

od0625

[华为OD机试 2025 B卷 - 最多等和不相交连续子序列 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025B卷 - 不含101的数 \(C++&Python&JAVA&JS&GO\)-CSDN博客](#)

[华为OD2025B卷 机试 - 最佳对手 \(Java & Python & JS & C++ &GO \)_实力差距最小总和-CSDN博客](#)

[华为OD上机考试2025B卷 - 最佳的出牌方法 \(C++ & Python & JAVA & JS & GO\)_华为od机考2025b卷-CSDN博客](#)

[华为OD 2025 B卷 - 单词倒序 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机考 2025 B卷 - 最短木板长度 \(C++ & Python & JAVA & JS & GO\)_华为od机考b卷-CSDN博客](#)

[华为OD机试 2025 B卷 - 游戏分组 / 王者荣耀 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025B卷 - 打印机队列 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试2025B卷 - 比赛 \(C++ & Python & JAVA & JS & GO\)_华为od机考2025b卷-CSDN博客](#)

[华为OD机试2025B卷 - 密码解密 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

[华为OD机试 2025B卷 - 箱子Z字形摆放 \(C++ & Python & JAVA & JS & GO\)_箱子z字形摆放华为od-CSDN博客](#)

[华为OD机考2025B卷 - 符合条件的元组个数 \(C++ & Python & JAVA & JS & GO\)_华为od机考2025a卷-CSDN博客](#)

[华为OD机试2025B卷 - MVP争夺战 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机试 2025 B卷 - 最多等和不相交连续子序列 (C++ & Python & JAVA & JS & GO)-CSDN 博客

最多等和不相交连续子序列

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

华为OD 2025B卷 200分题型

题目描述

给定一个数组，我们称其中连续的元素为连续子序列，称这些元素的和为连续子序列的和。数组中可能存在几组连续子序列，组内的连续子序列互不相交且有相同的和。求一组连续子序列，组内子序列的数目最多。输出这个数目。

输入描述

第一行输入为数组长度N， $1 \leq N \leq 10^3$
第二行为N个用空格分开的整数 C_i ， $-10^5 \leq C_i \leq 10^5$

输出描述

第一行是一个整数M，表示满足要求的最多的组内子序列的数目。

用例1

输入

	Plain Text
1	10
2	8 8 9 1 9 6 3 9 1 0

输出

	Plain Text
1	4

说明

四个子序列的第一个元素和最后一个元素的下标分别为

2	2
4	4
5	6
7	7

用例2

输入

▼ Plain Text

1	10
2	-1 0 4 -3 6 5 -6 5 -7 -3

输出

▼ Plain Text

1	3
---	---

说明

三个子序列的第一个元素和最后一个元素的下标分别为：

3	3
5	8
9	9

题解

- 思路：前缀和 + 区间问题
1. 题目数据量比较小，可使用 前缀和 + 区间不重叠计算 解答。
 2. 预计算 前缀和 `prefix` 。
 3. 定义一个哈希表 `map<int, int[][]>` 分类 存储连续区间和 和 区间起始坐标和终止坐标 映射关系。这一步通过上一步预先计算的前缀和进行处理。采用两重循环枚举起始坐标 `i`，终止坐标 `j`，`sum = prefix[j] - prefix[i-1]`，对应区间就为 `{i, j}`
 4. 通过第3步得到不同区间和sum可以由哪些连续区间组成，遍历计算不同sum中不重叠区间数量，最大值就为结果。计算不重叠区间数量逻辑如下：
 - 将区间坐标按照终止坐标进行升序排序.定义 `lastEnd` 表示上一个区间的终止坐标,定义 `count` 表示不重叠区间数量。
 - 初始设置 `lastEnd = coor[0].end, count = 1`

- 循环遍历排序后的坐标, 如果 `coord[i].start < lastEnd`, 则 `count += 1, lastEnd = coord[i].end`. 否则直接跳过。
- 不断执行上一步操作, 直到遍历完所有坐标, 则能求出不重叠区间数量。这里用到主要是 贪心的思想。

5. 对所有不同sum的区间坐标执行4 的操作, 就能其中所有相同和的最大不重叠数量。输出结果即可。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<map>
8  using namespace std;
9
10 // 计算不重叠区间数量
11 int calNotJoint(vector<pair<int, int>>& coor) {
12     if (coor.empty()) {
13         return 0;
14     }
15     int count = 1;
16     // 结束时间升序
17     sort(coor.begin(), coor.end(), [](const pair<int, int>& a, const pair<
int, int>& b){
18         return a.second < b.second; // 结束时间升序
19     });
20     int lastEnd = coor[0].second;
21     int n = coor.size();
22     for (int i = 1; i < n; i++) {
23         int start = coor[i].first;
24         int end = coor[i].second;
25         if (lastEnd < start) {
26             count++;
27             lastEnd = end;
28         }
29     }
30     return count;
31 }
32
33
34 int main() {
35     int n;
36     cin >> n;
37     vector<int> nums(n);
38     for (int i = 0; i < n; i++) {
39         cin >>nums[i];
40     }
41
42     vector<int> prefix(n + 1, 0);
43
44     // 计算前缀和
```

```

45     for (int i = 1; i <= n; i++) {
46         prefix[i] = prefix[i-1] + nums[i - 1];
47     }
48     // key 区间和 => 区间(起始坐标,结束坐标)
49     map<int, vector<pair<int, int>>> sumCoorMp;
50     for (int i = 1; i <= n; i++) {
51         for (int j = i; j <= n; j++) {
52             int sum = prefix[j] - prefix[i-1];
53             sumCoorMp[sum].push_back({i, j});
54         }
55     }
56
57     // 求出其中最大不重叠数量
58     int res = 0;
59     for (auto p : sumCoorMp) {
60         vector<pair<int, int>> coor = p.second;
61         res = max(res, calNotJoint(coor));
62     }
63     cout << res;
64     return 0;
65 }

```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 计算不重叠区间数量
5      static int calNotJoint(List<int[]> coor) {
6          if (coor.isEmpty()) return 0;
7          // 按结束时间升序排列
8          coor.sort(Comparator.comparingInt(a -> a[1]));
9          int count = 1;
10         int lastEnd = coor.get(0)[1];
11         for (int i = 1; i < coor.size(); i++) {
12             int start = coor.get(i)[0];
13             int end = coor.get(i)[1];
14             if (lastEnd < start) {
15                 count++;
16                 lastEnd = end;
17             }
18         }
19         return count;
20     }
21
22     public static void main(String[] args) {
23         Scanner sc = new Scanner(System.in);
24         int n = sc.nextInt();
25         int[] nums = new int[n];
26         for (int i = 0; i < n; i++) nums[i] = sc.nextInt();
27
28         int[] prefix = new int[n + 1];
29         // 前缀和构建
30         for (int i = 1; i <= n; i++) {
31             prefix[i] = prefix[i - 1] + nums[i - 1];
32         }
33
34         // key: 区间和 -> 多个坐标区间
35         Map<Integer, List<int[]>> sumCoorMap = new HashMap<>();
36         for (int i = 1; i <= n; i++) {
37             for (int j = i; j <= n; j++) {
38                 int sum = prefix[j] - prefix[i - 1];
39                 sumCoorMap.computeIfAbsent(sum, k -> new ArrayList<>()).ad
40                     d(new int[]{i, j});
41             }
42         }
43
44         int res = 0;
45         for (List<int[]> coor : sumCoorMap.values()) {
```

```
45         res = Math.max(res, calNotJoint(coor));
46     }
47     System.out.println(res);
48 }
49 }
```

Python


```
1 def cal_not_joint(coor):
2     if not coor:
3         return 0
4     # 按结束位置升序排序
5     coor.sort(key=lambda x: x[1])
6     count = 1
7     last_end = coor[0][1]
8     for i in range(1, len(coor)):
9         start, end = coor[i]
10        if last_end < start:
11            count += 1
12            last_end = end
13    return count
14
15 def main():
16     n = int(input())
17     nums = list(map(int, input().split()))
18     prefix = [0] * (n + 1)
19
20     # 计算前缀和
21     for i in range(1, n + 1):
22         prefix[i] = prefix[i - 1] + nums[i - 1]
23
24     # 和 -> 区间列表
25     from collections import defaultdict
26     sum_coor = defaultdict(list)
27
28     for i in range(1, n + 1):
29         for j in range(i, n + 1):
30             s = prefix[j] - prefix[i - 1]
31             sum_coor[s].append((i, j))
32
33     res = 0
34     for coor in sum_coor.values():
35         res = max(res, cal_not_joint(coor))
36     print(res)
37
38     main()
```

JavaScript

```
1 // 引入 readline 用于逐行读取输入
2 const readline = require('readline');
3
4 // 创建输入接口
5 const rl = readline.createInterface({
6   input: process.stdin,
7   output: process.stdout
8 });
9
10 const inputLines = [];
11 rl.on('line', line => {
12   inputLines.push(line.trim());
13 });
14
15 rl.on('close', () => {
16   main(inputLines);
17 });
18
19 // 计算不重叠区间数量 (结束时间升序的贪心算法)
20 function calNotJoint(coor) {
21   if (coor.length === 0) return 0;
22
23   // 按结束位置升序排序
24   coor.sort((a, b) => a[1] - b[1]);
25
26   let count = 1;
27   let lastEnd = coor[0][1];
28
29   for (let i = 1; i < coor.length; i++) {
30     const [start, end] = coor[i];
31     if (lastEnd < start) {
32       count++;
33       lastEnd = end;
34     }
35   }
36
37   return count;
38 }
39
40 function main(lines) {
41   const n = parseInt(lines[0]);
42   const nums = lines[1].split(' ').map(Number);
43
44   // 构造前缀和数组
45   const prefix = new Array(n + 1).fill(0);
```

```

46     for (let i = 1; i <= n; i++) {
47         prefix[i] = prefix[i - 1] + nums[i - 1];
48     }
49
50     // sum -> list of intervals (1-based index)
51     const sumCoorMap = new Map();
52
53     // 枚举所有区间，记录区间和
54     for (let i = 1; i <= n; i++) {
55         for (let j = i; j <= n; j++) {
56             const sum = prefix[j] - prefix[i - 1];
57             if (!sumCoorMap.has(sum)) {
58                 sumCoorMap.set(sum, []);
59             }
60             sumCoorMap.get(sum).push([i, j]);
61         }
62     }
63
64     // 对所有和的区间进行不重叠计数，求最大值
65     let res = 0;
66     for (let coor of sumCoorMap.values()) {
67         res = Math.max(res, calNotJoint(coor));
68     }
69
70     console.log(res);
71 }

```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "sort"
6 )
7
8 func calNotJoint(coor [][]int) int {
9     if len(coor) == 0 {
10         return 0
11     }
12     // 按结束时间升序排序
13     sort.Slice(coor, func(i, j int) bool {
14         return coor[i][1] < coor[j][1]
15     })
16
17     count := 1
18     lastEnd := coor[0][1]
19     for i := 1; i < len(coor); i++ {
20         start := coor[i][0]
21         end := coor[i][1]
22         if lastEnd < start {
23             count++
24             lastEnd = end
25         }
26     }
27     return count
28 }
29
30 func main() {
31     var n int
32     fmt.Scan(&n)
33     nums := make([]int, n)
34     for i := 0; i < n; i++ {
35         fmt.Scan(&nums[i])
36     }
37     // 计算前缀和
38     prefix := make([]int, n+1)
39     for i := 1; i <= n; i++ {
40         prefix[i] = prefix[i-1] + nums[i-1]
41     }
42     // key: 区间和 -> 多个坐标区间
43     sumCoor := make(map[int][][2]int)
44
45     for i := 1; i <= n; i++ {
```

```

46     for j := i; j <= n; j++ {
47         sum := prefix[j] - prefix[i-1]
48         sumCoor[sum] = append(sumCoor[sum], [2]int{i, j})
49     }
50 }
51
52 res := 0
53 for _, coor := range sumCoor {
54     tmp := calNotJoint(coor)
55     if tmp > res {
56         res = tmp
57     }
58 }
59 fmt.Println(res)
60 }

```

来自: [华为OD机试 2025 B卷 – 最多等和不相交连续子序列 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

华为OD机试2025B卷 - 不含101的数

(C++&Python&JAVA&JS&GO)-CSDN博客

不含101的数

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

2025B卷 200分题型

题目描述

小明在学习二进制时，发现了一类不含101的数：

将数字用二进制表示，不能出现 101 。

现在给定一个整数区间 $[l, r]$ ，请问这个区间包含了多少个不含 101 的数？

输入描述

输入的唯一一行包含两个正整数 l, r ($1 \leq l \leq r \leq 10^9$)。

输出描述

输出的唯一一行包含一个整数，表示在 $[l, r]$ 区间内一共有几个不含 101 的数。

用例1

输入

▼	Plain Text
1 1 10	

输出

▼	Plain Text
1 8	

说明

区间 $[1, 10]$ 内，5 的二进制表示为 101，10 的二进制表示为 1010，因此区间 $[1, 10]$ 内有 $10 - 2 = 8$ 个不含 101 的数。

用例2

输入

▼

Plain Text

11020

输出

▼

Plain Text

17

说明

区间 [10,20] 内，满足条件的数字有 [12,14,15,16,17,18,19] 因此答案为 7。

题解

思路： 数位DP + 记忆化搜索

- 1. 定义 `dp[][][]` 缓存数组， `dp[pos][pre][prepre]` 表示当前位置前一位为pre，前第二位为prepre,能够满足不含101的合法数量。用于后续重复访问时，直接返回，减少重复递归，提高代码执行效率。
- 2. DFS的逻辑为：从高位到低位构造二进制数，并统计所有不包含 101 子串的合法方案数，利用记忆化加速 具体逻辑可参照一下代码逻辑。
- 3. 结果就是 R 中 不含101的数的数量 - L中不含101的数的数量

C++

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  #define MAX_LEN 32
6
7  int dp[MAX_LEN][2][2];
8  int binary[MAX_LEN];
9
10 // 递归搜索 pos 当前位置 limit 之前是否为最大值标志(此标志会限制当前取值范围) pre
    前一位的值 前两位值
11 int dfs(int pos, int limit, int pre, int prepre) {
12     if (pos == -1) return 1; // 递归结束, 找到一个有效方案
13     // 利用记忆化数组缓存直接范围
14     if (!limit && dp[pos][pre][prepre] != -1) return dp[pos][pre][prepre];
15     // 之前选择为最大值,
16     int maxDigit = limit ? binary[pos] : 1;
17     int count = 0;
18
19     for (int i = 0; i <= maxDigit; i++) {
20         if (i == 1 && pre == 0 && prepre == 1) continue; // 跳过非法情况
21         // 递归
22         count += dfs(pos - 1, limit && (i == maxDigit), i, pre);
23     }
24     // 记忆化
25     if (!limit) dp[pos][pre][prepre] = count;
26     return count;
27 }
28
29 // 计算 0~num 内的合法数字个数
30 int digitSearch(int num) {
31     int len = 0;
32     memset(dp, -1, sizeof(dp));
33
34     for (int i = 0; i < MAX_LEN; i++) binary[i] = 0;
35
36     // 存储二进制位 用于前一位为最大
37     for (; num; num >>= 1) binary[len++] = num & 1;
38
39     return dfs(len - 1, 1, 0, 0);
40 }
41
42 int main() {
43     int L, R;
44     cin >> L >> R;
```



```
45     cout << digitSearch(R) - digitSearch(L - 1) << endl;  
46     return 0;  
47 }
```

JAVA

```

1  import java.util.Scanner;
2  import java.util.Arrays;
3
4  public class Main {
5      static final int MAX_LEN = 32;
6      static int[][][] dp = new int[MAX_LEN][2][2];
7      static int[] binary = new int[MAX_LEN];
8
9      // 递归搜索 pos 当前位置 limit 之前是否为最大值标志(此标志会限制当前取值范围) pre
e  前一位的值 prepre 前两位值
10     static int dfs(int pos, int limit, int pre, int prepre) {
11         if (pos == -1) return 1; // 递归结束, 找到一个有效方案
12         // 利用记忆化数组缓存直接范围
13         if (limit == 0 && dp[pos][pre][prepre] != -1) return dp[pos][pre]
[prepre];
14
15         int maxDigit = limit == 1 ? binary[pos] : 1;
16         int count = 0;
17
18         for (int i = 0; i <= maxDigit; i++) {
19             if (i == 1 && pre == 0 && prepre == 1) continue; // 跳过非法情况
20             count += dfs(pos - 1, limit == 1 && i == maxDigit ? 1 : 0, i,
pre);
21         }
22
23         if (limit == 0) dp[pos][pre][prepre] = count;
24         return count;
25     }
26
27     // 计算 0~num 内的合法数字个数
28     static int digitSearch(int num) {
29         int len = 0;
30         for (int[][] layer : dp) for (int[] row : layer) Arrays.fill(row,
-1);
31
32         Arrays.fill(binary, 0);
33
34         while (num > 0) {
35             binary[len++] = num & 1;
36             num >>= 1;
37         }
38
39         return dfs(len - 1, 1, 0, 0);
40     }
41

```

```
42     public static void main(String[] args) {  
43         Scanner scanner = new Scanner(System.in);  
44         int L = scanner.nextInt();  
45         int R = scanner.nextInt();  
46         System.out.println(digitSearch(R) - digitSearch(L - 1));  
47         scanner.close();  
48     }  
49 }
```

Python

```
1 import sys
2
3 MAX_LEN = 32
4 dp = [[[-1] * 2 for _ in range(2)] for _ in range(MAX_LEN)]
5 binary = [0] * MAX_LEN
6
7 # 递归搜索 pos 当前位置 limit 之前是否为最大值标志(此标志会限制当前取值范围) pre
  前一位的值 前两位值
8 def dfs(pos, limit, pre, prepre):
9     if pos == -1:
10         return 1 # 递归结束, 找到一个有效方案
11     # 利用记忆化数组缓存直接范围
12     if not limit and dp[pos][pre][prepre] != -1:
13         return dp[pos][pre][prepre]
14
15     maxDigit = binary[pos] if limit else 1
16     count = 0
17
18     for i in range(maxDigit + 1):
19         if i == 1 and pre == 0 and prepre == 1:
20             continue # 跳过非法情况
21         count += dfs(pos - 1, limit and (i == maxDigit), i, pre)
22
23     if not limit:
24         dp[pos][pre][prepre] = count
25     return count
26
27 # 计算 0~num 内的合法数字个数
28 def digitSearch(num):
29     global dp, binary
30     for i in range(MAX_LEN):
31         for j in range(2):
32             for k in range(2):
33                 dp[i][j][k] = -1
34
35     binary = [0] * MAX_LEN
36     len_ = 0
37
38     while num:
39         binary[len_] = num & 1
40         num >>= 1
41         len_ += 1
42
43     return dfs(len_ - 1, 1, 0, 0)
44
```

```
45 if __name__ == "__main__":  
46     L, R = map(int, sys.stdin.readline().split())  
47     print(digitSearch(R) - digitSearch(L - 1))
```

