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<=N<=10^3
第二行为N个用空格分开的整数 Ci，-10^5 <= Ci <= 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

```
10
8 8 9 1 9 6 3 9 1 0
```

输出

Plain Text

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

输出

```Plain Text
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`coor[i].start < lastEnd`，则`count += 1, lastEnd = coor[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++

Plain Text

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

// 计算不重叠区间数量
int calNotJoint(vector<pair<int, int>>& coor) {
    if (coor.empty()) {
        return 0;
    }
    int count = 1;
    // 结束时间升序
    sort(coor.begin(), coor.end(), [](const pair<int, int>& a, const pair<int, int>& b){
        return a.second < b.second;  // 结束时间升序
    });
    int lastEnd = coor[0].second;
    int n  = coor.size();
    for (int i = 1; i < n; i++) {
        int start = coor[i].first;
        int end = coor[i].second;
        if (lastEnd < start) {
            count++;
            lastEnd = end;
        }
    }
    return count;
}


int main() {
    int n;
    cin >> n;
    vector<int> nums(n);
    for (int i = 0; i < n; i++) {
        cin >>nums[i];
    }

    vector<int> prefix(n + 1, 0);

    // 计算前缀和
```

```
    for (int i = 1; i <= n; i++) {
        prefix[i] = prefix[i-1] + nums[i - 1];
    }
    // key 区间和 => 区间(起始坐标,结束坐标)
    map<int, vector<pair<int, int>>> sumCoorMp;
    for (int i = 1; i <= n; i++) {
        for (int j = i; j <= n; j++) {
            int sum = prefix[j] - prefix[i-1];
            sumCoorMp[sum].push_back({i, j});
        }
    }

    // 求出其中最大不重叠数量
    int res = 0;
    for (auto p : sumCoorMp) {
       vector<pair<int, int>> coor = p.second;
       res = max(res, calNotJoint(coor));
    }
    cout << res;
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 计算不重叠区间数量
    static int calNotJoint(List<int[]> coor) {
        if (coor.isEmpty()) return 0;
        // 按结束时间升序排列
        coor.sort(Comparator.comparingInt(a -> a[1]));
        int count = 1;
        int lastEnd = coor.get(0)[1];
        for (int i = 1; i < coor.size(); i++) {
            int start = coor.get(i)[0];
            int end = coor.get(i)[1];
            if (lastEnd < start) {
                count++;
                lastEnd = end;
            }
        }
        return count;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int[] nums = new int[n];
        for (int i = 0; i < n; i++) nums[i] = sc.nextInt();

        int[] prefix = new int[n + 1];
        // 前缀和构建
        for (int i = 1; i <= n; i++) {
            prefix[i] = prefix[i - 1] + nums[i - 1];
        }

        // key: 区间和 -> 多个坐标区间
        Map<Integer, List<int[]>> sumCoorMap = new HashMap<>();
        for (int i = 1; i <= n; i++) {
            for (int j = i; j <= n; j++) {
                int sum = prefix[j] - prefix[i - 1];
                sumCoorMap.computeIfAbsent(sum, k -> new ArrayList<>()).ad
d(new int[]{i, j});
            }
        }

        int res = 0;
        for (List<int[]> coor : sumCoorMap.values()) {
```

```
                res = Math.max(res, calNotJoint(coor));
            }
            System.out.println(res);
        }
    }
```

Python

Plain Text

```
def cal_not_joint(coor):
    if not coor:
        return 0
    # 按结束位置升序排序
    coor.sort(key=lambda x: x[1])
    count = 1
    last_end = coor[0][1]
    for i in range(1, len(coor)):
        start, end = coor[i]
        if last_end < start:
            count += 1
            last_end = end
    return count

def main():
    n = int(input())
    nums = list(map(int, input().split()))
    prefix = [0] * (n + 1)

    # 计算前缀和
    for i in range(1, n + 1):
        prefix[i] = prefix[i - 1] + nums[i - 1]

    # 和 -> 区间列表
    from collections import defaultdict
    sum_coor = defaultdict(list)

    for i in range(1, n + 1):
        for j in range(i, n + 1):
            s = prefix[j] - prefix[i - 1]
            sum_coor[s].append((i, j))

    res = 0
    for coor in sum_coor.values():
        res = max(res, cal_not_joint(coor))
    print(res)

main()
```

JavaScript

Plain Text

```
// 引入 readline 用于逐行读取输入
const readline = require('readline');

// 创建输入接口
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const inputLines = [];
rl.on('line', line => {
    inputLines.push(line.trim());
});

rl.on('close', () => {
    main(inputLines);
});

// 计算不重叠区间数量（结束时间升序的贪心算法）
function calNotJoint(coor) {
    if (coor.length === 0) return 0;

    // 按结束位置升序排序
    coor.sort((a, b) => a[1] - b[1]);

    let count = 1;
    let lastEnd = coor[0][1];

    for (let i = 1; i < coor.length; i++) {
        const [start, end] = coor[i];
        if (lastEnd < start) {
            count++;
            lastEnd = end;
        }
    }

    return count;
}

function main(lines) {
    const n = parseInt(lines[0]);
    const nums = lines[1].split(' ').map(Number);

    // 构造前缀和数组
    const prefix = new Array(n + 1).fill(0);
```

```
    for (let i = 1; i <= n; i++) {
        prefix[i] = prefix[i - 1] + nums[i - 1];
    }

    // sum -> list of intervals (1-based index)
    const sumCoorMap = new Map();

    // 枚举所有区间，记录区间和
    for (let i = 1; i <= n; i++) {
        for (let j = i; j <= n; j++) {
            const sum = prefix[j] - prefix[i - 1];
            if (!sumCoorMap.has(sum)) {
                sumCoorMap.set(sum, []);
            }
            sumCoorMap.get(sum).push([i, j]);
        }
    }

    // 对所有和的区间进行不重叠计数，求最大值
    let res = 0;
    for (let coor of sumCoorMap.values()) {
        res = Math.max(res, calNotJoint(coor));
    }

    console.log(res);
}
```

Go

Plain Text

```
package main

import (
  "fmt"
  "sort"
)

func calNotJoint(coor [][2]int) int {
  if len(coor) == 0 {
    return 0
  }
  // 按结束时间升序排序
  sort.Slice(coor, func(i, j int) bool {
    return coor[i][1] < coor[j][1]
  })

  count := 1
  lastEnd := coor[0][1]
  for i := 1; i < len(coor); i++ {
    start := coor[i][0]
    end := coor[i][1]
    if lastEnd < start {
      count++
      lastEnd = end
    }
  }
  return count
}

func main() {
  var n int
  fmt.Scan(&n)
  nums := make([]int, n)
  for i := 0; i < n; i++ {
    fmt.Scan(&nums[i])
  }
    // 计算前缀和
  prefix := make([]int, n+1)
  for i := 1; i <= n; i++ {
    prefix[i] = prefix[i-1] + nums[i-1]
  }
    // key: 区间和 -> 多个坐标区间
  sumCoor := make(map[int][][2]int)

  for i := 1; i <= n; i++ {
```

```
      for j := i; j <= n; j++ {
        sum := prefix[j] - prefix[i-1]
        sumCoor[sum] = append(sumCoor[sum], [2]int{i, j})
      }
    }

    res := 0
    for _, coor := range sumCoor {
      tmp := calNotJoint(coor)
      if tmp > res {
        res = tmp
      }
    }
    fmt.Println(res)
}
```

> 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 ⩽ l ⩽ r ⩽ 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

```
1 1 10
```

输出

```
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

```
10 20
```

输出

```
7
```

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++

Plain Text

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

#define MAX_LEN 32

int dp[MAX_LEN][2][2];
int binary[MAX_LEN];

// 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
int dfs(int pos, int limit, int pre, int prepre) {
    if (pos == -1) return 1;  // 递归结束，找到一个有效方案
    // 利用记忆化数组缓存直接范围
    if (!limit && dp[pos][pre][prepre] != -1) return dp[pos][pre][prepre];
    // 之前选择为最大值，
    int maxDigit = limit ? binary[pos] : 1;
    int count = 0;

    for (int i = 0; i <= maxDigit; i++) {
        if (i == 1 && pre == 0 && prepre == 1) continue;  // 跳过非法情况
        // 递归
        count += dfs(pos - 1, limit && (i == maxDigit), i, pre);
    }
    // 记忆化
    if (!limit) dp[pos][pre][prepre] = count;
    return count;
}

// 计算 0~num 内的合法数字个数
int digitSearch(int num) {
    int len = 0;
    memset(dp, -1, sizeof(dp));

    for (int i = 0; i < MAX_LEN; i++) binary[i] = 0;

    // 存储二进制位 用于前一位为最大
    for (; num; num >>= 1) binary[len++] = num & 1;

    return dfs(len - 1, 1, 0, 0);
}

int main() {
    int L, R;
    cin >> L >> R;
```

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

JAVA

Plain Text

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

public class Main {
    static final int MAX_LEN = 32;
    static int[][][] dp = new int[MAX_LEN][2][2];
    static int[] binary = new int[MAX_LEN];

   // 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
    static int dfs(int pos, int limit, int pre, int prepre) {
        if (pos == -1) return 1; // 递归结束，找到一个有效方案
         // 利用记忆化数组缓存直接范围
        if (limit == 0 && dp[pos][pre][prepre] != -1) return dp[pos][pre][prepre];

        int maxDigit = limit == 1 ? binary[pos] : 1;
        int count = 0;

        for (int i = 0; i <= maxDigit; i++) {
            if (i == 1 && pre == 0 && prepre == 1) continue; // 跳过非法情况
            count += dfs(pos - 1, limit == 1 && i == maxDigit ? 1 : 0, i, pre);
        }

        if (limit == 0) dp[pos][pre][prepre] = count;
        return count;
    }

    // 计算 0~num 内的合法数字个数
    static int digitSearch(int num) {
        int len = 0;
        for (int[][] layer : dp) for (int[] row : layer) Arrays.fill(row, -1);

        Arrays.fill(binary, 0);

        while (num > 0) {
            binary[len++] = num & 1;
            num >>= 1;
        }

        return dfs(len - 1, 1, 0, 0);
    }

```

```
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int L = scanner.nextInt();
        int R = scanner.nextInt();
        System.out.println(digitSearch(R) - digitSearch(L - 1));
        scanner.close();
    }
}
```

Python

Plain Text

```
import sys

MAX_LEN = 32
dp = [[[-1] * 2 for _ in range(2)] for _ in range(MAX_LEN)]
binary = [0] * MAX_LEN

# 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
def dfs(pos, limit, pre, prepre):
    if pos == -1:
        return 1  # 递归结束，找到一个有效方案
    # 利用记忆化数组缓存直接范围
    if not limit and dp[pos][pre][prepre] != -1:
        return dp[pos][pre][prepre]

    maxDigit = binary[pos] if limit else 1
    count = 0

    for i in range(maxDigit + 1):
        if i == 1 and pre == 0 and prepre == 1:
            continue  # 跳过非法情况
        count += dfs(pos - 1, limit and (i == maxDigit), i, pre)

    if not limit:
        dp[pos][pre][prepre] = count
    return count

# 计算 0~num 内的合法数字个数
def digitSearch(num):
    global dp, binary
    for i in range(MAX_LEN):
        for j in range(2):
            for k in range(2):
                dp[i][j][k] = -1

    binary = [0] * MAX_LEN
    len_ = 0

    while num:
        binary[len_] = num & 1
        num >>= 1
        len_ += 1

    return dfs(len_ - 1, 1, 0, 0)

```

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

JavaScript

Plain Text

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

const MAX_LEN = 32;
let dp = Array.from({ length: MAX_LEN }, () =>
    Array.from({ length: 2 }, () => Array(2).fill(-1))
);
let binary = new Array(MAX_LEN).fill(0);

//  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
function dfs(pos, limit, pre, prepre) {
    if (pos === -1) return 1; // 递归结束，找到一个有效方案

    if (!limit && dp[pos][pre][prepre] !== -1) return dp[pos][pre][prepre];
     // 利用记忆化数组缓存直接范围
    let maxDigit = limit ? binary[pos] : 1;
    let count = 0;

    for (let i = 0; i <= maxDigit; i++) {
        if (i === 1 && pre === 0 && prepre === 1) continue; // 跳过非法情况
        count += dfs(pos - 1, limit && i === maxDigit, i, pre);
    }

    if (!limit) dp[pos][pre][prepre] = count;
    return count;
}

// 计算 0~num 内的合法数字个数
function digitSearch(num) {
    dp = Array.from({ length: MAX_LEN }, () =>
        Array.from({ length: 2 }, () => Array(2).fill(-1))
    );
    binary.fill(0);

    let len = 0;
    while (num > 0) {
        binary[len++] = num & 1;
        num >>= 1;
    }

    return dfs(len - 1, 1, 0, 0);
}

// 读取输入
```

