, Max2 = 4, Max3 = 1, 满足要求)}$$

$$12 = 1 + 5 + 6 \text{ (Max1 = 6, Max2 = 5, Max3 = 1, 不满足要求)}$$

$$12 = 2 + 2 + 8 \text{ (Max1 = 8, Max2 = 2, 不满足要求)}$$

$$12 = 2 + 3 + 7 \text{ (Max1 = 7, Max2 = 3, 不满足要求)}$$

$$12 = 2 + 4 + 6 \text{ (Max1 = 6, Max2 = 4, Max3 = 2, 满足要求)}$$

$$12 = 2 + 5 + 5 \text{ (Max1 = 5, Max2 = 2, 满足要求)}$$

$$12 = 3 + 3 + 6 \text{ (Max1 = 6, Max2 = 3, 满足要求)}$$

$$12 = 3 + 4 + 5 \text{ (Max1 = 5, Max2 = 4, Max3 = 3, 满足要求)}$$

$$12 = 4 + 4 + 4 \text{ (Max1 = 4, 满足要求)}$$

题解

思路：DFS 求解

1. 使用 dfs 算法拆分整数为 $a_0 \ a_2 \ \dots \ a_{m-1}$ ，保持这个整数序列递增(可以通过这个性质进行去重)，并且 $a_j - a_i \leq 3$ ，其中 $j - i = 1$ 。

2. 由序列保持递增性质, 确定 a_1 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 $Max(n-1) - Max(n) \leq 3$ 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 *
13  2:
14          return
15
16      if number == m - 1:
17          if remain - last <= 3:
18              res += 1
19          return
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
11      if ((2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain *
12          2) return;
13
14      if (number === m - 1) {
15          if (remain - last <= 3 && remain - last >= 0) res++;
16          return;
17      }
18
19      for (let i = last; i <= last + 3; i++) {
20          dfs(number + 1, i, remain - i);
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  });
```

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
18     if (2 * (last + 3) + (m - number - 1) * 3) * (m - number) < remain * 2 {
19         return
20     }
21
22     if number == m - 1 {
23         if remain - last <= 3 && remain - last >= 0{
24             res++
25         }
26         return
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

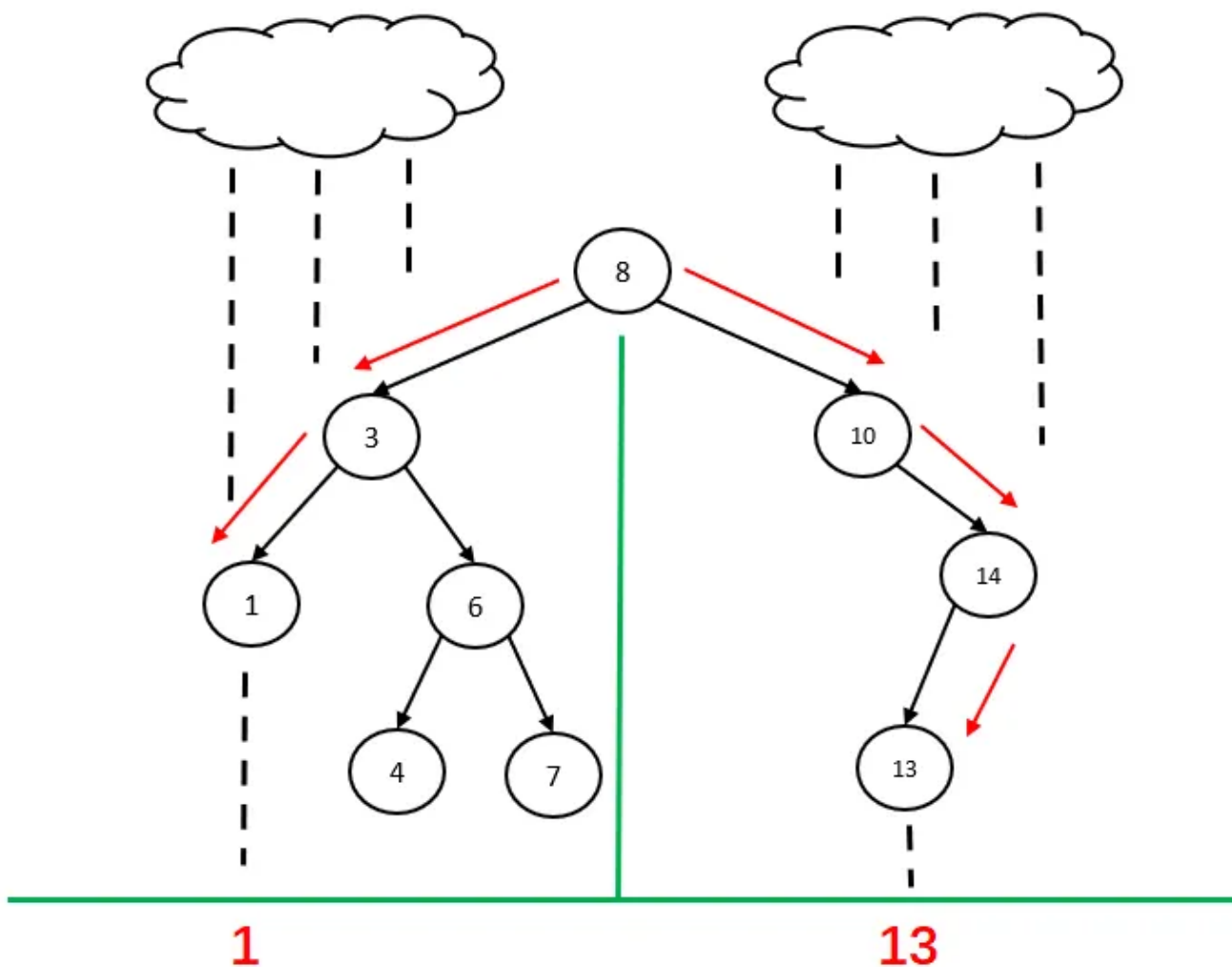
华为OD机试真题目录点击查看: [华为OD机试2025C卷真题题库目录](#) | [机考题库](#) + [算法考点详解](#)

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;
```

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 }
```

```

44         }
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     }

```

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     return True
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
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 <= 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

```
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 。

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, $x \oplus x = 0$, 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 $x \oplus 0 = x$, 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     }
```

```

45     } 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 }
```

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:

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

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 。
- 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, $x \oplus x = 0$, 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 $x \oplus 0 = x$, 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     }
```

```

45     } 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 }
```

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:

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

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 。

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， $x \oplus x = 0$ ，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 $x \oplus 0 = x$ ，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     }
```



```

45     } 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

▼ Plain Text |

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

输出

▼ Plain Text |

```
1 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：

- `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：
 - 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`。
 - 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。
 - 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。
- 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++


```
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     // 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

```
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