JavaScript

```
1  const readline = require("readline");
2
3  const MAX_LEN = 32;
4  let dp = Array.from({ length: MAX_LEN }, () =>
5      Array.from({ length: 2 }, () => Array(2).fill(-1))
6  );
7  let binary = new Array(MAX_LEN).fill(0);
8
9  // 递归搜索 pos 当前位置 limit 之前是否为最大值标志(此标志会限制当前取值范围) pre
   前一位的值 前两位值
10 function dfs(pos, limit, pre, prepre) {
11     if (pos === -1) return 1; // 递归结束, 找到一个有效方案
12
13     if (!limit && dp[pos][pre][prepre] !== -1) return dp[pos][pre][prepre];
14     // 利用记忆化数组缓存直接范围
15     let maxDigit = limit ? binary[pos] : 1;
16     let count = 0;
17
18     for (let i = 0; i <= maxDigit; i++) {
19         if (i === 1 && pre === 0 && prepre === 1) continue; // 跳过非法情况
20         count += dfs(pos - 1, limit && i === maxDigit, i, pre);
21     }
22
23     if (!limit) dp[pos][pre][prepre] = count;
24     return count;
25 }
26
27 // 计算 0~num 内的合法数字个数
28 function digitSearch(num) {
29     dp = Array.from({ length: MAX_LEN }, () =>
30         Array.from({ length: 2 }, () => Array(2).fill(-1))
31     );
32     binary.fill(0);
33
34     let len = 0;
35     while (num > 0) {
36         binary[len++] = num & 1;
37         num >>= 1;
38     }
39
40     return dfs(len - 1, 1, 0, 0);
41 }
42
43 // 读取输入
```

```
44 const rl = readline.createInterface({
45   input: process.stdin,
46   output: process.stdout,
47 });
48
49 rl.on("line", (line) => {
50   let [L, R] = line.split(" ").map(Number);
51   console.log(digitSearch(R) - digitSearch(L - 1));
52   rl.close();
53 });
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7      "strings"
8      "strconv"
9  )
10
11  const MAX_LEN = 32
12
13  var dp [MAX_LEN][2][2]int
14  var binary [MAX_LEN]int
15
16  // 递归搜索 pos 当前位置 limit 之前是否为最大值标志(此标志会限制当前取值范围) pre
17  // 前一位的值 前两位值
18  func dfs(pos, limit, pre, prepre int) int {
19      if pos == -1 {
20          return 1 // 递归结束, 找到一个有效方案
21      }
22      // 利用记忆化数组缓存直接范围
23      if limit == 0 && dp[pos][pre][prepre] != -1 {
24          return dp[pos][pre][prepre]
25      }
26      maxDigit := 1
27      if limit == 1 {
28          maxDigit = binary[pos]
29      }
30
31      count := 0
32      for i := 0; i <= maxDigit; i++ {
33          if i == 1 && pre == 0 && prepre == 1 {
34              continue // 跳过非法情况
35          }
36          newLimit := 0
37          if limit == 1 && i == maxDigit {
38              newLimit = 1
39          }
40          count += dfs(pos-1, newLimit, i, pre)
41      }
42
43      if limit == 0 {
44          dp[pos][pre][prepre] = count
```



```

45     }
46     return count
47 }
48
49 // 计算 0~num 内的合法数字个数
50 func digitSearch(num int) int {
51     for i := 0; i < MAX_LEN; i++ {
52         for j := 0; j < 2; j++ {
53             for k := 0; k < 2; k++ {
54                 dp[i][j][k] = -1
55             }
56         }
57     }
58 }
59
60 len := 0
61 for num > 0 {
62     binary[len] = num & 1
63     num >>= 1
64     len++
65 }
66
67 return dfs(len-1, 1, 0, 0)
68 }
69
70 func main() {
71     reader := bufio.NewReader(os.Stdin)
72     line, _ := reader.ReadString('\n')
73     line = strings.TrimSpace(line)
74     parts := strings.Split(line, " ")
75     L, _ := strconv.Atoi(parts[0])
76     R, _ := strconv.Atoi(parts[1])
77     fmt.Println(digitSearch(R) - digitSearch(L-1))
78 }

```

来自: [华为OD机试2025B卷 – 不含101的数 \(C++&Python&JAVA&JS&GO\)–CSDN博客](#)

华为OD2025B卷 机试 - 最佳对手 (Java & Python & JS & C++ & GO)_实力差距最小总和-CSDN博客

最佳对手实力差距最小总和

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

2025B卷 200分题型

题目描述

游戏里面, 队伍通过匹配实力相近的对手进行对战。但是如果匹配的队伍实力相差太大, 对于双方游戏体验都不会太好。

给定 n 个队伍的实力值, 对其进行两两实力匹配, 两支队伍实例差距在允许的最大差距 d 内, 则可以匹配。要求在匹配队伍最多的情况下匹配出的各组实力差距的总和最小。

输入描述

第一行, n, d 。队伍个数 n 。允许的最大实力差距 d 。

- $2 \leq n \leq 50$
- $0 \leq d \leq 100$

第二行, n 个队伍的实力值空格分割。

- $0 \leq \text{各队伍实力值} \leq 100$

输出描述

匹配后, 各组对战的实力差值的总和。若没有队伍可以匹配, 则输出-1。

示例1

输入

```
1 6 30
2 81 87 47 59 81 18
```

输出

▼		Plain Text
1	57	

说明

18与47配对，实力差距29
59与81配对，实力差距22
81与87配对，实力差距6
总实力差距29+22+6=57

示例2

输入

▼		Plain Text
1	6 20	
2	81 87 47 59 81 18	

输出

▼		Plain Text
1	12	

说明

最多能匹配成功4支队伍。
47与59配对，实力差距12，
81与81配对，实力差距0。
总实力差距12+0=12

题解

思路：使用 **dp动态规划** 来处理队伍匹配数量问题。定义dp数组，**dp[i]**代表*i*个队伍最多能匹配的队伍数量，定义minSum数组，**minSum[i]** *i*个队伍匹配之后最小差值的和。具体处理过程如下：

- 先将队伍按照实力进行排序。
- 增加一个队伍的匹配队伍的数量变化只会跟 **i-1** 和 **i-2** 队伍状态相关联。首先判断 **ans[i] - ans[i-1] <= k** 是否成立
 - 成立，说明两个队伍可以进行匹配。这时候状态转移方程为
 - **dp[i] = max(dp[i-2] + 1, dp[i-1])**
 - **dp[i-2] + 1 > dp[i-1]**时,minSum[i] = minSum[i-2] + ans[i] - ans[i-2]

- $dp[i-2] + 1 < dp[i-1]$ 时, $minSum[i] = minSum[i-1]$;
- $dp[i-2] + 1 = dp[i-1]$ 时, $minSum[i] = \min(minSum[i-1], minSum[i-2] + ans[i] - ans[i-2])$
- 不成立时, $dp[i] = dp[i-1], minSum[i] = minSum[i-1]$;

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
10 // 通用 split 函数
11 vector<string> split(const string& str, const string& delimiter) {
12     vector<string> result;
13     size_t start = 0;
14     size_t end = str.find(delimiter);
15     while (end != string::npos) {
16         result.push_back(str.substr(start, end - start));
17         start = end + delimiter.length();
18         end = str.find(delimiter, start);
19     }
20     // 添加最后一个部分
21     result.push_back(str.substr(start));
22     return result;
23 }
24
25 int main() {
26     int n,d;
27     cin >> n >> d;
28     // 处理换行符
29     cin.ignore();
30
31     string s;
32     getline(cin, s);
33     vector<string> ansStr = split(s, " ");
34     vector<int> ans(n);
35     for (int i = 0; i < n; i++) {
36         ans[i] = stoi(ansStr[i]);
37     }
38
39     // 排序
40     sort(ans.begin(), ans.end());
41     // dp[i] 代表i个队伍能匹配的队伍数量
42     vector<int> dp(n+1, 0);
43     //minSum[i] i个队伍匹配之后差值的和
44     vector<int> minSum(n+1, 0);
45 }
```

```

46     for (int i = 2; i <= n; i++) {
47         bool canMatch = false;
48         // 能否匹配
49         if (ans[i-1] - ans[i-2] <= d) {
50             canMatch = true;
51         }
52         if (canMatch) {
53             if (dp[i-1] < dp[i-2] + 1) {
54                 dp[i] = dp[i-2] + 1;
55                 minSum[i] = minSum[i-2] + ans[i-1] - ans[i-2];
56             } else if (dp[i-1] > dp[i-2] + 1) {
57                 dp[i] = dp[i-1];
58                 minSum[i] = minSum[i-1];
59             } else {
60                 dp[i] = dp[i-1];
61                 minSum[i] = min(minSum[i-1], minSum[i-2] + ans[i-1] - ans
62 [i-2]);
63             }
64         } else {
65             dp[i] = dp[i-1];
66             minSum[i] = minSum[i-1];
67         }
68     }
69     if (dp[n] == 0) {
70         cout << -1;
71     } else {
72         cout << minSum[n];
73     }
74     return 0;
75 }

```

Java

```
1  import java.util.*;
2
3  public class Main {
4      // 通用 split 函数
5      public static List<String> split(String str, String delimiter) {
6          return Arrays.asList(str.split(delimiter));
7      }
8
9      public static void main(String[] args) {
10         Scanner scanner = new Scanner(System.in);
11         int n = scanner.nextInt(), d = scanner.nextInt();
12         scanner.nextLine(); // 处理换行符
13
14         String s = scanner.nextLine();
15         List<String> ansStr = split(s, " ");
16         int[] ans = new int[n];
17
18         for (int i = 0; i < n; i++) {
19             ans[i] = Integer.parseInt(ansStr.get(i));
20         }
21
22         // 排序
23         Arrays.sort(ans);
24
25         // dp[i] 代表 i 个队伍能匹配的队伍数量
26         int[] dp = new int[n + 1];
27         // minSum[i] i 个队伍匹配之后差值的和
28         int[] minSum = new int[n + 1];
29
30         for (int i = 2; i <= n; i++) {
31             boolean canMatch = ans[i - 1] - ans[i - 2] <= d;
32
33             if (canMatch) {
34                 // 优先选择匹配数量多的, 匹配数量相等的选择差异值和小
35                 if (dp[i - 1] < dp[i - 2] + 1) {
36                     dp[i] = dp[i - 2] + 1;
37                     minSum[i] = minSum[i - 2] + ans[i - 1] - ans[i - 2];
38                 } else if (dp[i - 1] > dp[i - 2] + 1) {
39                     dp[i] = dp[i - 1];
40                     minSum[i] = minSum[i - 1];
41                 } else {
42                     dp[i] = dp[i - 1];
43                     minSum[i] = Math.min(minSum[i - 1], minSum[i - 2] + an
44 s[i - 1] - ans[i - 2]);
45                 }
46             }
47         }
48     }
49 }
```

```
45         } else {  
46             dp[i] = dp[i - 1];  
47             minSum[i] = minSum[i - 1];  
48         }  
49     }  
50  
51     System.out.println(dp[n] == 0 ? -1 : minSum[n]);  
52 }  
53 }
```

Python


```
1  import sys
2
3  # 读取输入
4  n, d = map(int, sys.stdin.readline().split())
5  arr = list(map(int, sys.stdin.readline().split()))
6
7  # 排序
8  arr.sort()
9
10 # dp[i] 代表 i 个队伍能匹配的队伍数量
11 dp = [0] * (n + 1)
12 # minSum[i] i 个队伍匹配之后差值的和
13 minSum = [0] * (n + 1)
14
15 for i in range(2, n + 1):
16     canMatch = arr[i - 1] - arr[i - 2] <= d
17     # 优先选择匹配数量多的，匹配数量相等的选择差异值和小
18     if canMatch:
19         if dp[i - 1] < dp[i - 2] + 1:
20             dp[i] = dp[i - 2] + 1
21             minSum[i] = minSum[i - 2] + arr[i - 1] - arr[i - 2]
22         elif dp[i - 1] > dp[i - 2] + 1:
23             dp[i] = dp[i - 1]
24             minSum[i] = minSum[i - 1]
25         else:
26             dp[i] = dp[i - 1]
27             minSum[i] = min(minSum[i - 1], minSum[i - 2] + arr[i - 1] - ar
28 r[i - 2])
29         else:
30             dp[i] = dp[i - 1]
31             minSum[i] = minSum[i - 1]
32
33 print(-1 if dp[n] == 0 else minSum[n])
```

JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  const inputLines = [];
9
10 rl.on("line", (line) => {
11   inputLines.push(line);
12 }).on("close", () => {
13   const [n, d] = inputLines[0].split(" ").map(Number);
14   const arr = inputLines[1].split(" ").map(Number);
15
16   // 排序
17   arr.sort((a, b) => a - b);
18
19   // dp[i] 代表 i 个队伍能匹配的队伍数量
20   const dp = new Array(n + 1).fill(0);
21   // minSum[i] i 个队伍匹配之后差值的和
22   const minSum = new Array(n + 1).fill(0);
23
24   for (let i = 2; i <= n; i++) {
25     const canMatch = arr[i - 1] - arr[i - 2] <= d;
26     // 优先选择匹配数量多的, 匹配数量相等的选择差异值和小
27     if (canMatch) {
28       if (dp[i - 1] < dp[i - 2] + 1) {
29         dp[i] = dp[i - 2] + 1;
30         minSum[i] = minSum[i - 2] + (arr[i - 1] - arr[i - 2]);
31       } else if (dp[i - 1] > dp[i - 2] + 1) {
32         dp[i] = dp[i - 1];
33         minSum[i] = minSum[i - 1];
34       } else {
35         dp[i] = dp[i - 1];
36         minSum[i] = Math.min(minSum[i - 1], minSum[i - 2] + (arr
37 [i - 1] - arr[i - 2]));
38       }
39     } else {
40       dp[i] = dp[i - 1];
41       minSum[i] = minSum[i - 1];
42     }
43   }
44   console.log(dp[n] === 0 ? -1 : minSum[n]);
```

```
45  });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8     "strings"
9     "sort"
10 )
11
12 func main() {
13     scanner := bufio.NewScanner(os.Stdin)
14
15     // 读取 n 和 d
16     scanner.Scan()
17     nd := strings.Split(scanner.Text(), " ")
18     n, _ := strconv.Atoi(nd[0])
19     d, _ := strconv.Atoi(nd[1])
20
21     // 读取数组
22     scanner.Scan()
23     strArr := strings.Split(scanner.Text(), " ")
24     arr := make([]int, n)
25     for i := 0; i < n; i++ {
26         arr[i], _ = strconv.Atoi(strArr[i])
27     }
28
29     // 排序
30     sort.Ints(arr)
31
32     // dp[i] 代表 i 个队伍能匹配的队伍数量
33     dp := make([]int, n+1)
34     // minSum[i] i 个队伍匹配之后差值的和
35     minSum := make([]int, n+1)
36
37     for i := 2; i <= n; i++ {
38         canMatch := arr[i-1] - arr[i-2] <= d
39         // 优先选择匹配数量多的, 匹配数量相等的选择差异值和小
40         if canMatch {
41             if dp[i-1] < dp[i-2]+1 {
42                 dp[i] = dp[i-2] + 1
43                 minSum[i] = minSum[i-2] + (arr[i-1] - arr[i-2])
44             } else if dp[i-1] > dp[i-2]+1 {
45                 dp[i] = dp[i-1]
```

```

46         minSum[i] = minSum[i-1]
47     } else {
48         dp[i] = dp[i-1]
49         minSum[i] = min(minSum[i-1], minSum[i-2]+(arr[i-1]-arr[i-2]))
50     }
51 } else {
52     dp[i] = dp[i-1]
53     minSum[i] = minSum[i-1]
54 }
55 }
56
57 if dp[n] == 0 {
58     fmt.Println(-1)
59 } else {
60     fmt.Println(minSum[n])
61 }
62 }

```

来自: [华为OD2025B卷 机试 - 最佳对手 \(Java & Python & JS & C++ &GO \)_实力差距最小总和-CSDN博客](#)

华为OD上机考试2025B卷 - 最佳的出牌方法

(C++ & Python & JAVA & JS & GO)_华为od机考

2025b卷-CSDN博客

最佳的出牌方法

华为OD机试真题目录: [点击查看](#)

2025 B卷 200分题型

题目描述

手上有一副**扑克牌**，每张牌按牌面数字记分（J=11,Q=12,K=13，没有大小王），出牌时按照以下规则记分：

- 出单张，记牌面分数，例如出一张2，得分为2
- 出对或3张，记牌面分数总和再 $\times 2$ ，例如出3张3，得分为 $(3+3+3)\times 2=18$
- 出5张顺，记牌面分数总和再 $\times 2$ ，例如出34567顺，得分为 $(3+4+5+6+7)\times 2=50$
- 出4张炸弹，记牌面分数总和再 $\times 3$ ，例如出4张4，得分为 $4\times 4\times 3=48$

求出一副牌最高的得分数

输入描述

按顺序排好的一副牌，最少1张，最多15张。

1-9输入为数字1-9，10输入为数字0，JQK输入为大写字母JQK。

无需考虑输入非法的情况，例如输入字符不在[0-9JQK]范围或某一张牌超过4张

备注

积分规则中没有的出牌方式不支持，例如不支持3带1、4带2，不支持5张以上的顺，且10JQKA (0JQK1)不算顺。

输出描述

最高的得分数

用例1

输入

▼	Plain Text
1	33445677

输出

▼	Plain Text
1	67

说明

▼	Plain Text
1	出对3、对4、对7，单张5、6，得分为67；
2	出34567顺，再出单张3、4、7，得分为64
3	因此最高得分是按对出，可得到最高分67，输出结果67

题解

思路：本题牌的个数只有[1-15]，[数据比较小](#)，可以直接使用 `递归回溯` 方法求解，枚举所有可能的出牌情况，记录获取到的最大值。具体逻辑参照下面代码

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 res = 0;
10
11 // cardCount 当前每种牌的数量 count 剩余牌数量 index 当前选牌位置 score 当前获得分
   数
12 void dfs(vector<int> cardCount, int count, int index , int score) {
13     if (count == 0) {
14         res = max(res, score);
15         return;
16     }
17
18     if (cardCount[index] == 0) {
19         dfs(cardCount, count, index + 1, score);
20         return;
21     }
22
23     // 顺子策略
24     if (index <= 9 && cardCount[index + 1] >= 1 && cardCount[index + 2] >
   = 1 && cardCount[index + 3] >= 1 && cardCount[index + 4] >= 1) {
25         cardCount[index]--;
26         cardCount[index+1]--;
27         cardCount[index+2]--;
28         cardCount[index+3]--;
29         cardCount[index+4]--;
30
31         dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2);
32         // 回溯
33         cardCount[index]++;
34         cardCount[index+1]++;
35         cardCount[index+2]++;
36         cardCount[index+3]++;
37         cardCount[index+4]++;
38     }
39     // 对子
40     if (cardCount[index] >= 2) {
41         cardCount[index] -=2;
42         dfs(cardCount, count - 2, index, score + index * 2 * 2);
43         cardCount[index] +=2;
```