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

rl.on("line", (line) => {
    let [L, R] = line.split(" ").map(Number);
    console.log(digitSearch(R) - digitSearch(L - 1));
    rl.close();
});
```

Go

Plain Text

```
package main

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

const MAX_LEN = 32

var dp [MAX_LEN][2][2]int
var binary [MAX_LEN]int

// 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
func dfs(pos, limit, pre, prepre int) int {
  if pos == -1 {
    return 1 // 递归结束，找到一个有效方案
  }
     // 利用记忆化数组缓存直接范围
  if limit == 0 && dp[pos][pre][prepre] != -1 {
    return dp[pos][pre][prepre]
  }

  maxDigit := 1
  if limit == 1 {
    maxDigit = binary[pos]
  }

  count := 0
  for i := 0; i <= maxDigit; i++ {
    if i == 1 && pre == 0 && prepre == 1 {
      continue // 跳过非法情况
    }
    newLimit := 0
    if limit == 1 && i == maxDigit {
      newLimit = 1
    }
    count += dfs(pos-1, newLimit, i, pre)
  }

  if limit == 0 {
    dp[pos][pre][prepre] = count
```

```
    }
    return count
}

// 计算 0~num 内的合法数字个数
func digitSearch(num int) int {
    for i := 0; i < MAX_LEN; i++ {
        for j := 0; j < 2; j++ {
            for k := 0; k < 2; k++ {
                dp[i][j][k] = -1
            }
        }
    }

    len := 0
    for num > 0 {
        binary[len] = num & 1
        num >>= 1
        len++
    }

    return dfs(len-1, 1, 0, 0)
}

func main() {
    reader := bufio.NewReader(os.Stdin)
    line, _ := reader.ReadString('\n')
    line = strings.TrimSpace(line)
    parts := strings.Split(line, " ")
    L, _ := strconv.Atoi(parts[0])
    R, _ := strconv.Atoi(parts[1])
    fmt.Println(digitSearch(R) - digitSearch(L-1))
}
```

> 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<=n <=50
- 0<=d<=100

第二行，n个队伍的实力值空格分割。

- 0<=各队伍实力值<=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

Plain Text

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

输出

```
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

```
6 20
81 87 47 59 81 18
```

输出

```
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++

Plain Text

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


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

int main() {
    int n,d;
    cin >> n >> d;
    // 处理换行符
    cin.ignore();

    string s;
    getline(cin, s);
    vector<string> ansStr = split(s, " ");
    vector<int> ans(n);
    for (int i = 0; i < n; i++) {
        ans[i] = stoi(ansStr[i]);
    }

    // 排序
    sort(ans.begin(), ans.end());
    //  dp[i] 代表i个队伍能匹配的队伍数量
    vector<int> dp(n+1, 0);
    //minSum[i] i个队伍匹配之后差值的和
    vector<int> minSum(n+1, 0);

```

```
    for (int i = 2; i <= n; i++) {

        bool canMatch = false;
        // 能否匹配
        if (ans[i-1] - ans[i-2] <= d) {
            canMatch = true;
        }
        if (canMatch) {
            if (dp[i-1]  < dp[i-2] + 1) {
                dp[i] = dp[i-2] + 1;
                minSum[i] = minSum[i-2] + ans[i-1] - ans[i-2];
            } else if (dp[i-1]  > dp[i-2] + 1) {
                dp[i] = dp[i-1];
                minSum[i] = minSum[i-1];
            } else {
                dp[i] = dp[i-1];
                minSum[i] = min(minSum[i-1], minSum[i-2] + ans[i-1] - ans
[i-2]);
            }
        } else {
            dp[i] = dp[i-1];
            minSum[i] = minSum[i-1];
        }


    }
    if (dp[n] == 0) {
        cout << -1;
    }else {
        cout << minSum[n];
    }
    return 0;
}
```

Java

Plain Text

```
import java.util.*;

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

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt(), d = scanner.nextInt();
        scanner.nextLine(); // 处理换行符

        String s = scanner.nextLine();
        List<String> ansStr = split(s, " ");
        int[] ans = new int[n];

        for (int i = 0; i < n; i++) {
            ans[i] = Integer.parseInt(ansStr.get(i));
        }

        // 排序
        Arrays.sort(ans);

        // dp[i] 代表 i 个队伍能匹配的队伍数量
        int[] dp = new int[n + 1];
        // minSum[i] i 个队伍匹配之后差值的和
        int[] minSum = new int[n + 1];

        for (int i = 2; i <= n; i++) {
            boolean canMatch = ans[i - 1] - ans[i - 2] <= d;

            if (canMatch) {
                // 优先选择匹配数量多的，匹配数量相等的选择差异值和小的
                if (dp[i - 1] < dp[i - 2] + 1) {
                    dp[i] = dp[i - 2] + 1;
                    minSum[i] = minSum[i - 2] + ans[i - 1] - ans[i - 2];
                } else if (dp[i - 1] > dp[i - 2] + 1) {
                    dp[i] = dp[i - 1];
                    minSum[i] = minSum[i - 1];
                } else {
                    dp[i] = dp[i - 1];
                    minSum[i] = Math.min(minSum[i - 1], minSum[i - 2] + an
s[i - 1] - ans[i - 2]);
                }
```

```
                } else {
                    dp[i] = dp[i - 1];
                    minSum[i] = minSum[i - 1];
                }
            }

            System.out.println(dp[n] == 0 ? -1 : minSum[n]);
        }
    }
```

Python

Plain Text

```
import sys

# 读取输入
n, d = map(int, sys.stdin.readline().split())
arr = list(map(int, sys.stdin.readline().split()))

# 排序
arr.sort()

# dp[i] 代表 i 个队伍能匹配的队伍数量
dp = [0] * (n + 1)
# minSum[i] i 个队伍匹配之后差值的和
minSum = [0] * (n + 1)

for i in range(2, n + 1):
    canMatch = arr[i - 1] - arr[i - 2] <= d
    # 优先选择匹配数量多的，匹配数量相等的选择差异值和小的
    if canMatch:
        if dp[i - 1] < dp[i - 2] + 1:
            dp[i] = dp[i - 2] + 1
            minSum[i] = minSum[i - 2] + arr[i - 1] - arr[i - 2]
        elif dp[i - 1] > dp[i - 2] + 1:
            dp[i] = dp[i - 1]
            minSum[i] = minSum[i - 1]
        else:
            dp[i] = dp[i - 1]
            minSum[i] = min(minSum[i - 1], minSum[i - 2] + arr[i - 1] - ar
r[i - 2])
    else:
        dp[i] = dp[i - 1]
        minSum[i] = minSum[i - 1]

print(-1 if dp[n] == 0 else minSum[n])
```

JavaScript

Plain Text

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

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

const inputLines = [];

rl.on("line", (line) => {
    inputLines.push(line);
}).on("close", () => {
    const [n, d] = inputLines[0].split(" ").map(Number);
    const arr = inputLines[1].split(" ").map(Number);

    // 排序
    arr.sort((a, b) => a - b);

    // dp[i] 代表 i 个队伍能匹配的队伍数量
    const dp = new Array(n + 1).fill(0);
    // minSum[i] i 个队伍匹配之后差值的和
    const minSum = new Array(n + 1).fill(0);

    for (let i = 2; i <= n; i++) {
        const canMatch = arr[i - 1] - arr[i - 2] <= d;
        // 优先选择匹配数量多的，匹配数量相等的选择差异值和小的
        if (canMatch) {
            if (dp[i - 1] < dp[i - 2] + 1) {
                dp[i] = dp[i - 2] + 1;
                minSum[i] = minSum[i - 2] + (arr[i - 1] - arr[i - 2]);
            } else if (dp[i - 1] > dp[i - 2] + 1) {
                dp[i] = dp[i - 1];
                minSum[i] = minSum[i - 1];
            } else {
                dp[i] = dp[i - 1];
                minSum[i] = Math.min(minSum[i - 1], minSum[i - 2] + (arr
[i - 1] - arr[i - 2]));
            }
        } else {
            dp[i] = dp[i - 1];
            minSum[i] = minSum[i - 1];
        }
    }

    console.log(dp[n] === 0 ? -1 : minSum[n]);
```

```
45  });
```

Go

Plain Text

```
package main

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

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

  // 读取 n 和 d
  scanner.Scan()
  nd := strings.Split(scanner.Text(), " ")
  n, _ := strconv.Atoi(nd[0])
  d, _ := strconv.Atoi(nd[1])

  // 读取数组
  scanner.Scan()
  strArr := strings.Split(scanner.Text(), " ")
  arr := make([]int, n)
  for i := 0; i < n; i++ {
    arr[i], _ = strconv.Atoi(strArr[i])
  }

  // 排序
  sort.Ints(arr)

  // dp[i] 代表 i 个队伍能匹配的队伍数量
  dp := make([]int, n+1)
  // minSum[i] i 个队伍匹配之后差值的和
  minSum := make([]int, n+1)

  for i := 2; i <= n; i++ {
    canMatch := arr[i-1] - arr[i-2] <= d
        // 优先选择匹配数量多的，匹配数量相等的选择差异值和小的
    if canMatch {
      if dp[i-1] < dp[i-2]+1 {
        dp[i] = dp[i-2] + 1
        minSum[i] = minSum[i-2] + (arr[i-1] - arr[i-2])
      } else if dp[i-1] > dp[i-2]+1 {
        dp[i] = dp[i-1]
```

```
            minSum[i] = minSum[i-1]
        } else {
            dp[i] = dp[i-1]
            minSum[i] = min(minSum[i-1], minSum[i-2]+(arr[i-1]-arr[i-2]))
        }
      } else {
        dp[i] = dp[i-1]
        minSum[i] = minSum[i-1]
      }
    }

    if dp[n] == 0 {
      fmt.Println(-1)
    } else {
      fmt.Println(minSum[n])
    }
}
```

> 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: 点击去查看

> `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x2，例如出3张3，得分为(3+3+3)x2=18
- 出5张顺，记牌面分数总和再x2，例如出34567顺，得分为(3+4+5+6+7)x2=50
- 出4张炸弹，记牌面分数总和再x3，例如出4张4，得分为4x4x3=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

```
33445677
```

输出

```
67
```

说明

```
出对3、对4、对7，单张5、6，得分为67;
出34567顺，再出单张3、4、7，得分为64
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++

Plain Text

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

int res = 0;

// 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
void dfs(vector<int> cardCount, int count, int index , int score) {
    if (count == 0) {
        res = max(res, score);
        return;
    }

    if (cardCount[index] == 0) {
        dfs(cardCount, count, index + 1, score);
        return;
    }