```

44     }
45
46     // 三张
47     if (cardCount[index] >= 3) {
48         cardCount[index] -=3;
49         dfs(cardCount, count - 3, index, score + index * 3 * 2);
50         cardCount[index] +=3;
51     }
52     // 四张
53     if (cardCount[index] >= 4) {
54         cardCount[index] -=4;
55         dfs(cardCount, count - 4, index, score + index * 3 * 4);
56         cardCount[index] +=4;
57     }
58
59     // 单张
60     cardCount[index]--;
61     dfs(cardCount, count - 1, index, score + index);
62     cardCount[index]++;
63 }
64
65 int main() {
66     string s;
67     cin >> s;
68     // 记录牌的数量
69     vector<int> cardCount(14, 0);
70     for (int i = 0; i < s.size(); i++) {
71         char c = s[i];
72         if (c == '0') {
73             cardCount[10]++;
74         }else if (c == 'J') {
75             cardCount[11]++;
76         }else if (c == 'Q') {
77             cardCount[12]++;
78         }else if (c == 'K') {
79             cardCount[13]++;
80         }else {
81             cardCount[c - '0']++;
82         }
83     }
84
85     dfs(cardCount, s.size(), 1, 0);
86
87     cout << res;
88     return 0;
89 }

```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      static int res = 0;
5
6      // cardCount 当前每种牌的数量 count 剩余牌数量 index 当前选牌位置 score 当前
   获得分数
7      public static void dfs(int[] cardCount, int count, int index, int score) {
8          if (count == 0) {
9              res = Math.max(res, score);
10             return;
11         }
12
13         if (cardCount[index] == 0) {
14             dfs(cardCount, count, index + 1, score);
15             return;
16         }
17
18         // 顺子策略
19         if (index <= 9 && cardCount[index + 1] >= 1 && cardCount[index +
20 2] >= 1 && cardCount[index + 3] >= 1 && cardCount[index + 4] >= 1) {
21             cardCount[index]--;
22             cardCount[index + 1]--;
23             cardCount[index + 2]--;
24             cardCount[index + 3]--;
25             cardCount[index + 4]--;
26
27             dfs(cardCount, count - 5, index, score + (index * 5 + 10) *
28 2);
29
30             // 回溯
31             cardCount[index]++;
32             cardCount[index + 1]++;
33             cardCount[index + 2]++;
34             cardCount[index + 3]++;
35             cardCount[index + 4]++;
36         }
37
38         // 对子
39         if (cardCount[index] >= 2) {
40             cardCount[index] -= 2;
41             dfs(cardCount, count - 2, index, score + index * 2 * 2);
42             cardCount[index] += 2;
43         }
44     }
```

```

42         // 三张
43         if (cardCount[index] >= 3) {
44             cardCount[index] -= 3;
45             dfs(cardCount, count - 3, index, score + index * 3 * 2);
46             cardCount[index] += 3;
47         }
48
49         // 四张
50         if (cardCount[index] >= 4) {
51             cardCount[index] -= 4;
52             dfs(cardCount, count - 4, index, score + index * 3 * 4);
53             cardCount[index] += 4;
54         }
55
56         // 单张
57         cardCount[index]--;
58         dfs(cardCount, count - 1, index, score + index);
59         cardCount[index]++;
60     }
61
62     public static void main(String[] args) {
63         Scanner scanner = new Scanner(System.in);
64         String s = scanner.nextLine();
65         // 记录牌的数量
66         int[] cardCount = new int[14];
67         for (int i = 0; i < s.length(); i++) {
68             char c = s.charAt(i);
69             if (c == '0') {
70                 cardCount[10]++;
71             } else if (c == 'J') {
72                 cardCount[11]++;
73             } else if (c == 'Q') {
74                 cardCount[12]++;
75             } else if (c == 'K') {
76                 cardCount[13]++;
77             } else {
78                 cardCount[c - '0']++;
79             }
80         }
81
82         dfs(cardCount, s.length(), 1, 0);
83
84         System.out.println(res);
85     }
86 }

```

```
1  # 全局变量记录最大得分
2  res = 0
3
4  # cardCount 当前每种牌的数量 count 剩余牌数量 index 当前选牌位置 score 当前获得分
   数
5  def dfs(cardCount, count, index, score):
6      global res
7      if count == 0:
8          res = max(res, score)
9          return
10
11     if cardCount[index] == 0:
12         dfs(cardCount, count, index + 1, score)
13         return
14
15     # 顺子策略
16     if index <= 9 and cardCount[index + 1] >= 1 and cardCount[index + 2] >
= 1 and cardCount[index + 3] >= 1 and cardCount[index + 4] >= 1:
17         cardCount[index] -= 1
18         cardCount[index + 1] -= 1
19         cardCount[index + 2] -= 1
20         cardCount[index + 3] -= 1
21         cardCount[index + 4] -= 1
22
23         dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2)
24     # 回溯
25     cardCount[index] += 1
26     cardCount[index + 1] += 1
27     cardCount[index + 2] += 1
28     cardCount[index + 3] += 1
29     cardCount[index + 4] += 1
30
31     # 对子
32     if cardCount[index] >= 2:
33         cardCount[index] -= 2
34         dfs(cardCount, count - 2, index, score + index * 2 * 2)
35         cardCount[index] += 2
36
37     # 三张
38     if cardCount[index] >= 3:
39         cardCount[index] -= 3
40         dfs(cardCount, count - 3, index, score + index * 3 * 2)
41         cardCount[index] += 3
42
43     # 四张
```

```

44     if cardCount[index] >= 4:
45         cardCount[index] -= 4
46         dfs(cardCount, count - 4, index, score + index * 3 * 4)
47         cardCount[index] += 4
48
49     # 单张
50     cardCount[index] -= 1
51     dfs(cardCount, count - 1, index, score + index)
52     cardCount[index] += 1
53
54 def main():
55     s = input().strip() # 读取输入字符串
56     # 记录牌的数量
57     cardCount = [0] * 14
58     for c in s:
59         if c == '0':
60             cardCount[10] += 1
61         elif c == 'J':
62             cardCount[11] += 1
63         elif c == 'Q':
64             cardCount[12] += 1
65         elif c == 'K':
66             cardCount[13] += 1
67         else:
68             cardCount[int(c)] += 1
69
70     dfs(cardCount, len(s), 1, 0)
71
72     print(res)
73
74 if __name__ == "__main__":
75     main()

```

JavaScript

```
1  let res = 0;
2
3  // cardCount 当前每种牌的数量 count 剩余牌数量 index 当前选牌位置 score 当前获得分
   数
4  function dfs(cardCount, count, index, score) {
5      if (count === 0) {
6          res = Math.max(res, score);
7          return;
8      }
9
10     if (cardCount[index] === 0) {
11         dfs(cardCount, count, index + 1, score);
12         return;
13     }
14
15     // 顺子策略
16     if (index <= 9 && cardCount[index + 1] >= 1 && cardCount[index + 2] >
   = 1 && cardCount[index + 3] >= 1 && cardCount[index + 4] >= 1) {
17         cardCount[index]--;
18         cardCount[index + 1]--;
19         cardCount[index + 2]--;
20         cardCount[index + 3]--;
21         cardCount[index + 4]--;
22
23         dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2);
24         // 回溯
25         cardCount[index]++;
26         cardCount[index + 1]++;
27         cardCount[index + 2]++;
28         cardCount[index + 3]++;
29         cardCount[index + 4]++;
30     }
31
32     // 对子
33     if (cardCount[index] >= 2) {
34         cardCount[index] -= 2;
35         dfs(cardCount, count - 2, index, score + index * 2 * 2);
36         cardCount[index] += 2;
37     }
38
39     // 三张
40     if (cardCount[index] >= 3) {
41         cardCount[index] -= 3;
42         dfs(cardCount, count - 3, index, score + index * 3 * 2);
43         cardCount[index] += 3;
```

```

44     }
45
46     // 四张
47     if (cardCount[index] >= 4) {
48         cardCount[index] -= 4;
49         dfs(cardCount, count - 4, index, score + index * 3 * 4);
50         cardCount[index] += 4;
51     }
52
53     // 单张
54     cardCount[index]--;
55     dfs(cardCount, count - 1, index, score + index);
56     cardCount[index]++;
57 }
58
59 // 主函数，负责输入输出和调用递归
60 function main() {
61     // 从控制台读取输入
62     const readline = require('readline');
63     const rl = readline.createInterface({
64         input: process.stdin,
65         output: process.stdout
66     });
67
68     rl.question('', (s) => {
69         const cardCount = new Array(14).fill(0);
70
71         // 记录牌的数量
72         for (let i = 0; i < s.length; i++) {
73             const c = s[i];
74             if (c === '0') {
75                 cardCount[10]++;
76             } else if (c === 'J') {
77                 cardCount[11]++;
78             } else if (c === 'Q') {
79                 cardCount[12]++;
80             } else if (c === 'K') {
81                 cardCount[13]++;
82             } else {
83                 cardCount[parseInt(c)]++;
84             }
85         }
86
87         dfs(cardCount, s.length, 1, 0);
88
89         // 输出最大得分
90         console.log(res);
91         rl.close();

```



```
92     });  
93 }  
94  
95 main();
```

Go

```
1  package main
2
3  import (
4      "bufio"
5      "fmt"
6      "os"
7  )
8
9  var res int
10
11 // dfs 深度优先搜索 (回溯)
12 func dfs(cardCount []int, count, index, score int) {
13     if count == 0 {
14         if score > res {
15             res = score
16         }
17         return
18     }
19
20     if cardCount[index] == 0 {
21         dfs(cardCount, count, index+1, score)
22         return
23     }
24
25     // 顺子策略
26     if index <= 9 && cardCount[index+1] >= 1 && cardCount[index+2] >= 1 && c
ardCount[index+3] >= 1 && cardCount[index+4] >= 1 {
27         cardCount[index]--
28         cardCount[index+1]--
29         cardCount[index+2]--
30         cardCount[index+3]--
31         cardCount[index+4]--
32
33         dfs(cardCount, count-5, index, score+(index*5+10)*2)
34         // 回溯
35         cardCount[index]++
36         cardCount[index+1]++
37         cardCount[index+2]++
38         cardCount[index+3]++
39         cardCount[index+4]++
40     }
41
42     // 对子
43     if cardCount[index] >= 2 {
44         cardCount[index] -= 2
```

```

45     dfs(cardCount, count-2, index, score+index*2*2)
46     cardCount[index] += 2
47 }
48
49 // 三张
50 if cardCount[index] >= 3 {
51     cardCount[index] -= 3
52     dfs(cardCount, count-3, index, score+index*3*2)
53     cardCount[index] += 3
54 }
55
56 // 四张
57 if cardCount[index] >= 4 {
58     cardCount[index] -= 4
59     dfs(cardCount, count-4, index, score+index*3*4)
60     cardCount[index] += 4
61 }
62
63 // 单张
64 cardCount[index]--
65 dfs(cardCount, count-1, index, score+index)
66 cardCount[index]++
67 }
68
69 func main() {
70     // 读取输入
71     reader := bufio.NewReader(os.Stdin)
72     s, _ := reader.ReadString('\n')
73     s = s[:len(s)-1] // 去除换行符
74
75     // 初始化牌的数量
76     cardCount := make([]int, 14)
77
78     // 统计每张牌的数量
79     for _, c := range s {
80         switch c {
81             case '0':
82                 cardCount[10]++
83             case 'J':
84                 cardCount[11]++
85             case 'Q':
86                 cardCount[12]++
87             case 'K':
88                 cardCount[13]++
89             default:
90                 cardCount[int(c-'0')]++
91         }
92     }

```

```
93
94 // 调用深度优先搜索
95 dfs(cardCount, len(s), 1, 0)
96
97 // 输出最大得分
98 fmt.Println(res)
99 }
```

来自: [华为OD上机考试2025B卷 – 最佳的出牌方法 \(C++ & Python & JAVA & JS & GO\)_华为od机考2025b卷-CSDN博客](#)

华为OD 2025 B卷 - 单词倒序 (C++ & Python & JAVA & JS & GO)-CSDN博客

单词倒序

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

华为OD机试2025B卷 100分题型

题目描述

输入单行英文句子，里面包含英文字母，空格以及,.?三种标点符号，请将句子内每个单词进行倒序，并输出倒序后的语句。

输入描述

输入字符串S，S的长度 $1 \leq N \leq 100$

备注

标点符号左右的空格 ≥ 0 ，单词间空格 > 0

输出描述

输出倒序后的字符串

用例1

输入

▼

Plain Text

1 yM eman si boB.

输出

▼

Plain Text

1 My name is Bob.

用例2

输入

▼ Plain Text |

```
1   woh era uoy ? I ma enif.
```

输出

▼ Plain Text |

```
1   how are you ? I am fine.
```

题解

思路： 模拟题 ，使用 `currentStr` 保存当前遍历的单词。

- 当遇到非字母字符(标点符号)时，将 `currentStr` 反转打印输出，并打印当前遍历字符，`currentStr` 置空。
- 否则将字符到拼接 `currentStr` 后。

额外注意遍历完原输入字符串之后，需要进行收尾操作。

C++

```
1  #include <cctype>
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 using namespace std;
11
12 int main() {
13     string s;
14     getline(cin, s);
15     string currentStr = "";
16     for (int i = 0; i < s.size(); i++) {
17         if (isalpha(s[i])) {
18             currentStr += s[i];
19         } else {
20             // 单词反转
21             if (!s.empty()) {
22                 reverse(currentStr.begin(), currentStr.end());
23                 cout << currentStr;
24             }
25             currentStr = "";
26             // 非字母字符直接输出
27             cout << s[i];
28         }
29     }
30     // 收尾操作
31     if (!currentStr.empty()) {
32         reverse(currentStr.begin(), currentStr.end());
33         cout << currentStr;
34     }
35     return 0;
36 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          String s = scanner.nextLine();
7          scanner.close();
8
9          StringBuilder currentStr = new StringBuilder();
10         StringBuilder result = new StringBuilder();
11
12         for (char c : s.toCharArray()) {
13             if (Character.isLetter(c)) {
14                 currentStr.append(c);
15             } else {
16                 // 反转单词并追加
17                 if (currentStr.length() > 0) {
18                     result.append(currentStr.reverse());
19                     currentStr.setLength(0); // 清空 currentStr
20                 }
21                 // 直接追加标点或空格
22                 result.append(c);
23             }
24         }
25         // 处理最后一个单词 收尾操作
26         if (currentStr.length() > 0) {
27             result.append(currentStr.reverse());
28         }
29         System.out.println(result);
30     }
31 }
```

Python


```
1  import sys
2
3  def reverse_words(sentence):
4      result = []
5      current_str = []
6
7      for char in sentence:
8          # 拼接
9          if char.isalpha():
10             current_str.append(char)
11         else:
12             # 反转
13             if current_str:
14                 result.append("".join(reversed(current_str)))
15                 current_str = []
16             result.append(char)
17         # 收尾操作
18         if current_str:
19             result.append("".join(reversed(current_str)))
20
21     print("".join(result))
22
23 if __name__ == "__main__":
24     sentence = sys.stdin.readline().strip()
25     reverse_words(sentence)
```

JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  rl.on("line", (s) => {
9    let currentStr = "";
10   let result = "";
11
12   for (let i = 0; i < s.length; i++) {
13     if (/[a-zA-Z]/.test(s[i])) {
14       currentStr += s[i];
15     } else {
16       if (currentStr.length > 0) {
17         result += currentStr.split("").reverse().join(""); // 反转
            单词
18         currentStr = "";
19       }
20       result += s[i]; // 追加标点
21     }
22   }
23   // 收尾操作
24   if (currentStr.length > 0) {
25     result += currentStr.split("").reverse().join(""); // 处理最后一个单
            词
26   }
27
28   console.log(result);
29   rl.close();
30 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "unicode"
8 )
9 // 反转
10 func reverseWord(word []rune) {
11     for i, j := 0, len(word)-1; i < j; i, j = i+1, j-1 {
12         word[i], word[j] = word[j], word[i]
13     }
14 }
15
16 func main() {
17     reader := bufio.NewReader(os.Stdin)
18     s, _ := reader.ReadString('\n')
19     s = s[:len(s)-1] // 移除换行符
20
21     var result []rune
22     var currentStr []rune
23
24     for _, c := range s {
25         // 直接拼接
26         if unicode.IsLetter(c) {
27             currentStr = append(currentStr, c)
28         } else {
29             // 反转
30             if len(currentStr) > 0 {
31                 reverseWord(currentStr)
32                 result = append(result, currentStr...)
33                 currentStr = nil
34             }
35             result = append(result, c)
36         }
37     }
38     // 收尾操作
39     if len(currentStr) > 0 {
40         reverseWord(currentStr)
41         result = append(result, currentStr...)
42     }
43
44     fmt.Println(string(result))
45 }
```

来自: [华为OD 2025 B卷 – 单词倒序 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

华为OD机考 2025 B卷 - 最短木板长度 (C++ & Python & JAVA & JS & GO)_华为od机考b卷-CSDN博客

最短木板长度

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

华为OD机试2025B卷 100分题型

题目描述

小明有 n 块木板，第 i ($1 \leq i \leq n$) 块木板长度为 a_i 。
小明买了一块长度为 m 的木料，这块木料可以切割成任意块，拼接到已有的木板上，用来加长木板。
小明想让最短的模板尽量长。请问小明加长木板后，最短木板的长度可以为多少？

输入描述

输入的第一行包含两个正整数， n ($1 \leq n \leq 10^3$), m ($1 \leq m \leq 10^6$)， n 表示木板数， m 表示木板长度。
输入的第二行包含 n 个正整数， a_1, a_2, \dots, a_n ($1 \leq a_i \leq 10^6$)。

输出描述

输出的唯一一行包含一个正整数，表示加长木板后，最短木板的长度最大可以为多少？

用例1

输入

	Plain Text
1	5 3
2	4 5 3 5 5

输出

	Plain Text
1	5

说明

给第1块木板长度增加1，给第3块木板长度增加2后，这5块木板长度变为[5,5,5,5,5]，最短的木板的长度最大为5。

用例2

输入

▼	Plain Text
1	5 2
2	4 5 3 5 5

输出

▼	Plain Text
1	4

说明

给第3块木板长度增加1后，这5块木板长度变为[4,5,4,5,5]，剩余木料的长度为1。此时剩余木料无论给哪块木板加长，最短木料的长度都为4。

题解

要想让最短的木板尽可能长，那么我们就需要不停地递进式补足最短板。
比如用例输入有5个板：4 5 3 5 5，可用材料m=3,最短的板长度是3，只有一个，那么我们就将他补足到4，此时消耗了一单位长度的材料，m=2,这样的话，只剩下两种长度的板4，5，且4长度有两个，5长度有三个，最短板是长度4.接下来我们应该尽量将最短板4长度的板补足到5长度，而刚好剩余材料m=2，可以将所有4长度的板补足到5长度，此时所有板都是5长度，且材料耗尽。

题解

思路：贪心，要想最短的木板长度更长，肯定是优先让最短木板边长。

1. 使用 哈希表 统计各个长度的木板数量。
2. 使用 优先队列 存储{木板长度，数量}格式，优先队列排序规则 长度短的位于栈顶 。
3. 接下来处理逻辑如下：
 - 根据优先队列长度不同：
 - 长度为1的情况：假设当前队列中唯一元素为{len, x}，那么最终最短木板的最长长度为 $len + (m / x)$ ， m / x 会向下取整。可以直接得出结果。
 - 长度不为1的情况：这时候的处理办法，肯定是让最短木板长度变更为 次最短长度。假设最短

的结构为 $\{\text{len1}, x1\}$,次短的结构为 $\{\text{len2}, x2\}$.此时根据m的值进行处理

- $(\text{len2} - \text{len1}) * x1 \leq m$ 时: 说明最短都可以变成次短长度, 此时进行的操作为 $m -= (\text{len2} - \text{len1}) * x1$, 次最短数量变为 $x1 + x2$.再进行一次循环处理。
- $(\text{len2} - \text{len1}) * x1 > m$: 不是最短都可以变成次短长度, 此时可以直接得出结果为 $\text{len1} + (m / x1)$, 会向下取整

4. 重复3的逻辑, 推出循环判断之后, 最终栈顶元素长度就是结果。

C++

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<map>
7  #include<algorithm>
8  using namespace std;
9
10 int main() {
11     int n,m;
12     cin >> n >>m;
13     vector<int> ans(n);
14     for (int i = 0; i < n; i++) {
15         cin >> ans[i];
16     }
17
18     sort(ans.begin(), ans.end());
19     // 贪心算法
20     while (m--) {
21         int pos = 1;
22         for (; pos < n; pos++) {
23             if (ans[pos] > ans[pos-1]) {
24                 break;
25             }
26         }
27         ans[pos-1]++;
28     }
29
30     cout << ans[0];
31     return 0;
32 }
```



```
1  // 优化实现
2  #include<iostream>
3  #include<vector>
4  #include<string>
5  #include <utility>
6  #include <sstream>
7  #include<map>
8  #include<algorithm>
9  #include<queue>
10 using namespace std;
11
12
13 struct Compare {
14     bool operator()(const pair<int, int>& a, const pair<int, int>& b) {
15         return a.first > b.first; // first 值越大优先级越低
16     }
17 };
18
19
20 int main() {
21     int n,m;
22     cin >> n >>m;
23     vector<int> ans(n);
24     map<int,int> mp;
25     for (int i = 0; i < n; i++) {
26         cin >> ans[i];
27         mp[ans[i]]++;
28     }
29
30     // 小顶堆
31     priority_queue<pair<int, int>, vector<pair<int, int>>, Compare> pq;
32     for (auto p : mp) {
33         pq.push({p.first, p.second});
34     }
35     while (m != 0) {
36         pair<int,int> top = pq.top();
37         // 都是同样长度, 平均分
38         if (pq.size() == 1) {
39             int count = top.second;
40             int num = top.first;
41             // 会自定向下取整的
42             int diff = m / count;
43             pq.pop();
44             pq.push({num+diff, count});
45             break;
```

```

46         } else {
47             pq.pop();
48             pair<int,int> se = pq.top();
49             int diff = se.first - top.first;
50             int count = top.second;
51
52             if (diff * count <= m) {
53                 pq.pop();
54                 // 将值提升至se.fisrt消耗的长度
55                 pq.push({se.first, count + se.second});
56                 m -= diff * count;
57             } else {
58                 // 这是数量写多少都不影响答案
59                 pq.push({top.first + (m / count), 1});
60                 break;
61             }
62         }
63     }
64     cout << pq.top().first;
65 }

```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 自定义比较器
5      static class Compare implements Comparator<Map.Entry<Integer, Integer>
6  > {
7          public int compare(Map.Entry<Integer, Integer> a, Map.Entry<Integer, Integer> b) {
8              return Integer.compare(a.getKey(), b.getKey()); // 按照数字升序
9              排序
10         }
11     }
12
13     public static void main(String[] args) {
14         Scanner scanner = new Scanner(System.in);
15         int n = scanner.nextInt();
16         int m = scanner.nextInt();
17         int[] ans = new int[n];
18         Map<Integer, Integer> mp = new HashMap<>();
19
20         // 输入数据并统计
21         for (int i = 0; i < n; i++) {
22             ans[i] = scanner.nextInt();
23             mp.put(ans[i], mp.getOrDefault(ans[i], 0) + 1);
24         }
25
26         // 小顶堆
27         PriorityQueue<Map.Entry<Integer, Integer>> pq = new PriorityQueue<
28 >(new Compare());
29         pq.addAll(mp.entrySet());
30
31         while (m != 0) {
32             Map.Entry<Integer, Integer> top = pq.poll();
33             // 都是同样长度, 平均分
34             if (pq.size() == 0) {
35                 int count = top.getValue();
36                 int num = top.getKey();
37                 int diff = m / count;
38                 pq.offer(new AbstractMap.SimpleEntry<>(num + diff, count));
39             } else {
40                 break;
41             }
42         }
43     }
44 }
```

```

41
42         if (diff * count <= m) {
43             pq.poll();
44             pq.offer(new AbstractMap.SimpleEntry<>(se.getKey(), co
45 unt + se.getValue()));
46             m -= diff * count;
47         } else {
48             pq.offer(new AbstractMap.SimpleEntry<>(top.getKey() +
49 (m / count), 1));
50             break;
51         }
52     }
53     System.out.println(pq.peek().getKey());
54 }
55 }

```

Python

```
1  import heapq
2  from collections import Counter
3
4  def main():
5      n, m = map(int, input().split())
6      ans = list(map(int, input().split()))
7      mp = Counter(ans)
8
9      # 小顶堆
10     pq = []
11     for num, count in mp.items():
12         heapq.heappush(pq, (num, count))
13
14     while m != 0:
15         top = heapq.heappop(pq)
16         # 都是同样长度, 平均分
17         if len(pq) == 0:
18             count = top[1]
19             num = top[0]
20             diff = m // count
21             heapq.heappush(pq, (num + diff, count))
22             break
23         else:
24             se = pq[0]
25             diff = se[0] - top[0]
26             count = top[1]
27
28             if diff * count <= m:
29                 heapq.heappop(pq)
30                 heapq.heappush(pq, (se[0], count + se[1]))
31                 m -= diff * count
32             else:
33                 heapq.heappush(pq, (top[0] + (m // count), 1))
34                 break
35
36     print(pq[0][0])
37
38 if __name__ == '__main__':
39     main()
```

JavaScript

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require("readline");
3
4  const rl = readline.createInterface({
5    input: process.stdin,
6    output: process.stdout,
7  });
8
9  const lines = [];
10 rl.on("line", (line) => {
11   lines.push(line);
12
13   if (lines.length === 2) {
14     const [n, m] = lines[0].split(" ").map(Number);
15     const plankLengths = lines[1].split(" ").map(Number);
16     console.log(findMaxEqualLength(m, plankLengths));
17
18     lines.length = 0;
19   }
20 });
21
22 function findMaxEqualLength(m, planks) {
23   // 统计每种长度的木板数量
24   const plankCount = new Map();
25   for (let length of planks) {
26     plankCount.set(length, (plankCount.get(length) || 0) + 1);
27   }
28
29   // 按照木板长度升序排序
30   const sortedPlanks = Array.from(plankCount.entries()).sort((a, b) => a[0] - b[0]);
31
32   while (m > 0) {
33     if (sortedPlanks.length === 1) {
34       // 只有一种木板长度，均分 m
35       return sortedPlanks[0][0] + Math.floor(m / sortedPlanks[0][1]);
36     }
37
38     let [minLength, minCount] = sortedPlanks.shift();
39     let [nextLength, nextCount] = sortedPlanks[0];
40
41     let requiredWood = (nextLength - minLength) * minCount;
42
43     if (requiredWood > m) {
44       return minLength + Math.floor(m / minCount);
45     }
46   }
47 }
```

```
45     } else {  
46         m -= requiredWood;  
47         sortedPlanks[0][1] += minCount; // 合并计数  
48     }  
49 }  
50  
51 return sortedPlanks[0][0];  
52 }
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "sort"
6 )
7
8 func getResult(m int, a []int) int {
9     // 统计木板长度及其数量
10    boardCount := make(map[int]int)
11    for _, length := range a {
12        boardCount[length]++
13    }
14
15    // 转换为切片，并按长度排序
16    type board struct {
17        length int
18        count  int
19    }
20
21    boards := make([]board, 0, len(boardCount))
22    for length, count := range boardCount {
23        boards = append(boards, board{length, count})
24    }
25    sort.Slice(boards, func(i, j int) bool {
26        return boards[i].length < boards[j].length
27    })
28
29    // 逐步填充最短板
30    for i := 0; i < len(boards)-1; i++ {
31        cur := boards[i]
32        next := boards[i+1]
33
34        // 计算当前木板与下一个木板的长度差距
35        diff := next.length - cur.length
36        totalCost := diff * cur.count
37
38        if totalCost > m {
39            return cur.length + m/cur.count
40        }
41
42        m -= totalCost
43        boards[i+1].count += cur.count
44    }
45}
```



```
46 // 只剩下一块木板时，平分剩余 m
47 return boards[len(boards)-1].length + m/boards[len(boards)-1].count
48 }
49
50 func main() {
51     var n, m int
52     fmt.Scan(&n, &m)
53
54     a := make([]int, n)
55     for i := range a {
56         fmt.Scan(&a[i])
57     }
58
59     fmt.Println(getResult(m, a))
60 }
```

来自: [华为OD机考 2025 B卷 – 最短木板长度 \(C++ & Python & JAVA & JS & GO\)_华为od机考b卷-CSDN博客](#)

华为OD机试 2025 B卷 - 游戏分组 / 王者荣耀

(C++ & Python & JAVA & JS & GO)-CSDN博客

游戏分组 / 王者荣耀

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

华为OD机试2025B卷 100分题型

题目描述

部门准备举办一场王者荣耀表演赛，有10名游戏爱好者参与，分5为两队，每队5人。每位参与者都有一个评分，代表着他的游戏水平。为了表演赛尽可能精彩，我们需要把10名参赛者分为实力尽量相近的两队。一队的实力可以表示为这一队5名队员的评分总和。

现在给你10名参与者的游戏水平评分，请你根据上述要求分队最后输出这两组的实力差绝对值。

例: 10名参赛者的评分分别为5 1 8 3 4 6 7 10 9 2，分组为 (135 8 10) (24 679)，两组实力差最小，差值为1。有多种分法，但实力差的绝对值最小为1。

输入描述

10个整数，表示10名参与者的游戏水平评分。范围在[1,10000]之间

输出描述

1个整数，表示分组后两组实力差绝对值的[最小值](#)。

示例1

输入

▼	Plain Text
1 1 2 3 4 5 6 7 8 9 10	

输出

▼	Plain Text
1 1	

10名队员分成两组，两组实力差绝对值最小为1。

题解一

思路：由于题目数据量非常小，采用 深度优先遍历 求出所有55分组差异的最小值。

1. 基本逻辑就是使用 DFS 枚举出从10个人选出5个人的所有方案。
2. 计算出不同方案下分组总和的差异，最小值就为结果。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <cmath>
5  #include <climits>
6
7  using namespace std;
8
9  void findMinDifference(vector<int>& nums, int idx, int count, int sum1, int totalSum, int& minDiff) {
10     if (count == 5) { // 当选出5个成员时
11         int sum2 = totalSum - sum1;
12         // 记录最小值
13         minDiff = min(minDiff, abs(sum1 - sum2));
14         return;
15     }
16
17     if (idx == 10 || count + (10 - idx) < 5) return; // 剩余人数不足时剪枝
18
19     // 选
20     findMinDifference(nums, idx + 1, count + 1, sum1 + nums[idx], totalSum, minDiff);
21     // 不选
22     findMinDifference(nums, idx + 1, count, sum1, totalSum, minDiff);
23 }
24
25 int main() {
26     vector<int> nums(10);
27     int totalSum = 0;
28     for (int& num : nums) {
29         cin >> num;
30         totalSum += num;
31     }
32
33     int minDiff = INT_MAX;
34     findMinDifference(nums, 0, 0, 0, totalSum, minDiff);
35     cout << minDiff << endl;
36     return 0;
37 }
```

Java

```
1  import java.util.*;
2
3  public class Main {
4      static int minDiff = Integer.MAX_VALUE;
5      static int totalSum = 0;
6
7      // 递归查找最小差值
8      public static void findMinDifference(int[] nums, int idx, int count, int sum1) {
9          if (count == 5) { // 当选出5个成员时
10              int sum2 = totalSum - sum1;
11              minDiff = Math.min(minDiff, Math.abs(sum1 - sum2));
12              return;
13          }
14
15          if (idx == 10 || count + (10 - idx) < 5) return; // 剩余人数不足时剪
枝
16          //选择当前元素
17          findMinDifference(nums, idx + 1, count + 1, sum1 + nums[idx]);
18          // 不选择当前元素
19          findMinDifference(nums, idx + 1, count, sum1);
20      }
21
22      public static void main(String[] args) {
23          Scanner scanner = new Scanner(System.in);
24          int[] nums = new int[10];
25          totalSum = 0;
26
27          // 读取10个整数
28          for (int i = 0; i < 10; i++) {
29              nums[i] = scanner.nextInt();
30              totalSum += nums[i];
31          }
32          scanner.close();
33
34          findMinDifference(nums, 0, 0, 0);
35          System.out.println(minDiff);
36      }
37  }
```

Python

```
1  import sys
2
3  def find_min_difference(nums, idx, count, sum1, total_sum, min_diff):
4      if count == 5: # 选出5个成员时
5          sum2 = total_sum - sum1
6          return min(min_diff, abs(sum1 - sum2))
7
8      if idx == 10 or count + (10 - idx) < 5: # 剩余人数不足时剪枝
9          return min_diff
10
11     # 选择当前元素
12     min_diff = find_min_difference(nums, idx + 1, count + 1, sum1 + nums[i
dx], total_sum, min_diff)
13
14     # 不选择当前元素
15     min_diff = find_min_difference(nums, idx + 1, count, sum1, total_sum,
min_diff)
16
17     return min_diff
18
19 def main():
20     nums = list(map(int, sys.stdin.read().split())) # 从标准输入读取 10 个整
数
21     total_sum = sum(nums)
22
23     result = find_min_difference(nums, 0, 0, 0, total_sum, float('inf'))
24     print(result)
25
26 if __name__ == "__main__":
27     main()
```

JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let nums = [];
9  rl.on("line", (line) => {
10    nums = line.trim().split(" ").map(Number); // 读取输入并转换为整数数组
11    rl.close();
12  });
13
14  rl.on("close", () => {
15    let totalSum = nums.reduce((a, b) => a + b, 0);
16    let minDiff = Infinity;
17
18    function findMinDifference(idx, count, sum1) {
19      if (count === 5) { // 选出5个成员时
20        let sum2 = totalSum - sum1;
21        minDiff = Math.min(minDiff, Math.abs(sum1 - sum2));
22        return;
23      }
24
25      if (idx === 10 || count + (10 - idx) < 5) return; // 剩余人数不足时
26      // 剪枝
27      findMinDifference(idx + 1, count + 1, sum1 + nums[idx]); // 选择当前元素
28      // 前元素
29      findMinDifference(idx + 1, count, sum1); // 不选择当前元素
30    }
31    findMinDifference(0, 0, 0);
32    console.log(minDiff);
33  });
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 var minDiff = math.MaxInt32
9 var totalSum int
10
11 func findMinDifference(nums []int, idx, count, sum1 int) {
12     if count == 5 { // 选出5个成员时
13         sum2 := totalSum - sum1
14         if diff := abs(sum1 - sum2); diff < minDiff {
15             minDiff = diff
16         }
17         return
18     }
19
20     if idx == 10 || count+(10-idx) < 5 { // 剩余人数不足时剪枝
21         return
22     }
23
24     findMinDifference(nums, idx+1, count+1, sum1+nums[idx]) // 选择当前元素
25     findMinDifference(nums, idx+1, count, sum1)             // 不选择当前元素
26 }
27
28 func abs(x int) int {
29     if x < 0 {
30         return -x
31     }
32     return x
33 }
34
35 func main() {
36     nums := make([]int, 10)
37     totalSum = 0
38
39     // 读取10个整数
40     for i := 0; i < 10; i++ {
41         fmt.Scan(&nums[i])
42         totalSum += nums[i]
43     }
44
45     findMinDifference(nums, 0, 0, 0)
```



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

题解二

思路：由于 $n == 10$ 数据量比较小，直接采用二进制暴力枚举判断。

1. 二进制为1的位置代表会选择对应成员。举个例子101表示会选中第1和第3个成员。
2. 预计算所有成员实力总和 `totalSum`
3. 枚举范围为 $[1, 2^{10} - 1]$ ，对于每个枚举值先求出所有二进制为1的位置用数组 `position` 保存。
 - 首先判断 `position` 的数量是否等于5, 不等于直接跳过。
 - 然后对选中成员进行实力求和 `sum`，两组成员当前的差值为 `abs((totalSum - sum) - sum)`。记录其中出现的最小值就是结果。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <cmath>
5  #include <climits>
6  using namespace std;
7
8  // 快速求出对应数字1个位置
9  vector<int> countOnesAndPositions(int n) {
10     vector<int> positions;
11     int index = 0;
12     while (n) {
13         if (n & 1) positions.push_back(index);
14         n >>= 1;
15         index++;
16     }
17     return positions;
18 }
19
20
21 int main() {
22     vector<int> nums(10);
23     int totalSum = 0;
24     for (int& num : nums) {
25         cin >> num;
26         totalSum += num;
27     }
28     // 初始定义一个绝对不可能取到的最大结果
29     int res = 10000 * 10;
30     for (int i = 1; i < pow(2, 10); i++) {
31         // 计算1的数量以及 位置
32         vector<int> onePosition = countOnesAndPositions(i);
33         if (onePosition.size() != 5) {
34             continue;
35         }
36         int oneTeamSum = 0;
37         for (int j = 0; j < 5; j++) {
38             oneTeamSum += nums[onePosition[j]];
39         }
40         // 计算差值
41         int diff = abs((totalSum - oneTeamSum) - oneTeamSum);
42         res = min(res, diff);
43     }
44     cout << res;
45     return 0;
}
```