    // 顺子策略
    if (index <= 9 && cardCount[index + 1] >= 1 && cardCount[index + 2] >= 1 && cardCount[index + 3] >= 1 && cardCount[index + 4] >= 1) {
        cardCount[index]--;
        cardCount[index+1]--;
        cardCount[index+2]--;
        cardCount[index+3]--;
        cardCount[index+4]--;

        dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2);
        // 回溯
        cardCount[index]++;
        cardCount[index+1]++;
        cardCount[index+2]++;
        cardCount[index+3]++;
        cardCount[index+4]++;
    }
    // 对子
    if (cardCount[index] >= 2) {
        cardCount[index] -=2;
        dfs(cardCount, count - 2, index, score + index * 2 * 2);
        cardCount[index] +=2;
```

```
    }

    // 三张
    if (cardCount[index] >= 3) {
        cardCount[index] -=3;
        dfs(cardCount, count - 3, index, score + index * 3 * 2);
        cardCount[index] +=3;
    }
    // 四张
    if (cardCount[index] >= 4) {
        cardCount[index] -=4;
        dfs(cardCount, count - 4, index, score + index * 3 * 4);
        cardCount[index] +=4;
    }

    // 单张
    cardCount[index]--;
    dfs(cardCount, count - 1, index, score + index);
    cardCount[index]++;
}

int main() {
    string s;
    cin >> s;
    // 记录牌的数量
    vector<int> cardCount(14, 0);
    for (int i = 0; i < s.size(); i++) {
        char c = s[i];
        if (c == '0') {
            cardCount[10]++;
        }else if (c == 'J') {
             cardCount[11]++;
        }else if (c == 'Q') {
             cardCount[12]++;
        }else if (c == 'K') {
             cardCount[13]++;
        }else {
            cardCount[c -'0']++;
        }
    }

    dfs(cardCount, s.size(), 1 , 0);

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

Plain Text

```
import java.util.*;

public class Main {
    static int res = 0;

    // 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
    public static void dfs(int[] cardCount, int count, int index, int score) {
        if (count == 0) {
            res = Math.max(res, score);
            return;
        }

        if (cardCount[index] == 0) {
            dfs(cardCount, count, index + 1, score);
            return;
        }

        // 顺子策略
        if (index <= 9 && cardCount[index + 1] >= 1 && cardCount[index + 2] >= 1 && cardCount[index + 3] >= 1 && cardCount[index + 4] >= 1) {
            cardCount[index]--;
            cardCount[index + 1]--;
            cardCount[index + 2]--;
            cardCount[index + 3]--;
            cardCount[index + 4]--;

            dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2);
            // 回溯
            cardCount[index]++;
            cardCount[index + 1]++;
            cardCount[index + 2]++;
            cardCount[index + 3]++;
            cardCount[index + 4]++;
        }

        // 对子
        if (cardCount[index] >= 2) {
            cardCount[index] -= 2;
            dfs(cardCount, count - 2, index, score + index * 2 * 2);
            cardCount[index] += 2;
        }

```

```
        // 三张
        if (cardCount[index] >= 3) {
            cardCount[index] -= 3;
            dfs(cardCount, count - 3, index, score + index * 3 * 2);
            cardCount[index] += 3;
        }

        // 四张
        if (cardCount[index] >= 4) {
            cardCount[index] -= 4;
            dfs(cardCount, count - 4, index, score + index * 3 * 4);
            cardCount[index] += 4;
        }

        // 单张
        cardCount[index]--;
        dfs(cardCount, count - 1, index, score + index);
        cardCount[index]++;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String s = scanner.nextLine();
        // 记录牌的数量
        int[] cardCount = new int[14];
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c == '0') {
                cardCount[10]++;
            } else if (c == 'J') {
                cardCount[11]++;
            } else if (c == 'Q') {
                cardCount[12]++;
            } else if (c == 'K') {
                cardCount[13]++;
            } else {
                cardCount[c - '0']++;
            }
        }

        dfs(cardCount, s.length(), 1, 0);

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

Python

```
# 全局变量记录最大得分
res = 0

# 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
def dfs(cardCount, count, index, score):
    global res
    if count == 0:
        res = max(res, score)
        return

    if cardCount[index] == 0:
        dfs(cardCount, count, index + 1, score)
        return

    # 顺子策略
    if index <= 9 and cardCount[index + 1] >= 1 and cardCount[index + 2] >= 1 and cardCount[index + 3] >= 1 and cardCount[index + 4] >= 1:
        cardCount[index] -= 1
        cardCount[index + 1] -= 1
        cardCount[index + 2] -= 1
        cardCount[index + 3] -= 1
        cardCount[index + 4] -= 1

        dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2)
        # 回溯
        cardCount[index] += 1
        cardCount[index + 1] += 1
        cardCount[index + 2] += 1
        cardCount[index + 3] += 1
        cardCount[index + 4] += 1

    # 对子
    if cardCount[index] >= 2:
        cardCount[index] -= 2
        dfs(cardCount, count - 2, index, score + index * 2 * 2)
        cardCount[index] += 2

    # 三张
    if cardCount[index] >= 3:
        cardCount[index] -= 3
        dfs(cardCount, count - 3, index, score + index * 3 * 2)
        cardCount[index] += 3

    # 四张
```

```
    if cardCount[index] >= 4:
        cardCount[index] -= 4
        dfs(cardCount, count - 4, index, score + index * 3 * 4)
        cardCount[index] += 4

    # 单张
    cardCount[index] -= 1
    dfs(cardCount, count - 1, index, score + index)
    cardCount[index] += 1

def main():
    s = input().strip()  # 读取输入字符串
    # 记录牌的数量
    cardCount = [0] * 14
    for c in s:
        if c == '0':
            cardCount[10] += 1
        elif c == 'J':
            cardCount[11] += 1
        elif c == 'Q':
            cardCount[12] += 1
        elif c == 'K':
            cardCount[13] += 1
        else:
            cardCount[int(c)] += 1

    dfs(cardCount, len(s), 1, 0)

    print(res)

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

JavaScript

```
let res = 0;

// 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
function dfs(cardCount, count, index, score) {
    if (count === 0) {
        res = Math.max(res, score);
        return;
    }

    if (cardCount[index] === 0) {
        dfs(cardCount, count, index + 1, score);
        return;
    }

    // 顺子策略
    if (index <= 9 && cardCount[index + 1] >= 1 && cardCount[index + 2] >= 1 && cardCount[index + 3] >= 1 && cardCount[index + 4] >= 1) {
        cardCount[index]--;
        cardCount[index + 1]--;
        cardCount[index + 2]--;
        cardCount[index + 3]--;
        cardCount[index + 4]--;

        dfs(cardCount, count - 5, index, score + (index * 5 + 10) * 2);
        // 回溯
        cardCount[index]++;
        cardCount[index + 1]++;
        cardCount[index + 2]++;
        cardCount[index + 3]++;
        cardCount[index + 4]++;
    }

    // 对子
    if (cardCount[index] >= 2) {
        cardCount[index] -= 2;
        dfs(cardCount, count - 2, index, score + index * 2 * 2);
        cardCount[index] += 2;
    }

    // 三张
    if (cardCount[index] >= 3) {
        cardCount[index] -= 3;
        dfs(cardCount, count - 3, index, score + index * 3 * 2);
        cardCount[index] += 3;
```

```
    }

    // 四张
    if (cardCount[index] >= 4) {
        cardCount[index] -= 4;
        dfs(cardCount, count - 4, index, score + index * 3 * 4);
        cardCount[index] += 4;
    }

    // 单张
    cardCount[index]--;
    dfs(cardCount, count - 1, index, score + index);
    cardCount[index]++;
}

// 主函数，负责输入输出和调用递归
function main() {
    // 从控制台读取输入
    const readline = require('readline');
    const rl = readline.createInterface({
        input: process.stdin,
        output: process.stdout
    });

    rl.question('', (s) => {
        const cardCount = new Array(14).fill(0);

        // 记录牌的数量
        for (let i = 0; i < s.length; i++) {
            const c = s[i];
            if (c === '0') {
                cardCount[10]++;
            } else if (c === 'J') {
                cardCount[11]++;
            } else if (c === 'Q') {
                cardCount[12]++;
            } else if (c === 'K') {
                cardCount[13]++;
            } else {
                cardCount[parseInt(c)]++;
            }
        }

        dfs(cardCount, s.length, 1, 0);

        // 输出最大得分
        console.log(res);
        rl.close();
```

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

Go

Plain Text

```
package main

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

var res int

// dfs 深度优先搜索（回溯）
func dfs(cardCount []int, count, index, score int) {
  if count == 0 {
    if score > res {
      res = score
    }
    return
  }

  if cardCount[index] == 0 {
    dfs(cardCount, count, index+1, score)
    return
  }

  // 顺子策略
  if index <= 9 && cardCount[index+1] >= 1 && cardCount[index+2] >= 1 && c
ardCount[index+3] >= 1 && cardCount[index+4] >= 1 {
    cardCount[index]--
    cardCount[index+1]--
    cardCount[index+2]--
    cardCount[index+3]--
    cardCount[index+4]--

    dfs(cardCount, count-5, index, score+(index*5+10)*2)
    // 回溯
    cardCount[index]++
    cardCount[index+1]++
    cardCount[index+2]++
    cardCount[index+3]++
    cardCount[index+4]++
  }

  // 对子
  if cardCount[index] >= 2 {
    cardCount[index] -= 2
```

```
      dfs(cardCount, count-2, index, score+index*2*2)
      cardCount[index] += 2
    }

    // 三张
    if cardCount[index] >= 3 {
      cardCount[index] -= 3
      dfs(cardCount, count-3, index, score+index*3*2)
      cardCount[index] += 3
    }

    // 四张
    if cardCount[index] >= 4 {
      cardCount[index] -= 4
      dfs(cardCount, count-4, index, score+index*3*4)
      cardCount[index] += 4
    }

    // 单张
    cardCount[index]--
    dfs(cardCount, count-1, index, score+index)
    cardCount[index]++
  }

  func main() {
    // 读取输入
    reader := bufio.NewReader(os.Stdin)
    s, _ := reader.ReadString('\n')
    s = s[:len(s)-1] // 去除换行符

    // 初始化牌的数量
    cardCount := make([]int, 14)

    // 统计每张牌的数量
    for _, c := range s {
      switch c {
      case '0':
        cardCount[10]++
      case 'J':
        cardCount[11]++
      case 'Q':
        cardCount[12]++
      case 'K':
        cardCount[13]++
      default:
        cardCount[int(c-'0')]++
      }
    }
```

```
    // 调用深度优先搜索
    dfs(cardCount, len(s), 1, 0)

    // 输出最大得分
    fmt.Println(res)
}
```

> 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 ≤ N ≤ 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
yM eman si boB.
```

输出

```Plain Text
My name is Bob.
```

用例2

输入

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

输出

```
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++

Plain Text

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

int main() {
    string s;
    getline(cin, s);
    string currentStr = "";
    for (int i = 0; i < s.size(); i++) {
        if (isalpha(s[i])) {
            currentStr += s[i];
        } else {
            // 单词反转
            if (!s.empty()) {
                reverse(currentStr.begin(), currentStr.end());
                cout << currentStr;
            }
            currentStr = "";
            // 非字母字符直接输出
            cout << s[i];
        }
    }
    // 收尾操作
    if (!currentStr.empty()) {
        reverse(currentStr.begin(), currentStr.end());
        cout << currentStr;
    }
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

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

        StringBuilder currentStr = new StringBuilder();
        StringBuilder result = new StringBuilder();

        for (char c : s.toCharArray()) {
            if (Character.isLetter(c)) {
                currentStr.append(c);
            } else {
                // 反转单词并追加
                if (currentStr.length() > 0) {
                    result.append(currentStr.reverse());
                    currentStr.setLength(0); // 清空 currentStr
                }
                // 直接追加标点或空格
                result.append(c);
            }
        }
        // 处理最后一个单词 收尾操作
        if (currentStr.length() > 0) {
            result.append(currentStr.reverse());
        }
        System.out.println(result);
    }
}
```

Python

Plain Text

```
import sys

def reverse_words(sentence):
    result = []
    current_str = []

    for char in sentence:
        #  拼接
        if char.isalpha():
            current_str.append(char)
        else:
            # 反转
            if current_str:
                result.append("".join(reversed(current_str)))
                current_str = []
            result.append(char)
    # 收尾操作
    if current_str:
        result.append("".join(reversed(current_str)))

    print("".join(result))

if __name__ == "__main__":
    sentence = sys.stdin.readline().strip()
    reverse_words(sentence)
```

JavaScript

Plain Text

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

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

rl.on("line", (s) => {
    let currentStr = "";
    let result = "";

    for (let i = 0; i < s.length; i++) {
        if (/[a-zA-Z]/.test(s[i])) {
            currentStr += s[i];
        } else {
            if (currentStr.length > 0) {
                result += currentStr.split("").reverse().join(""); // 反转单词
                currentStr = "";
            }
            result += s[i]; // 追加标点
        }
    }
    // 收尾操作
    if (currentStr.length > 0) {
        result += currentStr.split("").reverse().join(""); // 处理最后一个单词
    }

    console.log(result);
    rl.close();
});
```

Go

Plain Text

```
package main

import (
  "bufio"
  "fmt"
  "os"
  "unicode"
)
// 反转
func reverseWord(word []rune) {
  for i, j := 0, len(word)-1; i < j; i, j = i+1, j-1 {
    word[i], word[j] = word[j], word[i]
  }
}

func main() {
  reader := bufio.NewReader(os.Stdin)
  s, _ := reader.ReadString('\n')
  s = s[:len(s)-1] // 移除换行符

  var result []rune
  var currentStr []rune

  for _, c := range s {
        // 直接拼接
    if unicode.IsLetter(c) {
      currentStr = append(currentStr, c)
    } else {
          // 反转
      if len(currentStr) > 0 {
        reverseWord(currentStr)
        result = append(result, currentStr...)
        currentStr = nil
      }
      result = append(result, c)
    }
  }
    // 收尾操作
  if len(currentStr) > 0 {
    reverseWord(currentStr)
    result = append(result, currentStr...)
  }

  fmt.Println(string(result))
}
```

> 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 ≤ i ≤ n）块木板长度为 ai。
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 ≤ n ≤ 10^3），m（1 ≤ m ≤ 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， a1, a2,...an（1 ≤ ai ≤ 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

```
5 3
4 5 3 5 5
```

输出

Plain Text

```
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
5 2
4 5 3 5 5
```

输出

```
Plain Text
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 `{len1, x1}` ,次短的结构为 `{len2, x2}` .此时根据m的值进行处理

- `(len2 - len1) * x1 <= m` 时: 说明最短都可以变成次短长度，此时进行的操作为 `m -= (len2 - len1) * x1 , 次最短数量变为 x1 + x2` .再进行一次循环处理。
- `(len2 - 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 `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++

Plain Text

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

int main() {
    int n,m;
    cin >> n >>m;
    vector<int> ans(n);
    for (int i = 0; i < n; i++) {
        cin >> ans[i];
    }

    sort(ans.begin(), ans.end());
    // 贪心算法
    while (m--) {
        int pos = 1;
        for (; pos < n; pos++) {
            if (ans[pos] > ans[pos-1]) {
                break;
            }
        }
        ans[pos-1]++;
    }

    cout << ans[0];
    return 0;
}
```

优化方案

Plain Text

```
// 优化实现
#include<iostream>
#include<vector>
#include<string>
#include <utility>
#include <sstream>
#include<map>
#include<algorithm>
#include<queue>
using namespace std;


struct Compare {
    bool operator()(const pair<int, int>& a, const pair<int, int>& b) {
        return a.first > b.first; // first 值越大优先级越低
    }
};


int main() {
    int n,m;
    cin >> n >>m;
    vector<int> ans(n);
    map<int,int> mp;
    for (int i = 0; i < n; i++) {
        cin >> ans[i];
        mp[ans[i]]++;
    }

    // 小顶堆
    priority_queue<pair<int, int>, vector<pair<int, int>>, Compare> pq;
    for (auto p : mp) {
        pq.push({p.first, p.second});
    }
    while (m != 0) {
        pair<int,int> top = pq.top();
        // 都是同样长度，平均分
        if (pq.size() == 1) {
            int count = top.second;
            int num = top.first;
            // 会自定向下取整的
            int diff = m / count;
            pq.pop();
            pq.push({num+diff, count});
            break;
```

```
        } else {
            pq.pop();
            pair<int,int> se = pq.top();
            int diff = se.first - top.first;
            int count = top.second;

            if (diff * count <= m) {
                pq.pop();
                // 将值提升至se.fisrt消耗的长度
                pq.push({se.first, count + se.second});
                m -= diff * count;
            } else {
                // 这是数量写多少都不影响答案
                pq.push({top.first + (m / count), 1});
                break;
            }
        }
    }
    cout << pq.top().first;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    // 自定义比较器
    static class Compare implements Comparator<Map.Entry<Integer, Integer>
> {
        public int compare(Map.Entry<Integer, Integer> a, Map.Entry<Intege
r, Integer> b) {
            return Integer.compare(a.getKey(), b.getKey()); // 按照数字升序
排序
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int m = scanner.nextInt();
        int[] ans = new int[n];
        Map<Integer, Integer> mp = new HashMap<>();

        // 输入数据并统计
        for (int i = 0; i < n; i++) {
            ans[i] = scanner.nextInt();
            mp.put(ans[i], mp.getOrDefault(ans[i], 0) + 1);
        }

        // 小顶堆
        PriorityQueue<Map.Entry<Integer, Integer>> pq = new PriorityQueue<
>(new Compare());
        pq.addAll(mp.entrySet());

        while (m != 0) {
            Map.Entry<Integer, Integer> top = pq.poll();
            // 都是同样长度，平均分
            if (pq.size() == 0) {
                int count = top.getValue();
                int num = top.getKey();
                int diff = m / count;
                pq.offer(new AbstractMap.SimpleEntry<>(num + diff, coun
t));
                break;
            } else {
                Map.Entry<Integer, Integer> se = pq.peek();
                int diff = se.getKey() - top.getKey();
                int count = top.getValue();
```

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

Python

Plain Text

```
import heapq
from collections import Counter

def main():
    n, m = map(int, input().split())
    ans = list(map(int, input().split()))
    mp = Counter(ans)

    # 小顶堆
    pq = []
    for num, count in mp.items():
        heapq.heappush(pq, (num, count))

    while m != 0:
        top = heapq.heappop(pq)
        # 都是同样长度，平均分
        if len(pq) == 0:
            count = top[1]
            num = top[0]
            diff = m // count
            heapq.heappush(pq, (num + diff, count))
            break
        else:
            se = pq[0]
            diff = se[0] - top[0]
            count = top[1]

            if diff * count <= m:
                heapq.heappop(pq)
                heapq.heappush(pq, (se[0], count + se[1]))
                m -= diff * count
            else:
                heapq.heappush(pq, (top[0] + (m // count), 1))
                break

    print(pq[0][0])

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

JavaScript

Plain Text

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

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

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

  if (lines.length === 2) {
    const [n, m] = lines[0].split(" ").map(Number);
    const plankLengths = lines[1].split(" ").map(Number);
    console.log(findMaxEqualLength(m, plankLengths));

    lines.length = 0;
  }
});

function findMaxEqualLength(m, planks) {
  // 统计每种长度的木板数量
  const plankCount = new Map();
  for (let length of planks) {
    plankCount.set(length, (plankCount.get(length) || 0) + 1);
  }

  // 按照木板长度升序排序
  const sortedPlanks = Array.from(plankCount.entries()).sort((a, b) => a
[0] - b[0]);

  while (m > 0) {
    if (sortedPlanks.length === 1) {
      // 只有一种木板长度，均分 m
      return sortedPlanks[0][0] + Math.floor(m / sortedPlanks[0][1]);
    }

    let [minLength, minCount] = sortedPlanks.shift();
    let [nextLength, nextCount] = sortedPlanks[0];

    let requiredWood = (nextLength - minLength) * minCount;

    if (requiredWood > m) {
      return minLength + Math.floor(m / minCount);
```

```
      } else {
        m -= requiredWood;
        sortedPlanks[0][1] += minCount; // 合并计数
      }
    }

    return sortedPlanks[0][0];
  }
```

Go

Plain Text

```
package main

import (
  "fmt"
  "sort"
)

func getResult(m int, a []int) int {
  // 统计木板长度及其数量
  boardCount := make(map[int]int)
  for _, length := range a {
    boardCount[length]++
  }

  // 转换为切片，并按长度排序
  type board struct {
    length int
    count  int
  }

  boards := make([]board, 0, len(boardCount))
  for length, count := range boardCount {
    boards = append(boards, board{length, count})
  }
  sort.Slice(boards, func(i, j int) bool {
    return boards[i].length < boards[j].length
  })

  // 逐步填充最短板
  for i := 0; i < len(boards)-1; i++ {
    cur := boards[i]
    next := boards[i+1]

    // 计算当前木板与下一个木板的长度差距
    diff := next.length - cur.length
    totalCost := diff * cur.count

    if totalCost > m {
      return cur.length + m/cur.count
    }

    m -= totalCost
    boards[i+1].count += cur.count
  }

```

```
    // 只剩下一种木板时，平分剩余 m
    return boards[len(boards)-1].length + m/boards[len(boards)-1].count
}

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

    a := make([]int, n)
    for i := range a {
        fmt.Scan(&a[i])
    }

    fmt.Println(getResult(m, a))
}
```

> 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 6 710 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 2 3 4 5 6 7 8 9 10
```

输出

Plain Text

```
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++

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <climits>

using namespace std;

void findMinDifference(vector<int>& nums, int idx, int count, int sum1, int totalSum, int& minDiff) {
    if (count == 5) { // 当选出5个成员时
        int sum2 = totalSum - sum1;
        // 记录最小值
        minDiff = min(minDiff, abs(sum1 - sum2));
        return;
    }

    if (idx == 10 || count + (10 - idx) < 5) return; // 剩余人数不足时剪枝

    // 选
    findMinDifference(nums, idx + 1, count + 1, sum1 + nums[idx], totalSum, minDiff);
    // 不选
    findMinDifference(nums, idx + 1, count, sum1, totalSum, minDiff);
}

int main() {
    vector<int> nums(10);
    int totalSum = 0;
    for (int& num : nums) {
        cin >> num;
        totalSum += num;
    }

    int minDiff = INT_MAX;
    findMinDifference(nums, 0, 0, 0, totalSum, minDiff);
    cout << minDiff << endl;
    return 0;
}
```

Java

```
import java.util.*;

public class Main {
    static int minDiff = Integer.MAX_VALUE;
    static int totalSum = 0;

    // 递归查找最小差值
    public static void findMinDifference(int[] nums, int idx, int count, int sum1) {
        if (count == 5) { // 当选出5个成员时
            int sum2 = totalSum - sum1;
            minDiff = Math.min(minDiff, Math.abs(sum1 - sum2));
            return;
        }

        if (idx == 10 || count + (10 - idx) < 5) return; // 剩余人数不足时剪枝
        //选择当前元素
        findMinDifference(nums, idx + 1, count + 1, sum1 + nums[idx]);
        // 不选择当前元素
        findMinDifference(nums, idx + 1, count, sum1);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] nums = new int[10];
        totalSum = 0;

        // 读取10个整数
        for (int i = 0; i < 10; i++) {
            nums[i] = scanner.nextInt();
            totalSum += nums[i];
        }
        scanner.close();

        findMinDifference(nums, 0, 0, 0);
        System.out.println(minDiff);
    }
}
```

Python