```
46 }
```

Java

```
1  import java.util.*;
2
3  public class Main {
4      // 快速求出对应数字1个位置
5      public static List<Integer> countOnesAndPositions(int n) {
6          List<Integer> positions = new ArrayList<>();
7          int index = 0;
8          while (n != 0) {
9              if ((n & 1) == 1) positions.add(index);
10             n >>= 1;
11             index++;
12         }
13         return positions;
14     }
15
16     public static void main(String[] args) {
17         Scanner scanner = new Scanner(System.in);
18         int[] nums = new int[10];
19         int totalSum = 0;
20         for (int i = 0; i < 10; i++) {
21             nums[i] = scanner.nextInt();
22             totalSum += nums[i];
23         }
24         // 初始定义一个绝对不可能取到的最大结果
25         int res = 10000 * 10;
26         // 枚举所有非空子集 (1 到 2^10-1)
27         for (int i = 1; i < (1 << 10); i++) {
28             // 计算1的位置
29             List<Integer> onePosition = countOnesAndPositions(i);
30             if (onePosition.size() != 5) {
31                 continue;
32             }
33             int oneTeamSum = 0;
34             for (int pos : onePosition) {
35                 oneTeamSum += nums[pos];
36             }
37             // 计算差值
38             int diff = Math.abs((totalSum - oneTeamSum) - oneTeamSum);
39             res = Math.min(res, diff);
40         }
41         System.out.println(res);
42     }
43 }
```

Python

Plain Text

```
1  # 快速求出对应数字1个位置
2  def count_ones_and_positions(n):
3      positions = []
4      index = 0
5      while n:
6          if n & 1:
7              positions.append(index)
8              n >>= 1
9              index += 1
10     return positions
11
12 def main():
13     nums = list(map(int, input().split()))
14     total_sum = sum(nums)
15     # 初始定义一个绝对不可能取到的最大结果
16     res = 10000 * 10
17     # 枚举所有非空子集 (1 到 2^10-1)
18     for i in range(1, 1 << 10):
19         # 计算1的位置
20         one_position = count_ones_and_positions(i)
21         if len(one_position) != 5:
22             continue
23         one_team_sum = sum(nums[pos] for pos in one_position)
24         # 计算差值
25         diff = abs((total_sum - one_team_sum) - one_team_sum)
26         res = min(res, diff)
27     print(res)
28
29 if __name__ == "__main__":
30     main()
```

JavaScript

```
1  // 快速求出对应数字1个位置
2  function countOnesAndPositions(n) {
3      const positions = [];
4      let index = 0;
5      while (n) {
6          if (n & 1) positions.push(index);
7          n = n >>> 1;
8          index++;
9      }
10     return positions;
11 }
12
13 function main() {
14     const readline = require('readline');
15     const rl = readline.createInterface({
16         input: process.stdin,
17         output: process.stdout
18     });
19
20     rl.question('', (input) => {
21         const nums = input.split(' ').map(Number);
22         const totalSum = nums.reduce((a, b) => a + b, 0);
23         // 初始定义一个绝对不可能取到的最大结果
24         let res = 10000 * 10;
25         // 枚举所有非空子集 (1 到 2^10-1)
26         for (let i = 1; i < Math.pow(2, 10); i++) {
27             // 计算1的位置
28             const onePosition = countOnesAndPositions(i);
29             if (onePosition.length !== 5) {
30                 continue;
31             }
32             let oneTeamSum = 0;
33             for (const pos of onePosition) {
34                 oneTeamSum += nums[pos];
35             }
36             // 计算差值
37             const diff = Math.abs((totalSum - oneTeamSum) - oneTeamSum);
38             res = Math.min(res, diff);
39         }
40         console.log(res);
41         rl.close();
42     });
43 }
44
45 main();
```

Go

```
1 package main
2
3 import (
4     "fmt"
5     "math"
6 )
7
8 // 快速求出对应数字1个位置
9 func countOnesAndPositions(n int) []int {
10     positions := []int{}
11     index := 0
12     for n > 0 {
13         if n&1 == 1 {
14             positions = append(positions, index)
15         }
16         n >>= 1
17         index++
18     }
19     return positions
20 }
21
22 func main() {
23     nums := make([]int, 10)
24     totalSum := 0
25     for i := range nums {
26         fmt.Scan(&nums[i])
27         totalSum += nums[i]
28     }
29     // 初始定义一个绝对不可能取到的最大结果
30     res := 10000 * 10
31     // 枚举所有非空子集 (1 到 2^10-1)
32     for i := 1; i < int(math.Pow(2, 10)); i++ {
33         // 计算1的位置
34         onePosition := countOnesAndPositions(i)
35         if len(onePosition) != 5 {
36             continue
37         }
38         oneTeamSum := 0
39         for _, pos := range onePosition {
40             oneTeamSum += nums[pos]
41         }
42         // 计算差值
43         diff := int(math.Abs(float64((totalSum - oneTeamSum) - oneTeamSum)))
44         if diff < res {
45             res = diff
46         }
47     }
48 }
```



```
46     }  
47     }  
48     fmt.Println(res)  
49 }
```

来自: [华为OD机试 2025 B卷 – 游戏分组 / 王者荣耀 \(C++ & Python & JAVA & JS & GO\)](#)–CSDN博客

华为OD机试 2025B卷 - 打印机队列 (C++ & Python & JAVA & JS & GO)-CSDN博客

打印机队列

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

华为OD机试2025B卷 100分题型

题目描述

有5台打印机打印文件，每台打印机有自己的待打印队列。

因为打印的文件内容有轻重缓急之分，所以队列中的文件有1~10不同的优先级，其中数字越大优先级越高。

打印机会从自己的待打印队列中选择**优先级最高**的文件来打印。

如果存在两个优先级一样的文件，则选择**最早进入队列**的那个文件。

现在请你来模拟这5台打印机的打印过程。

输入描述

每个输入包含1个测试用例，

每个测试用例第一行给出发生事件的数量N ($0 < N < 1000$)。

接下来有 N 行，分别表示发生的事件。共有如下两种事件：

- **IN P NUM**，表示有一个拥有优先级 NUM 的文件放到了打印机 P 的待打印队列中。（ $0 < P \leq 5$, $0 < NUM \leq 10$ ）；
- **OUT P**，表示打印机 P 进行了一次文件打印，同时该文件从待打印队列中取出。（ $0 < P \leq 5$ ）。

输出描述

- 对于每个测试用例，每次“OUT P”事件，请在一行中**输出文件的编号**。
- 如果此时没有文件可以打印，请输出“NULL”。
- 文件的编号定义为“IN P NUM”事件发生第 x 次，此处待打印文件的编号为x。编号从1开始。

用例1

输入

▼	Plain Text
1 7	
2 IN 1 1	
3 IN 1 2	
4 IN 1 3	
5 IN 2 1	
6 OUT 1	
7 OUT 2	
8 OUT 2	

输出

▼	Plain Text
1 3	
2 4	
3 NULL	

用例2

输入

▼	Plain Text
1 5	
2 IN 1 1	
3 IN 1 3	
4 IN 1 1	
5 IN 1 3	
6 OUT 1	

输出

▼	Plain Text
1 2	

题解

思路： 优先队列 + 模拟 简单练习题

1. 定义一个优先队列数组，长度为5.同时根据题目要求设置优先队列的比较规则，规则为 优先级越大位于栈顶，优先级相同放入时间早的放入栈顶 。

2. 接下根据输入的操作模拟即可：

- **IN P NUM** 操作，将任务放入指定(对应数组索引 $P - 1$)的优先队列中。
- **OUT P** 操作，将指定优先对立(对应数组索引 $p - 1$)的栈顶元素弹出，并输出。如果此时优先队列为空则输出NULL

3. 重复执行2，知道执行所有操作之后，得到的就是结果。

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 using namespace std;
11
12
13 // 通用 split 函数
14 vector<string> split(const string& str, const string& delimiter) {
15     vector<string> result;
16     size_t start = 0;
17     size_t end = str.find(delimiter);
18     while (end != string::npos) {
19         result.push_back(str.substr(start, end - start));
20         start = end + delimiter.length();
21         end = str.find(delimiter, start);
22     }
23     // 添加最后一个部分
24     result.push_back(str.substr(start));
25     return result;
26 }
27
28 // 自定义比较函数（小顶堆）
29 struct cmp {
30     bool operator()(pair<int, int> p1, pair<int, int> p2) {
31         if (p1.first == p2.first) {
32             return p1.second > p2.second;
33         }
34         return p1.first < p2.first;
35     }
36 };
37
38 int main() {
39     int n ;
40     cin >> n;
41     cin.ignore();
42     // 优先队列
43     vector<priority_queue<pair<int,int>, vector<pair<int, int>>, cmp>> an
44     s(5);
45     int x = 1;
```

```

45     for (int i = 0; i < n; i++) {
46         string s;
47         getline(cin, s);
48         vector<string> tmp = split(s, " ");
49         int index = stoi(tmp[1]) - 1;
50         if (tmp[0] == "IN") {
51             ans[index].push({stoi(tmp[2]), x});
52             x++;
53         } else {
54             if (ans[index].empty()) {
55                 cout << "NULL" << endl;
56             } else {
57                 cout << ans[index].top().second << endl;
58                 ans[index].pop();
59             }
60         }
61     }
62     return 0;
63 }

```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      // 通用 split 函数
5      public static List<String> split(String str, String delimiter) {
6          return Arrays.asList(str.split(delimiter));
7      }
8
9      // 自定义比较器 (小顶堆)
10     static class CustomComparator implements Comparator<int[]> {
11         @Override
12         public int compare(int[] p1, int[] p2) {
13             if (p1[0] == p2[0]) {
14                 return Integer.compare(p1[1], p2[1]);
15             }
16             return Integer.compare(p2[0], p1[0]);
17         }
18     }
19
20     public static void main(String[] args) {
21         Scanner scanner = new Scanner(System.in);
22         int n = Integer.parseInt(scanner.nextLine());
23
24         // 初始化优先队列
25         List<PriorityQueue<int[]>> ans = new ArrayList<>();
26         for (int i = 0; i < 5; i++) {
27             ans.add(new PriorityQueue<>(new CustomComparator()));
28         }
29
30         int x = 1;
31         for (int i = 0; i < n; i++) {
32             String s = scanner.nextLine();
33             List<String> tmp = split(s, " ");
34             int index = Integer.parseInt(tmp.get(1)) - 1;
35
36             if (tmp.get(0).equals("IN")) {
37                 ans.get(index).offer(new int[]{Integer.parseInt(tmp.get
(2)), x});
38                 x++;
39             } else {
40                 if (ans.get(index).isEmpty()) {
41                     System.out.println("NULL");
42                 } else {
43                     System.out.println(ans.get(index).poll()[1]);
44                 }
45             }
46         }
47     }
48 }
```

```
45         }
46     }
47     scanner.close();
48 }
49 }
```

Python

```
▼ Plain Text |
1  import sys
2  import heapq
3
4  # 读取输入
5  n = int(sys.stdin.readline().strip())
6
7  # 优先队列数组 (最大堆)
8  ans = [ [] for _ in range(5) ]
9
10 x = 1
11 for _ in range(n):
12     tmp = sys.stdin.readline().strip().split()
13     index = int(tmp[1]) - 1
14
15     if tmp[0] == "IN":
16         # 由于 heapq 是最小堆, 为了实现最大堆, 我们将 first 取负数
17         heapq.heappush(ans[index], (-int(tmp[2]), x))
18         x += 1
19     else:
20         if not ans[index]:
21             print("NULL")
22         else:
23             # 取出时恢复 first 的原值
24             print(heapq.heappop(ans[index])[1])
```

JavaScript


```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout,
6  });
7
8  class PriorityQueue {
9    constructor(compare) {
10      this.heap = [];
11      this.compare = compare;
12    }
13
14    push(item) {
15      this.heap.push(item);
16      this._heapifyUp();
17    }
18
19    pop() {
20      if (this.isEmpty()) return null;
21      const top = this.heap[0];
22      const last = this.heap.pop();
23      if (!this.isEmpty()) {
24        this.heap[0] = last;
25        this._heapifyDown();
26      }
27      return top;
28    }
29
30    top() {
31      return this.isEmpty() ? null : this.heap[0];
32    }
33
34    isEmpty() {
35      return this.heap.length === 0;
36    }
37
38    _heapifyUp() {
39      let index = this.heap.length - 1;
40      while (index > 0) {
41        let parent = Math.floor((index - 1) / 2);
42        if (this.compare(this.heap[parent], this.heap[index]) > 0) {
43          [this.heap[parent], this.heap[index]] = [this.heap[index], this.h
44            eap[parent]];
45          index = parent;
46        }
47      }
48    }
49
50    _heapifyDown() {
51      let index = 0;
52      while (true) {
53        let left = 2 * index + 1;
54        let right = 2 * index + 2;
55        let smallest = index;
56        if (left < this.heap.length && this.compare(this.heap[left], this.heap[smallest]) < 0) {
57          smallest = left;
58        }
59        if (right < this.heap.length && this.compare(this.heap[right], this.heap[smallest]) < 0) {
60          smallest = right;
61        }
62        if (smallest === index) break;
63        [this.heap[index], this.heap[smallest]] = [this.heap[smallest], this.heap[index]];
64        index = smallest;
65      }
66    }
67  }
```

```

45         } else {
46             break;
47         }
48     }
49 }
50
51 _heapifyDown() {
52     let index = 0;
53     let length = this.heap.length;
54     while (true) {
55         let left = 2 * index + 1;
56         let right = 2 * index + 2;
57         let smallest = index;
58
59         if (left < length && this.compare(this.heap[left], this.heap[smallest]) < 0) {
60             smallest = left;
61         }
62         if (right < length && this.compare(this.heap[right], this.heap[smallest]) < 0) {
63             smallest = right;
64         }
65         if (smallest !== index) {
66             [this.heap[smallest], this.heap[index]] = [this.heap[index], this.heap[smallest]];
67             index = smallest;
68         } else {
69             break;
70         }
71     }
72 }
73
74
75 // 自定义比较函数 (实现最大堆, 保证 优先级 大的在前, 优先级 相等时 编号 小的在前)
76 const compare = (a, b) => {
77     if (a[0] === b[0]) return a[1] - b[1]; // 编号 小的在前
78     return b[0] - a[0]; // 优先级 大的在前
79 };
80
81 let ans = Array.from({ length: 5 }, () => new PriorityQueue(compare));
82 let x = 1;
83
84 rl.on("line", (line) => {
85     const tmp = line.split(" ");
86     if (tmp.length === 1) return; // 忽略无效输入
87
88     const index = parseInt(tmp[1]) - 1;
89

```

```
90     if (tmp[0] === "IN") {
91         ans[index].push([parseInt(tmp[2]), x]);
92         x++;
93     } else {
94         if (ans[index].isEmpty()) {
95             console.log("NULL");
96         } else {
97             console.log(ans[index].pop()[1]);
98         }
99     }
100 });
```

Go

```
1  package main
2
3  import (
4      "container/heap"
5      "bufio"
6      "fmt"
7      "os"
8      "strconv"
9      "strings"
10 )
11
12 // 定义优先队列结构体
13 type Pair struct {
14     first  int
15     second int
16 }
17
18 // 定义堆结构 (实现 container/heap 接口)
19 type PriorityQueue []Pair
20
21 func (pq PriorityQueue) Len() int { return len(pq) }
22
23 // 实现排序逻辑
24 // `优先级` 大的在前, `优先级` 相同的 `编号` 小的在前
25 func (pq PriorityQueue) Less(i, j int) bool {
26     if pq[i].first == pq[j].first {
27         return pq[i].second < pq[j].second
28     }
29     return pq[i].first > pq[j].first
30 }
31
32 func (pq PriorityQueue) Swap(i, j int) { pq[i], pq[j] = pq[j], pq[i] }
33
34 func (pq *PriorityQueue) Push(x interface{}) {
35     *pq = append(*pq, x.(Pair))
36 }
37
38 func (pq *PriorityQueue) Pop() interface{} {
39     old := *pq
40     n := len(old)
41     item := old[n-1]
42     *pq = old[0 : n-1]
43     return item
44 }
45
```

```

46 func main() {
47     scanner := bufio.NewScanner(os.Stdin)
48     scanner.Scan()
49     n, _ := strconv.Atoi(scanner.Text())
50
51     // 创建 5 个优先队列
52     ans := make([]PriorityQueue, 5)
53     for i := range ans {
54         heap.Init(&ans[i])
55     }
56
57     x := 1
58     for i := 0; i < n; i++ {
59         scanner.Scan()
60         tmp := strings.Split(scanner.Text(), " ")
61         index, _ := strconv.Atoi(tmp[1])
62         index--
63
64         if tmp[0] == "IN" {
65             first, _ := strconv.Atoi(tmp[2])
66             heap.Push(&ans[index], Pair{first, x})
67             x++
68         } else {
69             if len(ans[index]) == 0 {
70                 fmt.Println("NULL")
71             } else {
72                 fmt.Println(heap.Pop(&ans[index]).(Pair).second)
73             }
74         }
75     }
76 }

```

来自: [华为OD机试 2025B卷 – 打印机队列 \(C++ & Python & JAVA & JS & GO\)-CSDN博客](#)

华为OD机试2025B卷 - 比赛 (C++ & Python & JAVA & JS & GO)_华为od机考2025b卷-CSDN博客

比赛

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

华为OD2025B卷 100分题型

题目描述

一个有N个选手参加比赛，选手编号为1~N ($3 \leq N \leq 100$)，有M ($3 \leq M \leq 10$) 个评委对选手进行打分。

打分规则为每个评委对选手打分，最高分10分，最低分1分。

请计算得分最多的3位选手的编号。如果得分相同，则得分高分值最多的选手排名靠前 (10分数量相同，则比较9分的数量，以此类推，用例中不会出现多个选手得分完全相同的情况)。

输入描述

第一行为半角逗号分割的两个正整数，第一个数字表示M ($3 \leq M \leq 10$) 个评委，第二个数字表示N ($3 \leq N \leq 100$) 个选手。

第2到M+1行是半角逗号分割的整数序列，表示评委为每个选手的打分，0号下标数字表示1号选手分数，1号下标数字表示2号选手分数，依次类推。

输出描述

选手前3名的编号。

注：若输入异常，输出-1，如M、N、打分不在范围内。

用例1

输入

▼	Plain Text
1 4,5 2 10,6,9,7,6 3 9,10,6,7,5 4 8,10,6,5,10 5 9,10,8,4,9	

输出

▼	Plain Text
1 2,1,5	

说明

第一行代表有4个评委，5个选手参加比赛
矩阵代表是4*5，每个数字是选手的编号，每一行代表一个评委对选手的打分排序，
2号选手得分36分排第1，1号选手36分排第2，5号选手30分(2号10分值有3个，1号10分值只有1个，所以2号排第一)

用例2

输入

▼	Plain Text
1 2,5 2 7,3,5,4,2 3 8,5,4,4,3	

输出

▼	Plain Text
1 -1	

说明

只有2个评委，要求最少为3个评委。

用例3

输入

▼		Plain Text
1	4,2	
2	8,5	
3	5,6	
4	10,4	
5	8,9	

输出

▼		Plain Text
1	-1	

说明

只有两名选手参加

用例4

输入

▼		Plain Text
1	4,5	
2	11,6,9,7,8	
3	9,10,6,7,8	
4	8,10,6,9,7	
5	9,10,8,6,7	

输出

▼		Plain Text
1	-1	

说明

第一个评委给第一个选手打分11，无效分数

题解

思路： 模拟 + 自定义排序

1. 接收输入时，首先判断数据合法性，判断 选手数量、评委数量、打分值是否在指定范围内，如果不满足题目描述要求直接输出-1，结束。
2. 统计每个选手的获得的总分，以及得到各个分数的数量。
3. 自定义排序，排序规则 先按照选手获得总分降序进行排序，总分相同的情况按照获得高分值的数量进行降序排序。
4. 经过3排序之后，输出排名前3选手的编号。

C++