```
import sys

def find_min_difference(nums, idx, count, sum1, total_sum, min_diff):
    if count == 5:  # 选出5个成员时
        sum2 = total_sum - sum1
        return min(min_diff, abs(sum1 - sum2))

    if idx == 10 or count + (10 - idx) < 5:  # 剩余人数不足时剪枝
        return min_diff

    # 选择当前元素
    min_diff = find_min_difference(nums, idx + 1, count + 1, sum1 + nums[idx], total_sum, min_diff)

    # 不选择当前元素
    min_diff = find_min_difference(nums, idx + 1, count, sum1, total_sum, min_diff)

    return min_diff

def main():
    nums = list(map(int, sys.stdin.read().split()))  # 从标准输入读取 10 个整数
    total_sum = sum(nums)

    result = find_min_difference(nums, 0, 0, 0, total_sum, float('inf'))
    print(result)

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

JavaScript

Plain Text

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

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

let nums = [];
rl.on("line", (line) => {
    nums = line.trim().split(" ").map(Number); // 读取输入并转换为整数数组
    rl.close();
});

rl.on("close", () => {
    let totalSum = nums.reduce((a, b) => a + b, 0);
    let minDiff = Infinity;

    function findMinDifference(idx, count, sum1) {
        if (count === 5) { // 选出5个成员时
            let sum2 = totalSum - sum1;
            minDiff = Math.min(minDiff, Math.abs(sum1 - sum2));
            return;
        }

        if (idx === 10 || count + (10 - idx) < 5) return; // 剩余人数不足时剪枝

        findMinDifference(idx + 1, count + 1, sum1 + nums[idx]); // 选择当前元素
        findMinDifference(idx + 1, count, sum1); // 不选择当前元素
    }

    findMinDifference(0, 0, 0);
    console.log(minDiff);
});
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math"
)

var minDiff = math.MaxInt32
var totalSum int

func findMinDifference(nums []int, idx, count, sum1 int) {
  if count == 5 { // 选出5个成员时
    sum2 := totalSum - sum1
    if diff := abs(sum1 - sum2); diff < minDiff {
      minDiff = diff
    }
    return
  }

  if idx == 10 || count+(10-idx) < 5 { // 剩余人数不足时剪枝
    return
  }

  findMinDifference(nums, idx+1, count+1, sum1+nums[idx]) // 选择当前元素
  findMinDifference(nums, idx+1, count, sum1)             // 不选择当前元素
}

func abs(x int) int {
  if x < 0 {
    return -x
  }
  return x
}

func main() {
  nums := make([]int, 10)
  totalSum = 0

  // 读取10个整数
  for i := 0; i < 10; i++ {
    fmt.Scan(&nums[i])
    totalSum += nums[i]
  }

  findMinDifference(nums, 0, 0, 0)
```

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

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++

Plain Text

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

// 快速求出对应数字1个位置
vector<int> countOnesAndPositions(int n) {
    vector<int> positions;
    int index = 0;
    while (n) {
        if (n & 1) positions.push_back(index);
        n >>= 1;
        index++;
    }
    return  positions;
}


int main() {
    vector<int> nums(10);
    int totalSum = 0;
    for (int& num : nums) {
        cin >> num;
        totalSum += num;
    }
    // 初始定义一个绝对不可能取到的最大结果
    int res = 10000 * 10;
    for (int i = 1; i < pow(2, 10); i++) {
        // 计算1的数量以及 位置
        vector<int> onePosition = countOnesAndPositions(i);
        if (onePosition.size() != 5) {
            continue;
        }
        int oneTeamSum = 0;
        for (int j = 0; j < 5; j++) {
            oneTeamSum += nums[onePosition[j]];
        }
        // 计算差值
        int diff = abs((totalSum - oneTeamSum) - oneTeamSum);
        res = min(res, diff);
    }
    cout << res;
    return 0;
```

```
46  }
```

Java

Plain Text

```
import java.util.*;

public class Main {
    // 快速求出对应数字1个位置
    public static List<Integer> countOnesAndPositions(int n) {
        List<Integer> positions = new ArrayList<>();
        int index = 0;
        while (n != 0) {
            if ((n & 1) == 1) positions.add(index);
            n >>= 1;
            index++;
        }
        return positions;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int[] nums = new int[10];
        int totalSum = 0;
        for (int i = 0; i < 10; i++) {
            nums[i] = scanner.nextInt();
            totalSum += nums[i];
        }
        // 初始定义一个绝对不可能取到的最大结果
        int res = 10000 * 10;
        // 枚举所有非空子集 (1 到 2^10-1)
        for (int i = 1; i < (1 << 10); i++) {
            // 计算1的位置
            List<Integer> onePosition = countOnesAndPositions(i);
            if (onePosition.size() != 5) {
                continue;
            }
            int oneTeamSum = 0;
            for (int pos : onePosition) {
                oneTeamSum += nums[pos];
            }
            // 计算差值
            int diff = Math.abs((totalSum - oneTeamSum) - oneTeamSum);
            res = Math.min(res, diff);
        }
        System.out.println(res);
    }
}
```

Python

```
Plain Text
```

```
# 快速求出对应数字1个位置
def count_ones_and_positions(n):
    positions = []
    index = 0
    while n:
        if n & 1:
            positions.append(index)
        n >>= 1
        index += 1
    return positions

def main():
    nums = list(map(int, input().split()))
    total_sum = sum(nums)
    # 初始定义一个绝对不可能取到的最大结果
    res = 10000 * 10
    # 枚举所有非空子集 (1 到 2^10-1)
    for i in range(1, 1 << 10):
        # 计算1的位置
        one_position = count_ones_and_positions(i)
        if len(one_position) != 5:
            continue
        one_team_sum = sum(nums[pos] for pos in one_position)
        # 计算差值
        diff = abs((total_sum - one_team_sum) - one_team_sum)
        res = min(res, diff)
    print(res)

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

JavaScript

Plain Text

```
// 快速求出对应数字1个位置
function countOnesAndPositions(n) {
    const positions = [];
    let index = 0;
    while (n) {
        if (n & 1) positions.push(index);
        n = n >>> 1;
        index++;
    }
    return positions;
}

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

    rl.question('', (input) => {
        const nums = input.split(' ').map(Number);
        const totalSum = nums.reduce((a, b) => a + b, 0);
        // 初始定义一个绝对不可能取到的最大结果
        let res = 10000 * 10;
        // 枚举所有非空子集 (1 到 2^10-1)
        for (let i = 1; i < Math.pow(2, 10); i++) {
            // 计算1的位置
            const onePosition = countOnesAndPositions(i);
            if (onePosition.length !== 5) {
                continue;
            }
            let oneTeamSum = 0;
            for (const pos of onePosition) {
                oneTeamSum += nums[pos];
            }
            // 计算差值
            const diff = Math.abs((totalSum - oneTeamSum) - oneTeamSum);
            res = Math.min(res, diff);
        }
        console.log(res);
        rl.close();
    });
}

main();
```

Go

Plain Text

```
package main

import (
  "fmt"
  "math"
)

// 快速求出对应数字1个位置
func countOnesAndPositions(n int) []int {
  positions := []int{}
  index := 0
  for n > 0 {
    if n&1 == 1 {
      positions = append(positions, index)
    }
    n >>= 1
    index++
  }
  return positions
}

func main() {
  nums := make([]int, 10)
  totalSum := 0
  for i := range nums {
    fmt.Scan(&nums[i])
    totalSum += nums[i]
  }
  // 初始定义一个绝对不可能取到的最大结果
  res := 10000 * 10
  // 枚举所有非空子集 (1 到 2^10-1)
  for i := 1; i < int(math.Pow(2, 10)); i++ {
    // 计算1的位置
    onePosition := countOnesAndPositions(i)
    if len(onePosition) != 5 {
      continue
    }
    oneTeamSum := 0
    for _, pos := range onePosition {
      oneTeamSum += nums[pos]
    }
    // 计算差值
    diff := int(math.Abs(float64((totalSum - oneTeamSum) - oneTeamSum)))
    if diff < res {
      res = diff
```

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

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不同的代先级，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 <= 5, 0 < NUM <= 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 <= 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

```
7
IN 1 1
IN 1 2
IN 1 3
IN 2 1
OUT 1
OUT 2
OUT 2
```

输出

```
3
4
NULL
```

用例2

输入

```
5
IN 1 1
IN 1 3
IN 1 1
IN 1 3
OUT 1
```

输出

```
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++

Plain Text

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


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

// 自定义比较函数（小顶堆）
struct cmp {
    bool operator()(pair<int,  int> p1, pair<int,  int> p2) {
        if (p1.first == p2.first) {
            return p1.second > p2.second;
        }
        return p1.first < p2.first;
    }
};

int main() {
    int n ;
    cin >> n;
    cin.ignore();
    // 优先队列
    vector<priority_queue<pair<int,int>, vector<pair<int,  int>>, cmp>> an
s(5);
    int x = 1;
```

```
    for (int i = 0; i < n; i++) {
        string s;
        getline(cin, s);
        vector<string> tmp = split(s, " ");
        int index = stoi(tmp[1]) -1;
        if (tmp[0] == "IN") {
            ans[index].push({stoi(tmp[2]), x});
            x++;
        }else {
            if (ans[index].empty()) {
                cout << "NULL" << endl;
            } else {
                cout << ans[index].top().second << endl;
                ans[index].pop();
            }
        }
    }
    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

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

    // 自定义比较器（小顶堆）
    static class CustomComparator implements Comparator<int[]> {
        @Override
        public int compare(int[] p1, int[] p2) {
            if (p1[0] == p2[0]) {
                return Integer.compare(p1[1], p2[1]);
            }
            return Integer.compare(p2[0], p1[0]);
        }
    }

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

        // 初始化优先队列
        List<PriorityQueue<int[]>> ans = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            ans.add(new PriorityQueue<>(new CustomComparator()));
        }

        int x = 1;
        for (int i = 0; i < n; i++) {
            String s = scanner.nextLine();
            List<String> tmp = split(s, " ");
            int index = Integer.parseInt(tmp.get(1)) - 1;

            if (tmp.get(0).equals("IN")) {
                ans.get(index).offer(new int[]{Integer.parseInt(tmp.get
(2)), x});
                x++;
            } else {
                if (ans.get(index).isEmpty()) {
                    System.out.println("NULL");
                } else {
                    System.out.println(ans.get(index).poll()[1]);
                }
```

```
            }
        }
        scanner.close();
    }
}
```

Python

Plain Text

```
import sys
import heapq

# 读取输入
n = int(sys.stdin.readline().strip())

# 优先队列数组（最大堆）
ans = [ [] for _ in range(5) ]

x = 1
for _ in range(n):
    tmp = sys.stdin.readline().strip().split()
    index = int(tmp[1]) - 1

    if tmp[0] == "IN":
        # 由于 heapq 是最小堆，为了实现最大堆，我们将 first 取负数
        heapq.heappush(ans[index], (-int(tmp[2]), x))
        x += 1
    else:
        if not ans[index]:
            print("NULL")
        else:
            # 取出时恢复 first 的原值
            print(heapq.heappop(ans[index])[1])
```

JavaScript

Plain Text

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

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

class PriorityQueue {
  constructor(compare) {
    this.heap = [];
    this.compare = compare;
  }

  push(item) {
    this.heap.push(item);
    this._heapifyUp();
  }

  pop() {
    if (this.isEmpty()) return null;
    const top = this.heap[0];
    const last = this.heap.pop();
    if (!this.isEmpty()) {
      this.heap[0] = last;
      this._heapifyDown();
    }
    return top;
  }

  top() {
    return this.isEmpty() ? null : this.heap[0];
  }

  isEmpty() {
    return this.heap.length === 0;
  }

  _heapifyUp() {
    let index = this.heap.length - 1;
    while (index > 0) {
      let parent = Math.floor((index - 1) / 2);
      if (this.compare(this.heap[parent], this.heap[index]) > 0) {
        [this.heap[parent], this.heap[index]] = [this.heap[index], this.h
eap[parent]];
        index = parent;
```

```
      } else {
        break;
      }
    }
  }

  _heapifyDown() {
    let index = 0;
    let length = this.heap.length;
    while (true) {
      let left = 2 * index + 1;
      let right = 2 * index + 2;
      let smallest = index;

      if (left < length && this.compare(this.heap[left], this.heap[smallest]) < 0) {
        smallest = left;
      }
      if (right < length && this.compare(this.heap[right], this.heap[smallest]) < 0) {
        smallest = right;
      }
      if (smallest !== index) {
        [this.heap[smallest], this.heap[index]] = [this.heap[index], this.heap[smallest]];
        index = smallest;
      } else {
        break;
      }
    }
  }
}

// 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）
const compare = (a, b) => {
  if (a[0] === b[0]) return a[1] - b[1]; // 编号 小的在前
  return b[0] - a[0]; // 优先级 大的在前
};

let ans = Array.from({ length: 5 }, () => new PriorityQueue(compare));
let x = 1;

rl.on("line", (line) => {
  const tmp = line.split(" ");
  if (tmp.length === 1) return; // 忽略无效输入

  const index = parseInt(tmp[1]) - 1;
```

```
    if (tmp[0] === "IN") {
      ans[index].push([parseInt(tmp[2]), x]);
      x++;
    } else {
      if (ans[index].isEmpty()) {
        console.log("NULL");
      } else {
        console.log(ans[index].pop()[1]);
      }
    }
  });
```

Go

Plain Text

```
package main

import (
  "container/heap"
  "bufio"
  "fmt"
  "os"
  "strconv"
  "strings"
)

// 定义优先队列结构体
type Pair struct {
  first  int
  second int
}

// 定义堆结构（实现 container/heap 接口）
type PriorityQueue []Pair

func (pq PriorityQueue) Len() int { return len(pq) }

// 实现排序逻辑
// `优先级` 大的在前，`优先级` 相同的 `编号` 小的在前
func (pq PriorityQueue) Less(i, j int) bool {
  if pq[i].first == pq[j].first {
    return pq[i].second < pq[j].second
  }
  return pq[i].first > pq[j].first
}

func (pq PriorityQueue) Swap(i, j int) { pq[i], pq[j] = pq[j], pq[i] }

func (pq *PriorityQueue) Push(x interface{}) {
  *pq = append(*pq, x.(Pair))
}

func (pq *PriorityQueue) Pop() interface{} {
  old := *pq
  n := len(old)
  item := old[n-1]
  *pq = old[0 : n-1]
  return item
}

```

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

  // 创建 5 个优先队列
  ans := make([]PriorityQueue, 5)
  for i := range ans {
    heap.Init(&ans[i])
  }

  x := 1
  for i := 0; i < n; i++ {
    scanner.Scan()
    tmp := strings.Split(scanner.Text(), " ")
    index, _ := strconv.Atoi(tmp[1])
    index--

    if tmp[0] == "IN" {
      first, _ := strconv.Atoi(tmp[2])
      heap.Push(&ans[index], Pair{first, x})
      x++
    } else {
      if len(ans[index]) == 0 {
        fmt.Println("NULL")
      } else {
        fmt.Println(heap.Pop(&ans[index]).(Pair).second)
      }
    }
  }
}
```

> 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<=N<=100），有M（3<=M<=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<=M<=10）个评委，第二个数字表示N（3<=N<=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

```
4,5
10,6,9,7,6
9,10,6,7,5
8,10,6,5,10
9,10,8,4,9
```

输出

```
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

```
2,5
7,3,5,4,2
8,5,4,4,3
```

输出

```
-1
```

说明

> 只有2个评委，要求最少为3个评委。

用例3

输入

```
4,2
8,5
5,6
10,4
8,9
```

输出

```
-1
```

说明

> 只有两名选手参加

用例4

输入

```
4,5
11,6,9,7,8
9,10,6,7,8
8,10,6,9,7
9,10,8,6,7
```

输出

```
-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++

Plain Text

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

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

struct Person {
    // 记录选手获得[1-10]各个分数的数量
    vector<int> score;
    // 统计用户的获得的总分
    int sum;
    // 记录选手下表
    int index;
    Person() : score(11, 0), sum(0), index(0) {}
};

// 按照sum进行从大到小排序，sum相等按照score从大到小排序
bool cmp(Person a, Person b) {
  if (a.sum == b.sum) {
    for (int i = 10; i >= 0; i--) {
      if (a.score[i] != b.score[i]) {
        return a.score[i] > b.score[i];
      }
    }
  }
  return a.sum > b.sum;
}

```

```
int main() {
  int n, m;
  string line;
  getline(cin, line);
  vector<string> s = split(line, ",");
  n = stoi(s[1]);
  m = stoi(s[0]);
  // 判断数据合法性
  if (n < 3 || n > 100 || m < 3 || m > 10) {
    cout << -1;
    return 0;
  }
  vector<Person> p(n, Person());
  for (int i = 0; i < m; i++) {
    getline(cin, line);
    s = split(line, ",");
    for (int j = 0; j < s.size(); j++) {
      // 判断数据合法性
      if (stoi(s[j]) < 1 || stoi(s[j]) > 10) {
        cout << -1;
        return 0;
      }
      // 更新选手得到的分数情况
      p[j].score[stoi(s[j])] =  p[j].score[stoi(s[j])] + 1;
      p[j].sum += stoi(s[j]);
      p[j].index = j;
    }
  }
  // 根据自定义规则进行排序
  sort(p.begin(), p.end(), cmp);
  // 输出结果
  for (int i = 0; i < 3; i++) {
    if (i != 0) {
      cout << ",";
    }
    cout << p[i].index + 1;
  }
  return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {


    static class Person {
        int[] score = new int[11]; // 分数范围是 1 到 10，所以需要 11 个位置
        int sum = 0;
        int index = 0;

        public Person() {
            Arrays.fill(score, 0);
        }
    }

    // 按照 sum 从大到小排序；如果 sum 相等，则按 score 从大到小排序
    public static int cmp(Person a, Person b) {
        if (a.sum == b.sum) {
            for (int i = 10; i >= 1; i--) {
                if (a.score[i] != b.score[i]) {
                    return a.score[i] > b.score[i] ?  -1 : 1;
                }
            }
        }
        return a.sum > b.sum ?  -1 : 1;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String line = scanner.nextLine();
        String[] s = line.split(",");

        int n = Integer.parseInt(s[1]);
        int m = Integer.parseInt(s[0]);

        if (n < 3 || n > 100 || m < 3 || m > 10) {
            System.out.println(-1);
            return;
        }

        List<Person> persons = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            persons.add(new Person());
        }

```

```
        for (int i = 0; i < m; i++) {
            line = scanner.nextLine();
            s = line.split(",");

            for (int j = 0; j < s.length; j++) {
                int score = Integer.parseInt(s[j]);
                if (score < 1 || score > 10) {
                    System.out.println(-1);
                    return;
                }
                persons.get(j).score[score]++;
                persons.get(j).sum += score;
                persons.get(j).index = j;
            }
        }

        // 排序
        persons.sort((a, b) -> cmp(a, b) );

        // 输出前三名的 index
        for (int i = 0; i < 3; i++) {
            if (i != 0) {
                System.out.print(",");
            }
            System.out.print(persons.get(i).index + 1);
        }
    }
}
```

Python

Plain Text

```
class Person:
    def __init__(self):
        # 分数范围为 1 - 10 每个分数的数量
        self.score = [0] * 11
        # 总分
        self.sum = 0
        # 下标
        self.index = 0

def cmp(a, b):
    if a.sum == b.sum:
        for i in range(10, -1, -1):
            if a.score[i] != b.score[i]:
                return a.score[i] > b.score[i]
    return a.sum > b.sum

def main():
    line = input()
    s = line.split(',')
    m, n = int(s[0]), int(s[1])

    # 判断合法性
    if not (3 <= n <= 100 and 3 <= m <= 10):
        print(-1)
        return

    p = [Person() for _ in range(n)]

    for i in range(m):
        line = input()
        s = line.split(',')
        for j in range(len(s)):
            score = int(s[j])
            # 判断合法性
            if not (1 <= score <= 10):
                print(-1)
                return
            p[j].score[score] += 1
            p[j].sum += score
            p[j].index = j

    # 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的的逻辑
    p.sort(key=lambda person: (person.sum, person.score[::-1]), reverse=Tr
ue)

```

```
    # 输出结果
    print(",".join(str(p[i].index + 1) for i in range(3)))

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

JavaScript

Plain Text

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

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

let inputLines = [];

rl.on("line", (line) => {
    inputLines.push(line);
}).on("close", () => {
    const input = inputLines;

    const [m, n] = input[0].split(",").map(Number);

    // 判断数据合法性
    if (n < 3 || n > 100 || m < 3 || m > 10) {
        console.log(-1);
        return;
    }

    const players = Array.from({ length: n }, () => ({
        // 记录1 - 10分数的数量
        score: Array(11).fill(0),
        // 总分
        sum: 0,
        index: 0,
    }));

    for (let i = 1; i <= m; i++) {
        const scores = input[i].split(",").map(Number);

        for (let j = 0; j < scores.length; j++) {
            const score = scores[j];

            // 判断分数合法性
            if (score < 1 || score > 10) {
                console.log(-1);
                return;
            }

            players[j].score[score]++;
            players[j].sum += score;
            players[j].index = j;
```

```
        }
    }

    players.sort((a, b) => {
        if (a.sum !== b.sum) {
            return b.sum - a.sum;
        }
        // 总分相等按照分数排序
        for (let i = 10; i >= 1; i--) {
            if (a.score[i] !== b.score[i]) {
                return b.score[i] - a.score[i];
            }
        }
        return 0;
    });

    console.log(
        players
            .slice(0, 3)
            .map((player) => player.index + 1)
            .join(",")
    );
});
```

Go

Plain Text

```
package main

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



// Person 结构体
type Person struct {
  score []int // 记录选手获得[1-10]各个分数的数量
  sum   int   // 统计用户的获得的总分
  index int   // 记录选手下标
}

// 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
func cmp(a, b Person) bool {
  if a.sum == b.sum {
    for i := 10; i >= 1; i-- {
      if a.score[i] != b.score[i] {
        return a.score[i] > b.score[i]
      }
    }
  }
  return a.sum > b.sum
}

func main() {
  reader := bufio.NewReader(os.Stdin)

  // 读取第一行输入
  line, _ := reader.ReadString('\n')
  line = strings.TrimSpace(line)
  inputs := strings.Split(line, ",")

  m, _ := strconv.Atoi(inputs[0])
  n, _ := strconv.Atoi(inputs[1])

  // 判断数据合法性
  if n < 3 || n > 100 || m < 3 || m > 10 {
```

```
    fmt.Println(-1)
    return
  }

  // 初始化选手数据
  players := make([]Person, n)
  for i := 0; i < n; i++ {
    players[i] = Person{
      score: make([]int, 11),
      sum:   0,
      index: i,
    }
  }

  // 读取每轮分数
  for i := 0; i < m; i++ {
    line, _ := reader.ReadString('\n')
    line = strings.TrimSpace(line)
    scores := strings.Split(line, ",")

    for j, scoreStr := range scores {
      score, _ := strconv.Atoi(scoreStr)

      // 判断分数合法性
      if score < 1 || score > 10 {
        fmt.Println(-1)
        return
      }

      players[j].score[score]++
      players[j].sum += score
    }
  }

  // 排序选手
  sort.Slice(players, func(i, j int) bool {
    return cmp(players[i], players[j])
  })

  // 输出前三名选手的下标
  for i := 0; i < 3; i++ {
    if i > 0 {
      fmt.Print(",")
    }
    fmt.Print(players[i].index + 1)
  }
}
```