```
1  #include <algorithm>
2  #include <iostream>
3  #include <sstream>
4  #include <string>
5  #include <utility>
6  #include <vector>
7  using namespace std;
8
9  // 通用 split 函数
10 vector<string> split(const string &str, const string &delimiter) {
11     vector<string> result;
12     size_t start = 0;
13     size_t end = str.find(delimiter);
14     while (end != string::npos) {
15         result.push_back(str.substr(start, end - start));
16         start = end + delimiter.length();
17         end = str.find(delimiter, start);
18     }
19     // 添加最后一个部分
20     result.push_back(str.substr(start));
21     return result;
22 }
23
24 struct Person {
25     // 记录选手获得[1-10]各个分数的数量
26     vector<int> score;
27     // 统计用户的获得的总分
28     int sum;
29     // 记录选手下表
30     int index;
31     Person() : score(11, 0), sum(0), index(0) {}
32 };
33
34 // 按照sum进行从大到小排序, sum相等按照score从大到小排序
35 bool cmp(Person a, Person b) {
36     if (a.sum == b.sum) {
37         for (int i = 10; i >= 0; i--) {
38             if (a.score[i] != b.score[i]) {
39                 return a.score[i] > b.score[i];
40             }
41         }
42     }
43     return a.sum > b.sum;
44 }
45
```

```

46 int main() {
47     int n, m;
48     string line;
49     getline(cin, line);
50     vector<string> s = split(line, ",");
51     n = stoi(s[1]);
52     m = stoi(s[0]);
53     // 判断数据合法性
54     if (n < 3 || n > 100 || m < 3 || m > 10) {
55         cout << -1;
56         return 0;
57     }
58     vector<Person> p(n, Person());
59     for (int i = 0; i < m; i++) {
60         getline(cin, line);
61         s = split(line, ",");
62         for (int j = 0; j < s.size(); j++) {
63             // 判断数据合法性
64             if (stoi(s[j]) < 1 || stoi(s[j]) > 10) {
65                 cout << -1;
66                 return 0;
67             }
68             // 更新选手得到的分数情况
69             p[j].score[stoi(s[j])] = p[j].score[stoi(s[j])] + 1;
70             p[j].sum += stoi(s[j]);
71             p[j].index = j;
72         }
73     }
74     // 根据自定义规则进行排序
75     sort(p.begin(), p.end(), cmp);
76     // 输出结果
77     for (int i = 0; i < 3; i++) {
78         if (i != 0) {
79             cout << ",";
80         }
81         cout << p[i].index + 1;
82     }
83     return 0;
84 }

```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5
6      static class Person {
7          int[] score = new int[11]; // 分数范围是 1 到 10, 所以需要 11 个位置
8          int sum = 0;
9          int index = 0;
10
11         public Person() {
12             Arrays.fill(score, 0);
13         }
14     }
15
16     // 按照 sum 从大到小排序; 如果 sum 相等, 则按 score 从大到小排序
17     public static int cmp(Person a, Person b) {
18         if (a.sum == b.sum) {
19             for (int i = 10; i >= 1; i--) {
20                 if (a.score[i] != b.score[i]) {
21                     return a.score[i] > b.score[i] ? -1 : 1;
22                 }
23             }
24         }
25         return a.sum > b.sum ? -1 : 1;
26     }
27
28     public static void main(String[] args) {
29         Scanner scanner = new Scanner(System.in);
30         String line = scanner.nextLine();
31         String[] s = line.split(",");
32
33         int n = Integer.parseInt(s[1]);
34         int m = Integer.parseInt(s[0]);
35
36         if (n < 3 || n > 100 || m < 3 || m > 10) {
37             System.out.println(-1);
38             return;
39         }
40
41         List<Person> persons = new ArrayList<>();
42         for (int i = 0; i < n; i++) {
43             persons.add(new Person());
44         }
45     }
```

```

46     for (int i = 0; i < m; i++) {
47         line = scanner.nextLine();
48         s = line.split(",");
49
50         for (int j = 0; j < s.length; j++) {
51             int score = Integer.parseInt(s[j]);
52             if (score < 1 || score > 10) {
53                 System.out.println(-1);
54                 return;
55             }
56             persons.get(j).score[score]++;
57             persons.get(j).sum += score;
58             persons.get(j).index = j;
59         }
60     }
61
62     // 排序
63     persons.sort((a, b) -> cmp(a, b));
64
65     // 输出前三名的 index
66     for (int i = 0; i < 3; i++) {
67         if (i != 0) {
68             System.out.print(",");
69         }
70         System.out.print(persons.get(i).index + 1);
71     }
72 }
73 }

```

Python

```
1 class Person:
2     def __init__(self):
3         # 分数范围为 1 - 10 每个分数的数量
4         self.score = [0] * 11
5         # 总分
6         self.sum = 0
7         # 下标
8         self.index = 0
9
10    def cmp(a, b):
11        if a.sum == b.sum:
12            for i in range(10, -1, -1):
13                if a.score[i] != b.score[i]:
14                    return a.score[i] > b.score[i]
15        return a.sum > b.sum
16
17    def main():
18        line = input()
19        s = line.split(',')
20        m, n = int(s[0]), int(s[1])
21
22        # 判断合法性
23        if not (3 <= n <= 100 and 3 <= m <= 10):
24            print(-1)
25            return
26
27        p = [Person() for _ in range(n)]
28
29        for i in range(m):
30            line = input()
31            s = line.split(',')
32            for j in range(len(s)):
33                score = int(s[j])
34                # 判断合法性
35                if not (1 <= score <= 10):
36                    print(-1)
37                    return
38                p[j].score[score] += 1
39                p[j].sum += score
40                p[j].index = j
41
42        # 先按照总分排序, 总分相等按照10 -1 比较分数个数 看不懂可以参照上面cmp的逻辑
43        p.sort(key=lambda person: (person.sum, person.score[::-1]), reverse=True)
44
```

```
45     # 输出结果
46     print(",".join(str(p[i].index + 1) for i in range(3)))
47
48 if __name__ == "__main__":
49     main()
```

JavaScript

```
1  const readline = require("readline");
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout,
6  });
7
8  let inputLines = [];
9
10 rl.on("line", (line) => {
11   inputLines.push(line);
12 }).on("close", () => {
13   const input = inputLines;
14
15   const [m, n] = input[0].split(",").map(Number);
16
17   // 判断数据合法性
18   if (n < 3 || n > 100 || m < 3 || m > 10) {
19     console.log(-1);
20     return;
21   }
22
23   const players = Array.from({ length: n }, () => ({
24     // 记录1 - 10分数的数量
25     score: Array(11).fill(0),
26     // 总分
27     sum: 0,
28     index: 0,
29   }));
30
31   for (let i = 1; i <= m; i++) {
32     const scores = input[i].split(",").map(Number);
33
34     for (let j = 0; j < scores.length; j++) {
35       const score = scores[j];
36
37       // 判断分数合法性
38       if (score < 1 || score > 10) {
39         console.log(-1);
40         return;
41       }
42
43       players[j].score[score]++;
44       players[j].sum += score;
45       players[j].index = j;
```



```

46     }
47   }
48
49   players.sort((a, b) => {
50     if (a.sum !== b.sum) {
51       return b.sum - a.sum;
52     }
53     // 总分相等按照分数排序
54     for (let i = 10; i >= 1; i--) {
55       if (a.score[i] !== b.score[i]) {
56         return b.score[i] - a.score[i];
57       }
58     }
59     return 0;
60   });
61
62   console.log(
63     players
64       .slice(0, 3)
65       .map((player) => player.index + 1)
66       .join(",")
67   );
68 });

```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12
13
14 // Person 结构体
15 type Person struct {
16     score []int // 记录选手获得[1-10]各个分数的数量
17     sum    int    // 统计用户的获得的总分
18     index int    // 记录选手下标
19 }
20
21 // 比较函数: 按照 sum 从大到小排序, sum 相等时按照 score 从大到小排序
22 func cmp(a, b Person) bool {
23     if a.sum == b.sum {
24         for i := 10; i >= 1; i-- {
25             if a.score[i] != b.score[i] {
26                 return a.score[i] > b.score[i]
27             }
28         }
29     }
30     return a.sum > b.sum
31 }
32
33 func main() {
34     reader := bufio.NewReader(os.Stdin)
35
36     // 读取第一行输入
37     line, _ := reader.ReadString('\n')
38     line = strings.TrimSpace(line)
39     inputs := strings.Split(line, ",")
40
41     m, _ := strconv.Atoi(inputs[0])
42     n, _ := strconv.Atoi(inputs[1])
43
44     // 判断数据合法性
45     if n < 3 || n > 100 || m < 3 || m > 10 {
```

```

46     fmt.Println(-1)
47     return
48 }
49
50 // 初始化选手数据
51 players := make([]Person, n)
52 for i := 0; i < n; i++ {
53     players[i] = Person{
54         score: make([]int, 11),
55         sum:    0,
56         index: i,
57     }
58 }
59
60 // 读取每轮分数
61 for i := 0; i < m; i++ {
62     line, _ := reader.ReadString('\n')
63     line = strings.TrimSpace(line)
64     scores := strings.Split(line, ",")
65
66     for j, scoreStr := range scores {
67         score, _ := strconv.Atoi(scoreStr)
68
69         // 判断分数合法性
70         if score < 1 || score > 10 {
71             fmt.Println(-1)
72             return
73         }
74
75         players[j].score[score]++
76         players[j].sum += score
77     }
78 }
79
80 // 排序选手
81 sort.Slice(players, func(i, j int) bool {
82     return cmp(players[i], players[j])
83 })
84
85 // 输出前三名选手的下标
86 for i := 0; i < 3; i++ {
87     if i > 0 {
88         fmt.Print(",")
89     }
90     fmt.Print(players[i].index + 1)
91 }
92 }

```

来自: [华为OD机试2025B卷 – 比赛 \(C++ & Python & JAVA & JS & GO\)_华为od机考2025b卷–CSDN博客](#)

华为OD机试2025B卷 - 密码解密 (C++ & Python & JAVA & JS & GO)-CSDN博客

密码解密

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

华为OD机试2025B卷 100分题型

题目描述

给定一段“密文”字符串 s，其中字符都是经过“密码本”映射的，现需要将“密文”解密并输出。映射的规则（‘a’ ~ ‘i’）分别用（‘1’ ~ ‘9’）表示；（‘j’ ~ ‘z’）分别用（“10*” ~ “26*”）表示。约束：映射始终唯一。

输入描述

“密文”字符串

输出描述

明文字符串。翻译后的文本长度在100以内

用例1

输入

	Plain Text
1	20*19*20*

输出

	Plain Text
1	tst

题解一

思路：使用 正则表达式 替换

使用正则表达式进行替换，不过要优先替换 10* - 26*，然后替换 1-9。不然 1-9 的替换会影响到 10* - 26*，具体逻辑可参照下面代码。

C++

Plain Text

```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<regex>
8  using namespace std;
9
10 int main() {
11     string input;
12     getline(cin, input);
13     // 正则替换优先选择替换 10* - 26*
14     for (int i = 26; i >= 1; i--) {
15         string regexStr = to_string(i);
16         if (i > 9) {
17             regexStr += "\\*";
18         }
19         regex pattern(regexStr);
20         string replaceChar = string(1, 'a' + i - 1);
21         input = regex_replace(input, pattern, replaceChar);
22     }
23     cout << input;
24     return 0;
25 }
```

JAVA

```
1  import java.util.Scanner;
2  import java.util.regex.*;
3
4  public class Main {
5      public static void main(String[] args) {
6          Scanner scanner = new Scanner(System.in);
7          String input = scanner.nextLine();
8
9          // 正则替换优先选择替换 10* - 26*
10         for (int i = 26; i >= 1; i--) {
11             String regexStr = Integer.toString(i);
12             if (i > 9) {
13                 regexStr += "\\*"; // 需要转义 *
14             }
15             Pattern pattern = Pattern.compile(regexStr);
16             String replaceChar = String.valueOf((char) ('a' + i - 1));
17             Matcher matcher = pattern.matcher(input);
18             input = matcher.replaceAll(replaceChar);
19         }
20
21         System.out.println(input);
22     }
23 }
```

Python

```
1 import re
2
3 def main():
4     input_str = input()
5
6     # 正则替换优先选择替换 10* - 26*
7     for i in range(26, 0, -1):
8         regex_str = str(i)
9         if i > 9:
10             regex_str += r"\*" # 转义 *
11             replace_char = chr(ord('a') + i - 1)
12             input_str = re.sub(regex_str, replace_char, input_str)
13
14     print(input_str)
15
16 if __name__ == "__main__":
17     main()
```

JavaScript

```
1 function main() {
2     const fs = require("fs");
3     let input = fs.readFileSync(0, "utf-8").trim();
4
5     // 正则替换优先选择替换 10* - 26*
6     for (let i = 26; i >= 1; i--) {
7         let regexStr = i.toString();
8         if (i > 9) {
9             regexStr += "\\*"; // 需要转义 *
10        }
11        const pattern = new RegExp(regexStr, "g");
12        const replaceChar = String.fromCharCode('a'.charCodeAt(0) + i -
13    1);
14        input = input.replace(pattern, replaceChar);
15    }
16    console.log(input);
17 }
18
19 main();
```



```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "regexp"
8     "strconv"
9 )
10
11 func main() {
12     reader := bufio.NewReader(os.Stdin)
13     input, _ := reader.ReadString('\n')
14     input = input[:len(input)-1] // 去除换行
15
16     // 正则替换优先选择替换 10* - 26*
17     for i := 26; i >= 1; i-- {
18         regexStr := strconv.Itoa(i)
19         if i > 9 {
20             regexStr += "\\*" // 转义 *
21         }
22         re := regexp.MustCompile(regexStr)
23         replaceChar := string('a' + i - 1)
24         input = re.ReplaceAllString(input, replaceChar)
25     }
26
27     fmt.Println(input)
28 }
```

题解二

思路：使用栈进行替换。比较原始的一种解题代码

1. 大体思路其实也是先替换 `10*` 这种形式的子字符串。
2. 代码的基本逻辑：定义一个 `stk` 存储字符，从前往后遍历输入字符串每个字符，分情况处理
 - 当前遍历字符为 `数字字符` 直接压入栈。
 - 当前遍历字符为 `*`，开始进行转换栈中数字字符，栈顶两位数字和*进行匹配对应为[j,z]其中一个字符。剩余栈中其它数字字符每一个对应[a,i]其中一个字符。除了栈顶两个数字字符，其它数字字符都是单个匹配一个英文字符，不然早就转换了
3. 按照2的逻辑进行遍历原输入字符串处理。注意：最后需要进行收尾操作。
4. 输出转换后的字符串。


```
1  #include<iostream>
2  #include<vector>
3  #include<string>
4  #include <utility>
5  #include <sstream>
6  #include<algorithm>
7  #include<stack>
8  using namespace std;
9
10 int main() {
11     string input;
12     getline(cin, input);
13     string res = "";
14     stack<char> stk;
15     int n = input.size();
16     for (int i = 0; i < n; i++) {
17         // 压栈
18         if (input[i] != '*') {
19             stk.push(input[i]);
20         } else {
21             // 遇到*出栈, 并弹出两位
22             char c1 = stk.top();
23             stk.pop();
24             char c2 = stk.top();
25             stk.pop();
26             string prefix = "";
27             // 栈中剩余单个的数字都是 属于 a - i
28             while (!stk.empty()) {
29                 char c = stk.top();
30                 stk.pop();
31                 prefix.push_back('a' + (c - '1'));
32             }
33             if (!prefix.empty()) {
34                 reverse(prefix.begin(), prefix.end());
35             }
36
37             string tmp = string() + c2 + c1;
38             prefix.push_back('j' + (stoi(tmp) - 10));
39             res += prefix;
40         }
41     }
42
43     // 收尾操作
44     if (!stk.empty()) {
45         string prefix = "";
```

```
46         while (!stk.empty()) {
47             char c = stk.top();
48             stk.pop();
49             prefix.push_back('a' + (c - '1'));
50         }
51         reverse(prefix.begin(), prefix.end());
52         res += prefix;
53     }
54     cout << res;
55 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) {
5          Scanner scanner = new Scanner(System.in);
6          String input = scanner.nextLine();
7          StringBuilder res = new StringBuilder();
8          Stack<Character> stk = new Stack<>();
9
10         int n = input.length();
11         for (int i = 0; i < n; i++) {
12             // 压栈
13             if (input.charAt(i) != '*') {
14                 stk.push(input.charAt(i));
15             } else {
16                 // 遇到*出栈, 并弹出两位
17                 char c1 = stk.pop();
18                 char c2 = stk.pop();
19                 StringBuilder prefix = new StringBuilder();
20
21                 // 栈中剩余单个的数字都是属于 a - i
22                 while (!stk.isEmpty()) {
23                     char c = stk.pop();
24                     prefix.append((char) ('a' + (c - '1')));
25                 }
26
27                 if (prefix.length() > 0) {
28                     prefix.reverse();
29                 }
30
31                 String tmp = "" + c2 + c1;
32                 prefix.append((char) ('j' + (Integer.parseInt(tmp) - 1
0)))));
33
34                 res.append(prefix);
35             }
36         }
37
38         // 收尾操作
39         if (!stk.isEmpty()) {
40             StringBuilder prefix = new StringBuilder();
41             while (!stk.isEmpty()) {
42                 char c = stk.pop();
43                 prefix.append((char) ('a' + (c - '1')));
44             }
```

```
45         prefix.reverse();
46         res.append(prefix);
47     }
48
49     System.out.println(res.toString());
50 }
51 }
```

Python

```
1 def main():
2     input_str = input()
3     res = []
4     stk = []
5
6     for ch in input_str:
7         # 压栈
8         if ch != '*':
9             stk.append(ch)
10        else:
11            # 遇到*出栈, 并弹出两位
12            c1 = stk.pop()
13            c2 = stk.pop()
14            prefix = []
15
16            # 栈中剩余单个的数字都是属于 a - i
17            while stk:
18                c = stk.pop()
19                prefix.append(chr(ord('a') + int(c) - 1))
20
21            prefix.reverse()
22
23            tmp = c2 + c1
24            prefix.append(chr(ord('j') + int(tmp) - 10))
25            res.extend(prefix)
26
27        # 收尾操作
28        if stk:
29            prefix = []
30            while stk:
31                c = stk.pop()
32                prefix.append(chr(ord('a') + int(c) - 1))
33            prefix.reverse()
34            res.extend(prefix)
35
36        print("".join(res))
37
38 if __name__ == "__main__":
39     main()
```

JavaScript

```
1  function main() {
2      const fs = require("fs");
3      const input = fs.readFileSync(0, "utf-8").trim();
4      const stk = [];
5      let res = "";
6
7      for (let i = 0; i < input.length; i++) {
8          if (input[i] !== '*') {
9              stk.push(input[i]);
10         } else {
11             // 遇到*出栈，并弹出两位
12             const c1 = stk.pop();
13             const c2 = stk.pop();
14             const prefix = [];
15
16             // 栈中剩余单个的数字都是属于 a - i
17             while (stk.length > 0) {
18                 const c = stk.pop();
19                 prefix.push(String.fromCharCode('a'.charCodeAt(0) + Number
(c) - 1));
20             }
21
22             prefix.reverse();
23
24             const tmp = c2 + c1;
25             prefix.push(String.fromCharCode('j'.charCodeAt(0) + Number(tm
p) - 10));
26             res += prefix.join("");
27         }
28     }
29
30     // 收尾操作
31     if (stk.length > 0) {
32         const prefix = [];
33         while (stk.length > 0) {
34             const c = stk.pop();
35             prefix.push(String.fromCharCode('a'.charCodeAt(0) + Number(c)
- 1));
36         }
37         prefix.reverse();
38         res += prefix.join("");
39     }
40
41     console.log(res);
42 }
```


43
44

```
main();
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strconv"
8 )
9
10 func main() {
11     var input string
12     reader := bufio.NewReader(os.Stdin)
13     input, _ = reader.ReadString('\n')
14     input = input[:len(input)-1] // 去掉换行符
15
16     var stk []byte
17     var res []byte
18
19     for i := 0; i < len(input); i++ {
20         if input[i] != '*' {
21             stk = append(stk, input[i])
22         } else {
23             // 遇到*出栈，并弹出两位
24             c1 := stk[len(stk)-1]
25             stk = stk[:len(stk)-1]
26             c2 := stk[len(stk)-1]
27             stk = stk[:len(stk)-1]
28
29             var prefix []byte
30             // 栈中剩余单个的数字都是属于 a - i
31             for len(stk) > 0 {
32                 c := stk[len(stk)-1]
33                 stk = stk[:len(stk)-1]
34                 prefix = append(prefix, byte('a'+(c-'1'))))
35             }
36
37             // 反转 prefix
38             for i, j := 0, len(prefix)-1; i < j; i, j = i+1, j-1 {
39                 prefix[i], prefix[j] = prefix[j], prefix[i]
40             }
41
42             tmp, _ := strconv.Atoi(string([]byte{c2, c1}))
43             prefix = append(prefix, byte('j'+(tmp-10)))
44             res = append(res, prefix...)
45         }
46     }
```

```

46     }
47
48     // 收尾操作
49     if len(stk) > 0 {
50         var prefix []byte
51         for len(stk) > 0 {
52             c := stk[len(stk)-1]
53             stk = stk[:len(stk)-1]
54             prefix = append(prefix, byte('a'+(c-'1')))
55         }
56         // 反转 prefix
57         for i, j := 0, len(prefix)-1; i < j; i, j = i+1, j-1 {
58             prefix[i], prefix[j] = prefix[j], prefix[i]
59         }
60         res = append(res, prefix...)
61     }
62
63     fmt.Println(string(res))
64 }

```

来自: [华为OD机试2025B卷 – 密码解密 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)

华为OD机试 2025B卷 - 箱子Z字形摆放 (C++ & Python & JAVA & JS & GO)_箱子z字形摆放华为od-CSDN博客

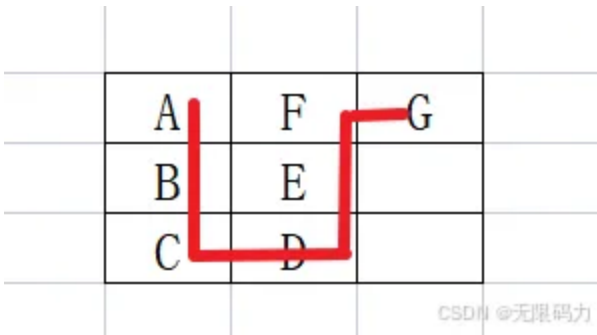
箱子Z字形摆放

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

华为OD机试真题2025B卷 100分题型

题目描述

有一批箱子（形式为字符串，设为str），要求将这批箱子按从上到下以之字形的顺序摆放在宽度为 n 的空地，请输出箱子的摆放位置。例如：箱子ABCDEFGG，空地宽度为3，摆放结果如图：



则输出结果为：

▼ Plain Text

```
1  `AFG`  
2  `BE`  
3  `CD`
```

输入描述

输入一行字符串，通过空格分隔，前面部分为字母或数字组成的字符串str，表示箱子；后面部分为数字n，表示空地的宽度。例如：

▼	Plain Text
1	ABCDEFGG 3

输出描述

箱子摆放结果，如题目示例所示

▼	Plain Text
1	AFG
2	BE
3	CD

备注

- 1. 请不要在最后一行输出额外的空行
- 2. str只包含字母和数字， $1 \leq \text{len}(\text{str}) \leq 1000$
- 3. $1 \leq n \leq 1000$

用例1

输入

▼	Plain Text
1	ABCDEFGG 3

输出

▼	Plain Text
1	AFG
2	BE
3	CD

题解

思路：

定义一个二维数组，一维长度为 n，以上面测试数据为例n= 3

▼ Plain Text |

```
1  {
2    {},
3    {},
4    {},
5  }
```

遍历每个字符串，前3个字符串使用 $(i \% 3)$ 确定插入位置。插入之后如下图

▼ Plain Text |

```
1  {
2    {A},
3    {B},
4    {C},
5  }
```

当触底之后，肯定遍历第四个时, $i = 3, (3 \% 3) = 0$, 上述公式不成立。其实可以明显知道 (i / n) 为偶数时向下，奇数是向上。这时候要进行反转 采用公式变换为 $(n - 1 - (i \% n))$ 。

▼ Plain Text |

```
1  {
2    {A},
3    {B},
4    {CD},
5  }
```

所以对于 (i / n) 奇数和偶数不同情况下，需要采用不同的计算公式，在实际过程中可以通过布尔值进行标记。根据标记采用不同公式。

C++

```
1  #include <iostream>
2  #include <vector>
3  #include <string>
4  using namespace std;
5
6  // 按照之字形方式填充矩阵并输出
7  void zigzagFill(const string& str, int n) {
8      // 创建矩阵, 每行是一个 vector<char>
9      vector<vector<char>> matrix(n);
10
11      bool reverse = true; // 控制方向
12      for (int i = 0; i < str.length(); i++) {
13          int k = i % n; // 计算当前字符所在的行号
14          if (k == 0) reverse = !reverse; // 每当到达行的起点时, 反转方向
15
16          // 如果是反向填充, 调整行号
17          if (reverse) k = n - 1 - k;
18
19          matrix[k].push_back(str[i]);
20      }
21
22      // 输出矩阵内容
23      for (const auto& row : matrix) {
24          for (char c : row) {
25              cout << c;
26          }
27          cout << endl; // 输出换行
28      }
29
30 }
31
32 int main() {
33     string str;
34     int n;
35     cin >> str >> n;
36
37     zigzagFill(str, n);
38
39     return 0;
40 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5      // 按照之字形方式填充矩阵并输出
6      public static void zigzagFill(String str, int n) {
7          // 创建矩阵, 每行是一个 List<Character>
8          List<List<Character>> matrix = new ArrayList<>();
9          for (int i = 0; i < n; i++) {
10             matrix.add(new ArrayList<>());
11         }
12
13         boolean reverse = true; // 控制方向
14         for (int i = 0; i < str.length(); i++) {
15             int k = i % n; // 计算当前字符所在的行号
16             if (k == 0) reverse = !reverse; // 每当到达行的起点时, 反转方向
17
18             // 如果是反向填充, 调整行号
19             if (reverse) k = n - 1 - k;
20
21             matrix.get(k).add(str.charAt(i));
22         }
23
24         // 输出矩阵内容
25         for (List<Character> row : matrix) {
26             for (char c : row) {
27                 System.out.print(c);
28             }
29             System.out.println(); // 输出换行
30         }
31     }
32
33     public static void main(String[] args) {
34         Scanner scanner = new Scanner(System.in);
35         String str = scanner.next();
36         int n = scanner.nextInt();
37         zigzagFill(str, n);
38     }
39 }
40 }
```

Python


```
1  # 按照之字形方式填充矩阵并输出
2  def zigzag_fill(s, n):
3      # 创建矩阵，每行是一个列表
4      matrix = [[] for _ in range(n)]
5
6      reverse = True # 控制方向
7      for i in range(len(s)):
8          k = i % n # 计算当前字符所在的行号
9          if k == 0:
10             reverse = not reverse # 每当到达行的起点时，反转方向
11
12             # 如果是反向填充，调整行号
13             if reverse:
14                 k = n - 1 - k
15
16             matrix[k].append(s[i])
17
18     # 输出矩阵内容
19     for row in matrix:
20         for c in row:
21             print(c, end="")
22         print() # 输出换行
23
24
25 if __name__ == "__main__":
26     s, n = input().split()
27     n = int(n)
28     zigzag_fill(s, n)
```

JavaScript

```
1  /* JavaScript Node ACM模式 控制台输入获取 */
2  const readline = require('readline');
3
4  // 创建接口来读取输入
5  const rl = readline.createInterface({
6      input: process.stdin,
7      output: process.stdout
8  });
9
10 // 按照之字形方式填充矩阵并输出
11 function zigzagFill(str, n) {
12     // 创建一个二维数组表示矩阵，每行是一个数组
13     let matrix = Array.from({ length: n }, () => []);
14
15
16     let reverse = true; // 控制填充方向，开始时是正向
17
18     // 遍历字符串中的每个字符，填充到对应的矩阵行中
19     for (let i = 0; i < str.length; i++) {
20         let k = i % n;
21         if (k == 0) {
22             reverse = !reverse;
23         }
24         if (reverse) {
25             k = n - 1 - k;
26         }
27         matrix[k].push(str[i]);
28     }
29
30     // 输出矩阵内容
31     for (let i = 0; i < n; i++) {
32         console.log(matrix[i].join('')); // 输出当前行的内容
33     }
34 }
35
36 // 读取输入
37 rl.question('', (input) => {
38     const strAndN = input.split(' '); // 读取一行输入并分割
39     const str = strAndN[0]; // 获取字符串部分
40     const n = parseInt(strAndN[1], 10); // 获取行数部分
41
42     zigzagFill(str, n);
43     rl.close();
44 });
```

Go

```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "strings"
8 )
9
10 // 按照之字形方式填充矩阵并输出
11 func zigzagFill(str string, n int) {
12     // 如果 n 小于等于 0, 直接返回
13     if n <= 0 {
14         fmt.Println("Invalid input: number of rows must be greater than 0")
15         return
16     }
17
18     // 创建矩阵, 每行是一个切片
19     matrix := make([][]rune, n)
20
21     reverse := true // 控制方向
22     for i := 0; i < len(str); i++ {
23         k := i % n // 计算当前字符所在的行号
24         if k == 0 {
25             reverse = !reverse // 每当到达行的起点时, 反转方向
26         }
27
28         // 如果是反向填充, 调整行号
29         if reverse {
30             k = n - 1 - k
31         }
32
33         matrix[k] = append(matrix[k], rune(str[i]))
34     }
35
36     // 输出矩阵内容
37     for _, row := range matrix {
38         for _, c := range row {
39             fmt.Print(string(c))
40         }
41         fmt.Println() // 输出换行
42     }
43 }
44 }
45
```

```
46 func main() {
47     // 创建输入读取器
48     reader := bufio.NewReader(os.Stdin)
49
50     // 读取输入
51     input, _ := reader.ReadString('\n')
52     input = strings.TrimSpace(input)
53
54     // 分割输入，获取字符串和数字
55     var str string
56     var n int
57     fmt.Sscanf(input, "%s %d", &str, &n)
58
59     // 调用填充函数
60     zigzagFill(str, n)
61 }
```

来自: [华为OD机试 2025B卷 – 箱子Z字形摆放 \(C++ & Python & JAVA & JS & GO\)_箱子z字形摆放](#)
[华为od-CSDN博客](#)

华为OD机考2025B卷 - 符合条件的元组个数 (C++ & Python & JAVA & JS & GO)_华为od机考 2025a卷-CSDN博客

符合条件的元组个数

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

2025B卷 100分题型

题目描述

给定一个整数数组 `nums`、一个数字 `k`，一个整数目标值 `target`，请问 `nums` 中 [是否存在](#) `k` 个元素使得其相加结果为 `target`，请输出所有符合条件且不重复的 `k` 元组的个数

数据范围如下：

- $2 \leq \text{nums.length} \leq 200$
- $-10^9 \leq \text{nums}[i] \leq 10^9$
- $-10^9 \leq \text{target} \leq 10^9$
- $2 \leq k \leq 100$

输入描述

第一行是 `nums` 取值：2 7 11 15

第二行是 `k` 的取值：2

第三行是 `target` 取值：9

输出描述

输出第一行是符合要求的元组个数：1

补充说明：[2,7]满足，输出个数是1

用例1

输入

```
1  -1 0 1 2 -1 -4
2  3
3  0
```

Plain Text

输出

▼		Plain Text
1	2	

说明

[-1,0,1], [-1,-1,2]满足条件

用例2

输入

▼		Plain Text
1	2 7 11 15	
2	2	
3	9	

输出

▼		Plain Text
1	1	

说明

[2,7]符合条件

题解

思路：两数之和/三数之和 变形题。

- 三数之和解决方式是将固定一个值，转换为两数之和。类比，k数之和，可以先固定k-2个数然后使用两数之和解决就行。
- 使用 递归回溯 方案尝试不同组合固定住k-1个元素，之后使用 两数之和 方案在剩余数字中寻找两个元素满足整体和为tareget的情况。
- 注意点
 - 注意不允许重复(如果两个组合所有元素都相同认作重复)
 - 剪枝操作，数据量还是比较大的。(机考时测试案例数据量比较小)

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 #include<set>
12 using namespace std;
13
14 // 通用 split 函数
15 vector<int> split(const string& str, const string& delimiter) {
16     vector<int> result;
17     size_t start = 0;
18     size_t end = str.find(delimiter);
19     while (end != string::npos) {
20         result.push_back(stoi(str.substr(start, end - start)));
21         start = end + delimiter.length();
22         end = str.find(delimiter, start);
23     }
24     // 添加最后一个部分
25     result.push_back(stoi(str.substr(start)));
26     return result;
27 }
28
29 int twoSumsolve(vector<int> ans, int target, int index, long sum) {
30     int l = index;
31     int r = ans.size() - 1;
32     // 记录成立的搭配方案
33     int res = 0;
34     while (l < r) {
35         long tmp = sum + ans[l] + ans[r];
36         if (tmp < target) {
37             l++;
38         } else if (tmp > target) {
39             r--;
40         } else {
41             res++;
42             // 去重
43             while (l + 1 < r && ans[l] == ans[l+1]) {
44                 l++;
45             }
46         }
47     }
48 }
```



```

46         // 去重
47         while (r - 1 > l && ans[r] == ans[r-1]) {
48             r--;
49         }
50         l++;
51         r--;
52     }
53 }
54 return res;
55 }
56
57
58 int solve(vector<int> ans, int k , int target, int index, long sum) {
59     int res = 0;
60     // 直接使用两数之和判断
61     if (k == 2) {
62         return twoSumsolve(ans, target, index, sum);
63     }
64
65     for (int i = index; i <= ans.size() - k; i++) {
66         // 剪枝
67         if (ans[i] > 0 && sum + ans[i] > target) {
68             break;
69         }
70         // 防止重复
71         if (i > index && ans[i] == ans[i-1]) {
72             continue;
73         }
74         res += solve(ans, k - 1, target, i + 1, sum + ans[i]);
75     }
76     return res;
77 }
78
79
80 int main() {
81     string input;
82     getline(cin, input);
83
84     int k;
85     cin >> k;
86     int target;
87     cin >> target;
88     vector<int> ans = split(input, " ");
89     int n = ans.size();
90     if (n < k) {
91         cout << 0;
92         return 0;
93     }

```

```
94     sort(ans.begin(), ans.end());
95     int res = solve(ans, k, target, 0, 0);
96     cout << res;
97     return 0;
98 }
```

JAVA

```
1  import java.util.*;
2
3  public class Main {
4
5      // 两数之和逻辑
6      static int twoSumSolve(List<Integer> nums, int target, int index, long sum) {
7          int l = index;
8          int r = nums.size() - 1;
9          int res = 0;
10         while (l < r) {
11             long tmp = sum + nums.get(l) + nums.get(r);
12             if (tmp < target) {
13                 l++;
14             } else if (tmp > target) {
15                 r--;
16             } else {
17                 res++;
18                 int left = nums.get(l), right = nums.get(r);
19                 while (l < r && nums.get(l).equals(left)) l++;
20                 while (l < r && nums.get(r).equals(right)) r--;
21             }
22         }
23         return res;
24     }
25
26     // k 数之和递归
27     static int solve(List<Integer> nums, int k, int target, int index, long sum) {
28         if (k == 2) {
29             return twoSumSolve(nums, target, index, sum);
30         }
31         int res = 0;
32         for (int i = index; i <= nums.size() - k; i++) {
33             if (nums.get(i) > 0 && sum + nums.get(i) > target) break;
34             if (i > index && nums.get(i).equals(nums.get(i - 1))) continue;
35             res += solve(nums, k - 1, target, i + 1, sum + nums.get(i));
36         }
37         return res;
38     }
39
40     public static void main(String[] args) {
41         Scanner sc = new Scanner(System.in);
42         String[] strNums = sc.nextLine().split(" ");
```

```

43         int k = sc.nextInt();
44         int target = sc.nextInt();
45
46         List<Integer> nums = new ArrayList<>();
47         for (String s : strNums) {
48             nums.add(Integer.parseInt(s));
49         }
50
51         if (nums.size() < k) {
52             System.out.println(0);
53             return;
54         }
55
56         Collections.sort(nums);
57         System.out.println(solve(nums, k, target, 0, 0));
58     }
59 }