> 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卷真题题库目录丨机考题库 + 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

```
20*19*20*
```

输出

```
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

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

int main() {
  string input;
  getline(cin, input);
  // 正则替换优先选择替换 10* - 26*
  for (int i = 26; i >= 1; i--) {
    string regexStr = to_string(i);
    if (i > 9) {
      regexStr += "\\*";
    }
    regex pattern(regexStr);
    string replaceChar =  string(1, 'a' + i -1);
    input = regex_replace(input, pattern, replaceChar);
  }
  cout << input;
  return 0;
}
```

JAVA

Plain Text

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

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

        // 正则替换优先选择替换 10* - 26*
        for (int i = 26; i >= 1; i--) {
            String regexStr = Integer.toString(i);
            if (i > 9) {
                regexStr += "\\*";  // 需要转义 *
            }
            Pattern pattern = Pattern.compile(regexStr);
            String replaceChar = String.valueOf((char) ('a' + i - 1));
            Matcher matcher = pattern.matcher(input);
            input = matcher.replaceAll(replaceChar);
        }

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

Python

```
Plain Text
```

```
import re

def main():
    input_str = input()

    # 正则替换优先选择替换 10* - 26*
    for i in range(26, 0, -1):
        regex_str = str(i)
        if i > 9:
            regex_str += r"\*"  # 转义 *
        replace_char = chr(ord('a') + i - 1)
        input_str = re.sub(regex_str, replace_char, input_str)

    print(input_str)

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

JavaScript

```
Plain Text
```

```
function main() {
    const fs = require("fs");
    let input = fs.readFileSync(0, "utf-8").trim();

    // 正则替换优先选择替换 10* - 26*
    for (let i = 26; i >= 1; i--) {
        let regexStr = i.toString();
        if (i > 9) {
            regexStr += "\\*";  // 需要转义 *
        }
        const pattern = new RegExp(regexStr, "g");
        const replaceChar = String.fromCharCode('a'.charCodeAt(0) + i - 1);
        input = input.replace(pattern, replaceChar);
    }

    console.log(input);
}

main();
```

Go

```
Plain Text
```

```
package main

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

func main() {
  reader := bufio.NewReader(os.Stdin)
  input, _ := reader.ReadString('\n')
  input = input[:len(input)-1] // 去除换行

  // 正则替换优先选择替换 10* - 26*
  for i := 26; i >= 1; i-- {
    regexStr := strconv.Itoa(i)
    if i > 9 {
      regexStr += "\\*" // 转义 *
    }
    re := regexp.MustCompile(regexStr)
    replaceChar := string('a' + i - 1)
    input = re.ReplaceAllString(input, replaceChar)
  }

  fmt.Println(input)
}
```

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。

Plain Text

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

int main() {
    string input;
    getline(cin, input);
    string res = "";
    stack<char> stk;
    int n = input.size();
    for (int i = 0; i < n; i++) {
        // 压栈
        if (input[i] != '*') {
            stk.push(input[i]);
        } else {
            // 遇到*出栈，并弹出两位
            char c1 = stk.top();
            stk.pop();
            char c2 = stk.top();
            stk.pop();
            string prefix = "";
            // 栈中剩余单个的数字都是 属于 a - i
            while (!stk.empty()) {
                char c = stk.top();
                stk.pop();
                prefix.push_back('a' + (c - '1'));
            }
            if (!prefix.empty()) {
                reverse(prefix.begin(), prefix.end());
            }

            string tmp = string() + c2 + c1;
            prefix.push_back('j' + (stoi(tmp) - 10));
            res += prefix;
        }
    }

    // 收尾操作
    if (!stk.empty()) {
        string prefix = "";
```

```
            while (!stk.empty()) {
                char c = stk.top();
                stk.pop();
                prefix.push_back('a' + (c - '1'));
            }
            reverse(prefix.begin(), prefix.end());
            res += prefix;
        }
        cout << res;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String input = scanner.nextLine();
        StringBuilder res = new StringBuilder();
        Stack<Character> stk = new Stack<>();

        int n = input.length();
        for (int i = 0; i < n; i++) {
            // 压栈
            if (input.charAt(i) != '*') {
                stk.push(input.charAt(i));
            } else {
                // 遇到*出栈，并弹出两位
                char c1 = stk.pop();
                char c2 = stk.pop();
                StringBuilder prefix = new StringBuilder();

                // 栈中剩余单个的数字都是属于 a - i
                while (!stk.isEmpty()) {
                    char c = stk.pop();
                    prefix.append((char) ('a' + (c - '1')));
                }

                if (prefix.length() > 0) {
                    prefix.reverse();
                }

                String tmp = "" + c2 + c1;
                prefix.append((char) ('j' + (Integer.parseInt(tmp) - 1
0)));

                res.append(prefix);
            }
        }

        // 收尾操作
        if (!stk.isEmpty()) {
            StringBuilder prefix = new StringBuilder();
            while (!stk.isEmpty()) {
                char c = stk.pop();
                prefix.append((char) ('a' + (c - '1')));
            }
```

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

Python

Plain Text

```
def main():
    input_str = input()
    res = []
    stk = []

    for ch in input_str:
        # 压栈
        if ch != '*':
            stk.append(ch)
        else:
            # 遇到*出栈，并弹出两位
            c1 = stk.pop()
            c2 = stk.pop()
            prefix = []

            # 栈中剩余单个的数字都是属于 a - i
            while stk:
                c = stk.pop()
                prefix.append(chr(ord('a') + int(c) - 1))

            prefix.reverse()

            tmp = c2 + c1
            prefix.append(chr(ord('j') + int(tmp) - 10))
            res.extend(prefix)

    # 收尾操作
    if stk:
        prefix = []
        while stk:
            c = stk.pop()
            prefix.append(chr(ord('a') + int(c) - 1))
        prefix.reverse()
        res.extend(prefix)

    print("".join(res))

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

JavaScript

Plain Text

```
function main() {
    const fs = require("fs");
    const input = fs.readFileSync(0, "utf-8").trim();
    const stk = [];
    let res = "";

    for (let i = 0; i < input.length; i++) {
        if (input[i] !== '*') {
            stk.push(input[i]);
        } else {
            // 遇到*出栈，并弹出两位
            const c1 = stk.pop();
            const c2 = stk.pop();
            const prefix = [];

            // 栈中剩余单个的数字都是属于 a - i
            while (stk.length > 0) {
                const c = stk.pop();
                prefix.push(String.fromCharCode('a'.charCodeAt(0) + Number(c) - 1));
            }

            prefix.reverse();

            const tmp = c2 + c1;
            prefix.push(String.fromCharCode('j'.charCodeAt(0) + Number(tmp) - 10));
            res += prefix.join("");
        }
    }

    // 收尾操作
    if (stk.length > 0) {
        const prefix = [];
        while (stk.length > 0) {
            const c = stk.pop();
            prefix.push(String.fromCharCode('a'.charCodeAt(0) + Number(c) - 1));
        }
        prefix.reverse();
        res += prefix.join("");
    }

    console.log(res);
}
```

```
43
44
    main();
```

Go

Plain Text

```
package main

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

func main() {
  var input string
  reader := bufio.NewReader(os.Stdin)
  input, _ = reader.ReadString('\n')
  input = input[:len(input)-1] // 去掉换行符

  var stk []byte
  var res []byte

  for i := 0; i < len(input); i++ {
    if input[i] != '*' {
      stk = append(stk, input[i])
    } else {
      // 遇到*出栈，并弹出两位
      c1 := stk[len(stk)-1]
      stk = stk[:len(stk)-1]
      c2 := stk[len(stk)-1]
      stk = stk[:len(stk)-1]

      var prefix []byte
      // 栈中剩余单个的数字都是属于 a - i
      for len(stk) > 0 {
        c := stk[len(stk)-1]
        stk = stk[:len(stk)-1]
        prefix = append(prefix, byte('a'+(c-'1')))
      }

      // 反转 prefix
      for i, j := 0, len(prefix)-1; i < j; i, j = i+1, j-1 {
        prefix[i], prefix[j] = prefix[j], prefix[i]
      }

      tmp, _ := strconv.Atoi(string([]byte{c2, c1}))
      prefix = append(prefix, byte('j'+(tmp-10)))
      res = append(res, prefix...)
    }
```

```
    }

    // 收尾操作
    if len(stk) > 0 {
      var prefix []byte
      for len(stk) > 0 {
        c := stk[len(stk)-1]
        stk = stk[:len(stk)-1]
        prefix = append(prefix, byte('a'+(c-'1')))
      }
      // 反转 prefix
      for i, j := 0, len(prefix)-1; i < j; i, j = i+1, j-1 {
        prefix[i], prefix[j] = prefix[j], prefix[i]
      }
      res = append(res, prefix...)
    }

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

> 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

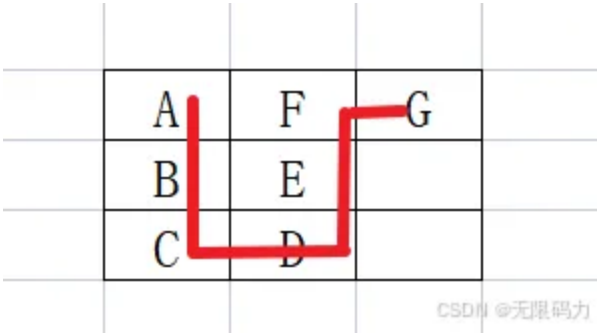
箱子Z字形摆放

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ABCDEFG，空地宽度为3，摆放结果如图：



则输出结果为：

```
`AFG`
`BE`
`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：

```
ABCDEFG 3
```

输出描述

箱子摆放结果，如题目示例所示

```
AFG
BE
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 <= len(str) <= 1000
3. 1 <= n <= 1000

用例1

输入

```
ABCDEFG 3
```

输出

```
AFG
BE
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

```
{
  {},
  {},
  {},
}
```

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

```
{
  {A},
  {B},
  {C},
}
```

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))。

```
{
  {A},
  {B},
  {CD},
}
```

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++

Plain Text

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

// 按照之字形方式填充矩阵并输出
void zigzagFill(const string& str, int n) {
    // 创建矩阵，每行是一个 vector<char>
    vector<vector<char>> matrix(n);

    bool reverse = true;  // 控制方向
    for (int i = 0; i < str.length(); i++) {
        int k = i % n;  // 计算当前字符所在的行号
        if (k == 0) reverse = !reverse;  // 每当到达行的起点时，反转方向

        // 如果是反向填充，调整行号
        if (reverse) k = n - 1 - k;

        matrix[k].push_back(str[i]);
    }

    // 输出矩阵内容
    for (const auto& row : matrix) {
        for (char c : row) {
            cout << c;
        }
        cout << endl;  // 输出换行
    }

}

int main() {
    string str;
    int n;
    cin >> str >> n;

    zigzagFill(str, n);

    return 0;
}
```

JAVA

Plain Text

```
import java.util.*;

public class Main {

    // 按照之字形方式填充矩阵并输出
    public static void zigzagFill(String str, int n) {
        // 创建矩阵，每行是一个 List<Character>
        List<List<Character>> matrix = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            matrix.add(new ArrayList<>());
        }

        boolean reverse = true;  // 控制方向
        for (int i = 0; i < str.length(); i++) {
            int k = i % n;  // 计算当前字符所在的行号
            if (k == 0) reverse = !reverse;  // 每当到达行的起点时，反转方向

            // 如果是反向填充，调整行号
            if (reverse) k = n - 1 - k;

            matrix.get(k).add(str.charAt(i));
        }

        // 输出矩阵内容
        for (List<Character> row : matrix) {
            for (char c : row) {
                System.out.print(c);
            }
            System.out.println();  // 输出换行
        }

    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        String str = scanner.next();
        int n = scanner.nextInt();
        zigzagFill(str, n);
    }
}
```