```

Python

```
1 def two_sum_solve(nums, target, index, total):
2     # 两数之和：双指针处理
3     l, r = index, len(nums) - 1
4     res = 0
5     while l < r:
6         tmp = total + nums[l] + nums[r]
7         if tmp < target:
8             l += 1
9         elif tmp > target:
10            r -= 1
11        else:
12            res += 1
13            # 去重
14            left, right = nums[l], nums[r]
15            while l < r and nums[l] == left:
16                l += 1
17            while l < r and nums[r] == right:
18                r -= 1
19    return res
20
21 def solve(nums, k, target, index, total):
22     if k == 2:
23         return two_sum_solve(nums, target, index, total)
24     res = 0
25     for i in range(index, len(nums) - k + 1):
26         # 剪枝
27         if nums[i] > 0 and total + nums[i] > target:
28             break
29         # 跳过重复
30         if i > index and nums[i] == nums[i - 1]:
31             continue
32         res += solve(nums, k - 1, target, i + 1, total + nums[i])
33     return res
34
35 def main():
36     # 第一行：数组
37     nums = list(map(int, input().strip().split()))
38     k = int(input()) # 第二行：k
39     target = int(input()) # 第三行：目标值
40
41     if len(nums) < k:
42         print(0)
43         return
44
45     nums.sort() # 排序方便剪枝
```

```
46     print(solve(nums, k, target, 0, 0))
47
48 if __name__ == '__main__':
49     main()
```

JavaScript

```
1  const readline = require('readline');
2
3  const rl = readline.createInterface({
4    input: process.stdin,
5    output: process.stdout
6  });
7
8  let lines = [];
9  rl.on('line', function(line) {
10    lines.push(line.trim());
11    if (lines.length === 3) {
12      main();
13    }
14  });
15
16  // 两数之和逻辑：双指针+去重
17  function twoSumSolve(nums, target, index, total) {
18    let l = index, r = nums.length - 1;
19    let res = 0;
20    while (l < r) {
21      let tmp = total + nums[l] + nums[r];
22      if (tmp < target) {
23        l++;
24      } else if (tmp > target) {
25        r--;
26      } else {
27        res++;
28        let left = nums[l], right = nums[r];
29        while (l < r && nums[l] === left) l++;
30        while (l < r && nums[r] === right) r--;
31      }
32    }
33    return res;
34  }
35
36  // k 数之和递归
37  function solve(nums, k, target, index, total) {
38    if (k === 2) return twoSumSolve(nums, target, index, total);
39    let res = 0;
40    for (let i = index; i <= nums.length - k; i++) {
41      if (nums[i] > 0 && total + nums[i] > target) break;
42      if (i > index && nums[i] === nums[i - 1]) continue;
43      res += solve(nums, k - 1, target, i + 1, total + nums[i]);
44    }
45    return res;
```

```
46 }  
47  
48 // 主函数处理输入  
49 function main() {  
50     const nums = lines[0].split(' ').map(Number);  
51     const k = parseInt(lines[1]);  
52     const target = parseInt(lines[2]);  
53  
54     if (nums.length < k) {  
55         console.log(0);  
56         return;  
57     }  
58  
59     nums.sort((a, b) => a - b);  
60     console.log(solve(nums, k, target, 0, 0));  
61 }
```

Go


```
1 package main
2
3 import (
4     "bufio"
5     "fmt"
6     "os"
7     "sort"
8     "strconv"
9     "strings"
10 )
11
12 // twoSumSolve 实现两数之和的逻辑，使用双指针并去重
13 func twoSumSolve(nums []int, target int, index int, sum int) int {
14     left := index
15     right := len(nums) - 1
16     count := 0
17
18     for left < right {
19         total := sum + nums[left] + nums[right]
20         if total < target {
21             left++
22         } else if total > target {
23             right--
24         } else {
25             count++
26             // 去重处理
27             lv := nums[left]
28             rv := nums[right]
29             for left < right && nums[left] == lv {
30                 left++
31             }
32             for left < right && nums[right] == rv {
33                 right--
34             }
35         }
36     }
37
38     return count
39 }
40
41 // solve 实现 k 数之和的递归逻辑
42 func solve(nums []int, k int, target int, index int, sum int) int {
43     if k == 2 {
44         return twoSumSolve(nums, target, index, sum)
45     }
```

```

46
47     count := 0
48     for i := index; i <= len(nums)-k; i++ {
49         // 剪枝: 如果当前数加上 sum 已经超过目标, 后续不可能满足
50         if i > index && nums[i] == nums[i-1] {
51             continue // 去重
52         }
53         // 进一步剪枝优化 (因为数组已排序)
54         if nums[i] > 0 && sum+nums[i] > target {
55             break
56         }
57         count += solve(nums, k-1, target, i+1, sum+nums[i])
58     }
59     return count
60 }
61
62 func main() {
63     scanner := bufio.NewScanner(os.Stdin)
64
65     // 读取第一行: 一行空格分隔的整数
66     scanner.Scan()
67     line := scanner.Text()
68     parts := strings.Fields(line)
69
70     nums := make([]int, 0, len(parts))
71     for _, s := range parts {
72         num, _ := strconv.Atoi(s)
73         nums = append(nums, num)
74     }
75
76     // 读取第二行: k
77     scanner.Scan()
78     k, _ := strconv.Atoi(scanner.Text())
79
80     // 读取第三行: target
81     scanner.Scan()
82     target, _ := strconv.Atoi(scanner.Text())
83
84     if len(nums) < k {
85         fmt.Println(0)
86         return
87     }
88
89     sort.Ints(nums)
90     result := solve(nums, k, target, 0, 0)
91     fmt.Println(result)
92 }

```

来自: [华为OD机考2025B卷 – 符合条件的元组个数 \(C++ & Python & JAVA & JS & GO\)_华为od机考2025a卷-CSDN博客](#)

华为OD机试2025B卷 - MVP争夺战 (C++ & Python & JAVA & JS & GO)-CSDN博客

华为OD机试真题目录: [点击查看](#)

华为OD机试2025B卷 100分题型

题目描述

在星球争霸篮球赛对抗赛中，最大的宇宙战队希望每个人都能拿到MVP，MVP的条件是单场最高分得分获得者，可以并列。所以宇宙战队决定在比赛中尽可能让更多队员上场，并且让所有得分的选手得分都相同，然而比赛过程中的每1分钟的得分都只能由某一个人包揽。

输入描述

输入第一行为一个数字 t ，表示为有得分的分钟数 $1 \leq t \leq 50$ 。
第二行为 t 个数字，代表每一分钟的得分 p ， $1 \leq p \leq 50$ 。

输出描述

输出有得分的队员都是MVP时，最少得MVP得分。

用例1

输入

▼	Plain Text
1 9	
2 5 2 1 5 2 1 5 2 1	

输出

▼	Plain Text
1 6	

说明

样例解释 一共 4 人得分，分别都是 6 分 $5 + 1$ ， $5 + 1$ ， $5 + 1$ ， $2 + 2 + 2$

题解

思路：递归回溯

1. 暴力枚举允许获得MVP人数可能的情况,可能的情况为 $[1, t]$.
2. 根据1的枚举的MVP人数 x , 判断是否可以满足 指定人数的MVP情况, 不满足则将 $x - 1$, 继续进行判断, 判断过程逻辑如下:
 - a. 首先判断 $\text{sum}(\text{score}) \% x == 0$, 不满足直接返回false.
 - b. 判断 $\text{max}(\text{score}) > (\text{sum}(\text{score}) / x)$, 每一场分数不能进行拆分, 肯定不满足所有人都拿 $\text{sum}(\text{score}) / x$ 的分数。
 - c. 使用 递归回溯 尝试不同组合方案, 判断是否能组合出 x 个 分数为 $\text{sum}(\text{score}) / x$ 的方案。其中涉及部门剪枝优化, 具体逻辑可参照代码注释。
3. 执行2的逻辑, 解决条件为 $x == 1$ 或者 在 $[1, t]$ 中可以找到一个合法情况 。输出最终的 x 就是答案。

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  bool cmp(int x , int y) {
10     return x > y;
11 }
12
13 bool backtrack(int index, int right, vector<int> & buckets, int subSum, vector<int> ans) {
14     if (index == right) {
15         for (int i = 0; i < buckets.size(); i++) {
16             if (buckets[i] != subSum) {
17                 return false;
18             }
19         }
20         return true;
21     }
22
23     for (int i = 0; i < buckets.size(); i++) {
24         // 剪枝 防止重复运算
25         if (i > 0 && buckets[i] == buckets[i - 1]) {
26             continue;
27         }
28
29         if (buckets[i] + ans[index] > subSum) {
30             continue;
31         }
32         // 递归
33         buckets[i] += ans[index];
34         if (backtrack(index + 1, right, buckets, subSum, ans)) {
35             return true;
36         }
37         // 回溯
38         buckets[i] -= ans[index];
39
40     }
41     return false;
42 }
43
44 bool canPartition(vector<int> ans, int n, int sum) {
```

```

45     //不能整除说明当前假设mvp人数下，肯定不成立
46     if (sum % n) {
47         return false;
48     }
49     // 为每个人得到的分数
50     int subSum = sum / n;
51
52     int size = ans.size();
53     // 分数不能进行分割，所以肯定不成立
54     if (ans[0] > subSum) {
55         return false;
56     }
57
58     int beginIndex = 0;
59     // 预处理数据，减少数据量，提高算法执行效率
60     while (beginIndex < size && ans[beginIndex] == subSum) {
61         n--;
62         beginIndex++;
63     }
64
65     if (n == 0) {
66         return true;
67     }
68
69     vector<int> buckets(n, 0);
70     return backtrack(beginIndex, size, buckets, subSum, ans);
71 }
72
73
74
75
76
77 int main() {
78     int n ;
79     cin >> n;
80     vector<int> ans(n);
81     // 所有得分
82     int sum = 0;
83     for (int i = 0; i < n; i++) {
84         cin >> ans[i];
85         sum += ans[i];
86     }
87     // 将分数进行逆序排序
88     sort(ans.begin(), ans.end(), cmp);
89     // 从大到小枚举获取MVP的人数， 大小范围为 1 - m
90     while (n != 1) {
91         if (canPartition(ans, n, sum)) {
92             break;

```

```
93         }  
94         n--;  
95     }  
96     cout << sum / n;  
97     return 0;  
98 }
```

JAVA


```
1  import java.util.*;
2
3  public class Main {
4
5      // 比较函数, 用于降序排序
6      public static boolean cmp(int x, int y) {
7          return x > y;
8      }
9
10     // 回溯函数
11     public static boolean backtrack(int index, int right, int[] buckets, int subSum, List<Integer> ans) {
12         if (index == right) {
13             for (int bucket : buckets) {
14                 if (bucket != subSum) {
15                     return false;
16                 }
17             }
18             return true;
19         }
20
21         for (int i = 0; i < buckets.length; i++) {
22             // 剪枝: 防止重复运算
23             if (i > 0 && buckets[i] == buckets[i - 1]) {
24                 continue;
25             }
26
27             if (buckets[i] + ans.get(index) > subSum) {
28                 continue;
29             }
30             // 递归
31             buckets[i] += ans.get(index);
32             if (backtrack(index + 1, right, buckets, subSum, ans)) {
33                 return true;
34             }
35             // 回溯
36             buckets[i] -= ans.get(index);
37         }
38         return false;
39     }
40
41     // 判断是否可以分配
42     public static boolean canPartition(List<Integer> ans, int n, int sum)
43     {
44         if (sum % n != 0) {
```

```

44         return false;
45     }
46     int subSum = sum / n;
47     if (ans.get(0) > subSum) {
48         return false;
49     }
50
51     int beginIndex = 0;
52     // 预处理数据, 减少数据量
53     while (beginIndex < ans.size() && ans.get(beginIndex) == subSum) {
54         n--;
55         beginIndex++;
56     }
57
58     if (n == 0) {
59         return true;
60     }
61
62     int[] buckets = new int[n];
63     return backtrack(beginIndex, ans.size(), buckets, subSum, ans);
64 }
65
66 public static void main(String[] args) {
67     Scanner scanner = new Scanner(System.in);
68     int n = scanner.nextInt();
69     List<Integer> ans = new ArrayList<>();
70     int sum = 0;
71     for (int i = 0; i < n; i++) {
72         int score = scanner.nextInt();
73         ans.add(score);
74         sum += score;
75     }
76
77     // 将分数按降序排列
78     ans.sort((x, y) -> y - x);
79
80     // 从大到小枚举获取MVP人数, 范围为 1 - m
81     while (n != 1) {
82         if (canPartition(ans, n, sum)) {
83             break;
84         }
85         n--;
86     }
87     System.out.println(sum / n);
88 }
89 }

```



```
1 def cmp(x, y):
2     return x > y
3
4 def backtrack(index, right, buckets, sub_sum, ans, memo):
5     if index == right:
6         # 所有桶的值都已经达到 sub_sum
7         return all(bucket == sub_sum for bucket in buckets)
8
9     # 使用 memo 来缓存已经尝试过的状态
10    state = tuple(buckets)
11    if state in memo:
12        return False
13    memo.add(state)
14
15    for i in range(len(buckets)):
16        # 剪枝: 避免重复运算
17        if i > 0 and buckets[i] == buckets[i - 1]:
18            continue
19
20        if buckets[i] + ans[index] > sub_sum:
21            continue
22
23        # 递归
24        buckets[i] += ans[index]
25        if backtrack(index + 1, right, buckets, sub_sum, ans, memo):
26            return True
27        # 回溯
28        buckets[i] -= ans[index]
29
30    return False
31
32 def can_partition(ans, n, total_sum):
33     if total_sum % n != 0:
34         return False
35
36     sub_sum = total_sum // n
37     if ans[0] > sub_sum:
38         return False
39
40     begin_index = 0
41     while begin_index < len(ans) and ans[begin_index] == sub_sum:
42         n -= 1
43         begin_index += 1
44
45     if n == 0:
```

```

46         return True
47
48     buckets = [0] * n
49     memo = set() # 用于缓存已经遍历的桶状态
50     return backtrack(begin_index, len(ans), buckets, sub_sum, ans, memo)
51
52 def main():
53     n = int(input())
54     ans = list(map(int, input().split()))
55     total_sum = sum(ans)
56
57     # 降序排序
58     ans.sort(reverse=True)
59
60     # 从大到小枚举获取MVP人数
61     while n != 1:
62         if can_partition(ans, n, total_sum):
63             break
64         n -= 1
65
66     print(total_sum // n)
67
68 if __name__ == "__main__":
69     main()

```

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 === 2) {
13      const n = parseInt(lines[0], 10);
14      const arr = lines[1].split(" ").map(Number);
15
16      console.log(findMaxAverage(arr, n));
17      lines.length = 0;
18    }
19  });
20
21  function findMaxAverage(arr, n) {
22    // 计算总和并从大到小排序
23    const sum = arr.sort((a, b) => b - a).reduce((acc, val) => acc + val,
24    0);
25
26    let count = n;
27    // 从n开始枚举MVP人数
28    while (count >= 1) {
29      if (isPartitionPossible([...arr], sum, count)) {
30        return sum / count;
31      }
32      count--;
33    }
34  }
35
36  function isPartitionPossible(arr, sum, count) {
37    // 如果总和不能均分
38    if (sum % count !== 0) return false;
39
40    const subSum = sum / count;
41    // 如果最大的分数超过每个人应该得到的分数, 直接返回false
42    if (subSum < arr[0]) return false;
43
44    // 剔除已经等于subSum的值, 减少数据量
45    while (arr[0] === subSum) {
```

```

45     arr.shift();
46     count--;
47 }
48
49     const buckets = new Array(count).fill(0);
50     return canPartitionRecursively(0, arr, subSum, buckets);
51 }
52
53 function canPartitionRecursively(index, arr, subSum, buckets) {
54     if (index === arr.length) {
55         // 所有元素都已经分配完
56         return true;
57     }
58
59     const current = arr[index];
60
61     for (let i = 0; i < buckets.length; i++) {
62         // 剪枝: 避免重复运算
63         if (i > 0 && buckets[i] === buckets[i - 1]) continue;
64
65         // 如果当前桶加入当前元素没有超过目标分数
66         if (buckets[i] + current <= subSum) {
67             buckets[i] += current;
68             if (canPartitionRecursively(index + 1, arr, subSum, buckets)) return true;
69             // 回溯
70             buckets[i] -= current;
71         }
72     }
73
74     return false;
75 }

```

Go

```
1  package main
2
3  import (
4      "fmt"
5      "sort"
6  )
7
8  func cmp(x, y int) bool {
9      return x > y
10 }
11
12 func backtrack(index, right int, buckets []int, subSum int, ans []int) bool {
13     if index == right {
14         for _, bucket := range buckets {
15             if bucket != subSum {
16                 return false
17             }
18         }
19         return true
20     }
21
22     for i := 0; i < len(buckets); i++ {
23         // 剪枝, 避免重复运算
24         if i > 0 && buckets[i] == buckets[i-1] {
25             continue
26         }
27
28         if buckets[i]+ans[index] > subSum {
29             continue
30         }
31
32         // 递归
33         buckets[i] += ans[index]
34         if backtrack(index+1, right, buckets, subSum, ans) {
35             return true
36         }
37         // 回溯
38         buckets[i] -= ans[index]
39     }
40
41     return false
42 }
43
44 func canPartition(ans []int, n, totalSum int) bool {
```



```

45 // 如果不能整除, 直接返回 false
46 if totalSum%n != 0 {
47     return false
48 }
49
50 // 每个人应该得到的分数
51 subSum := totalSum / n
52 size := len(ans)
53
54 // 如果最大的得分大于 subSum, 直接返回 false
55 if ans[0] > subSum {
56     return false
57 }
58
59 beginIndex := 0
60
61 // 预处理数据, 减少计算量
62 for beginIndex < size && ans[beginIndex] == subSum {
63     n--
64     beginIndex++
65 }
66
67 if n == 0 {
68     return true
69 }
70
71 buckets := make([]int, n)
72 return backtrack(beginIndex, size, buckets, subSum, ans)
73 }
74
75 func main() {
76     var n int
77     fmt.Scan(&n) // 输入人数
78     ans := make([]int, n)
79     totalSum := 0
80
81     // 输入所有的得分并计算总分
82     for i := 0; i < n; i++ {
83         fmt.Scan(&ans[i])
84         totalSum += ans[i]
85     }
86
87     // 将得分按降序排序
88     sort.Slice(ans, func(i, j int) bool {
89         return cmp(ans[i], ans[j])
90     })
91
92     // 从大到小枚举获取 MVP 的人数

```

```
93     for n != 1 {
94         if canPartition(ans, n, totalSum) {
95             break
96         }
97         n--
98     }
99
100     // 输出每个人的分数
101     fmt.Println(totalSum / n)
102 }
```

来自: [华为OD机试2025B卷 – MVP争夺战 \(C++ & Python & JAVA & JS & GO\)–CSDN博客](#)