Python

Plain Text

```
# 按照之字形方式填充矩阵并输出
def zigzag_fill(s, n):
    # 创建矩阵，每行是一个列表
    matrix = [[] for _ in range(n)]

    reverse = True  # 控制方向
    for i in range(len(s)):
        k = i % n  # 计算当前字符所在的行号
        if k == 0:
            reverse = not reverse  # 每当到达行的起点时，反转方向

        # 如果是反向填充，调整行号
        if reverse:
            k = n - 1 - k

        matrix[k].append(s[i])

    # 输出矩阵内容
    for row in matrix:
        for c in row:
            print(c, end="")
        print()  # 输出换行


if __name__ == "__main__":
    s, n = input().split()
    n = int(n)
    zigzag_fill(s, n)
```

JavaScript

Plain Text

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

// 创建接口来读取输入
const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

// 按照之字形方式填充矩阵并输出
function zigzagFill(str, n) {
    // 创建一个二维数组表示矩阵，每行是一个数组
    let matrix = Array.from({ length: n }, () => []);


    let reverse = true;  // 控制填充方向，开始时是正向

    // 遍历字符串中的每个字符，填充到对应的矩阵行中
    for (let i = 0; i < str.length; i++) {
        let k = i % n;
        if (k == 0) {
            reverse = !reverse;
        }
        if (reverse) {
            k = n - 1 - k;
        }
        matrix[k].push(str[i]);
    }

    // 输出矩阵内容
    for (let i = 0; i < n; i++) {
        console.log(matrix[i].join(''));  // 输出当前行的内容
    }
}

// 读取输入
rl.question('', (input) => {
    const strAndN = input.split(' ');  // 读取一行输入并分割
    const str = strAndN[0];  // 获取字符串部分
    const n = parseInt(strAndN[1], 10);  // 获取行数部分

    zigzagFill(str, n);
    rl.close();
});
```

Go

Plain Text

```
package main

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

// 按照之字形方式填充矩阵并输出
func zigzagFill(str string, n int) {
  // 如果 n 小于等于 0，直接返回
  if n <= 0 {
    fmt.Println("Invalid input: number of rows must be greater than 0")
    return
  }

  // 创建矩阵，每行是一个切片
  matrix := make([][]rune, n)

  reverse := true  // 控制方向
  for i := 0; i < len(str); i++ {
    k := i % n  // 计算当前字符所在的行号
    if k == 0 {
      reverse = !reverse // 每当到达行的起点时，反转方向
    }

    // 如果是反向填充，调整行号
    if reverse {
      k = n - 1 - k
    }

    matrix[k] = append(matrix[k], rune(str[i]))
  }

  // 输出矩阵内容
  for _, row := range matrix {
    for _, c := range row {
      fmt.Print(string(c))
    }
    fmt.Println() // 输出换行
  }

}

```

```
func main() {
  // 创建输入读取器
  reader := bufio.NewReader(os.Stdin)

  // 读取输入
  input, _ := reader.ReadString('\n')
  input = strings.TrimSpace(input)

  // 分割输入，获取字符串和数字
  var str string
  var n int
  fmt.Sscanf(input, "%s %d", &str, &n)

  // 调用填充函数
  zigzagFill(str, n)
}
```

> 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 ⩽ nums.length ⩽ 200
- −10^9 ⩽ nums[i] ⩽ 10^9
- −10^9 ⩽ target ⩽ 10^9
- 2 ⩽ k ⩽ 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

Plain Text

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

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++

Plain Text

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

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

int twoSumsolve(vector<int> ans,  int target, int index, long sum) {
    int l = index;
    int r = ans.size() - 1;
    // 记录成立的搭配方案
    int res = 0;
    while (l < r) {
        long tmp = sum  + ans[l] + ans[r];
        if (tmp < target) {
            l++;
        } else if (tmp > target) {
            r--;
        } else {
            res++;
            // 去重
            while (l + 1 < r && ans[l] == ans[l+1]) {
                l++;
            }
```

```
            // 去重
            while (r - 1 > l && ans[r] == ans[r-1]) {
                r--;
            }
            l++;
            r--;
        }
    }
    return res;
}


int  solve(vector<int> ans, int k , int target, int index, long sum) {
    int res = 0;
    // 直接使用两数之和判断
    if (k == 2) {
        return twoSumsolve(ans, target, index, sum);
    }

    for (int i = index; i <= ans.size() - k; i++) {
        // 剪枝
        if (ans[i] > 0 && sum + ans[i] > target) {
            break;
        }
        // 防止重复
        if (i > index && ans[i] == ans[i-1]) {
            continue;
        }
        res += solve(ans, k  - 1, target, i + 1, sum + ans[i]);
    }
    return res;
}


int main() {
    string input;
    getline(cin, input);

    int k;
    cin >> k;
    int target;
    cin >> target;
    vector<int> ans = split(input, " ");
    int n = ans.size();
    if (n < k) {
        cout << 0;
        return 0;
    }
```

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

JAVA

Plain Text

```
import java.util.*;

public class Main {

    // 两数之和逻辑
    static int twoSumSolve(List<Integer> nums, int target, int index, lon
g sum) {
        int l = index;
        int r = nums.size() - 1;
        int res = 0;
        while (l < r) {
            long tmp = sum + nums.get(l) + nums.get(r);
            if (tmp < target) {
                l++;
            } else if (tmp > target) {
                r--;
            } else {
                res++;
                int left = nums.get(l), right = nums.get(r);
                while (l < r && nums.get(l).equals(left)) l++;
                while (l < r && nums.get(r).equals(right)) r--;
            }
        }
        return res;
    }

    // k 数之和递归
    static int solve(List<Integer> nums, int k, int target, int index, lon
g sum) {
        if (k == 2) {
            return twoSumSolve(nums, target, index, sum);
        }
        int res = 0;
        for (int i = index; i <= nums.size() - k; i++) {
            if (nums.get(i) > 0 && sum + nums.get(i) > target) break;
            if (i > index && nums.get(i).equals(nums.get(i - 1))) continu
e;
            res += solve(nums, k - 1, target, i + 1, sum + nums.get(i));
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        String[] strNums = sc.nextLine().split(" ");
```

```
        int k = sc.nextInt();
        int target = sc.nextInt();

        List<Integer> nums = new ArrayList<>();
        for (String s : strNums) {
            nums.add(Integer.parseInt(s));
        }

        if (nums.size() < k) {
            System.out.println(0);
            return;
        }

        Collections.sort(nums);
        System.out.println(solve(nums, k, target, 0, 0));
    }
}
```

Python

Plain Text

```
def two_sum_solve(nums, target, index, total):
    # 两数之和：双指针处理
    l, r = index, len(nums) - 1
    res = 0
    while l < r:
        tmp = total + nums[l] + nums[r]
        if tmp < target:
            l += 1
        elif tmp > target:
            r -= 1
        else:
            res += 1
            # 去重
            left, right = nums[l], nums[r]
            while l < r and nums[l] == left:
                l += 1
            while l < r and nums[r] == right:
                r -= 1
    return res

def solve(nums, k, target, index, total):
    if k == 2:
        return two_sum_solve(nums, target, index, total)
    res = 0
    for i in range(index, len(nums) - k + 1):
        # 剪枝
        if nums[i] > 0 and total + nums[i] > target:
            break
        # 跳过重复
        if i > index and nums[i] == nums[i - 1]:
            continue
        res += solve(nums, k - 1, target, i + 1, total + nums[i])
    return res

def main():
    # 第一行：数组
    nums = list(map(int, input().strip().split()))
    k = int(input())      # 第二行：k
    target = int(input()) # 第三行：目标值

    if len(nums) < k:
        print(0)
        return

    nums.sort()  # 排序方便剪枝
```

```
    print(solve(nums, k, target, 0, 0))

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

JavaScript

Plain Text

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

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

let lines = [];
rl.on('line', function(line) {
    lines.push(line.trim());
    if (lines.length === 3) {
        main();
    }
});

// 两数之和逻辑：双指针+去重
function twoSumSolve(nums, target, index, total) {
    let l = index, r = nums.length - 1;
    let res = 0;
    while (l < r) {
        let tmp = total + nums[l] + nums[r];
        if (tmp < target) {
            l++;
        } else if (tmp > target) {
            r--;
        } else {
            res++;
            let left = nums[l], right = nums[r];
            while (l < r && nums[l] === left) l++;
            while (l < r && nums[r] === right) r--;
        }
    }
    return res;
}

// k 数之和递归
function solve(nums, k, target, index, total) {
    if (k === 2) return twoSumSolve(nums, target, index, total);
    let res = 0;
    for (let i = index; i <= nums.length - k; i++) {
        if (nums[i] > 0 && total + nums[i] > target) break;
        if (i > index && nums[i] === nums[i - 1]) continue;
        res += solve(nums, k - 1, target, i + 1, total + nums[i]);
    }
    return res;
```

```
}

// 主函数处理输入
function main() {
    const nums = lines[0].split(' ').map(Number);
    const k = parseInt(lines[1]);
    const target = parseInt(lines[2]);

    if (nums.length < k) {
        console.log(0);
        return;
    }

    nums.sort((a, b) => a - b);
    console.log(solve(nums, k, target, 0, 0));
}
```

Go

Plain Text

```
package main

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

// twoSumSolve 实现两数之和的逻辑，使用双指针并去重
func twoSumSolve(nums []int, target int, index int, sum int) int {
  left := index
  right := len(nums) - 1
  count := 0

  for left < right {
    total := sum + nums[left] + nums[right]
    if total < target {
      left++
    } else if total > target {
      right--
    } else {
      count++
      // 去重处理
      lv := nums[left]
      rv := nums[right]
      for left < right && nums[left] == lv {
        left++
      }
      for left < right && nums[right] == rv {
        right--
      }
    }
  }

  return count
}

// solve 实现 k 数之和的递归逻辑
func solve(nums []int, k int, target int, index int, sum int) int {
  if k == 2 {
    return twoSumSolve(nums, target, index, sum)
  }
```

```
    count := 0
    for i := index; i <= len(nums)-k; i++ {
      // 剪枝：如果当前数加上 sum 已经超过目标，后续不可能满足
      if i > index && nums[i] == nums[i-1] {
        continue // 去重
      }
      // 进一步剪枝优化（因为数组已排序）
      if nums[i] > 0 && sum+nums[i] > target {
        break
      }
      count += solve(nums, k-1, target, i+1, sum+nums[i])
    }
    return count
}

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

    // 读取第一行：一行空格分隔的整数
    scanner.Scan()
    line := scanner.Text()
    parts := strings.Fields(line)

    nums := make([]int, 0, len(parts))
    for _, s := range parts {
      num, _ := strconv.Atoi(s)
      nums = append(nums, num)
    }

    // 读取第二行：k
    scanner.Scan()
    k, _ := strconv.Atoi(scanner.Text())

    // 读取第三行：target
    scanner.Scan()
    target, _ := strconv.Atoi(scanner.Text())

    if len(nums) < k {
      fmt.Println(0)
      return
    }

    sort.Ints(nums)
    result := solve(nums, k, target, 0, 0)
    fmt.Println(result)
}
```

> 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: 点击去查看

> 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 ⩽ t ⩽ 50。
第二行为 t 个数字，代表每一分钟的得分 p， 1 ⩽ p ⩽ 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
9
5 2 1 5 2 1 5 2 1
```

输出

```
Plain Text
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 `sum(score) % x == 0` , 不满足直接返回false.
   b. 判断 `max(score) > (sum(score) / x)` , 每一场分数不能进行拆分，肯定不满足所有人都拿 `sum(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 `sum(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++

Plain Text

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

bool cmp(int x , int y) {
    return x > y;
}

bool backtrack(int index, int right, vector<int> & buckets, int subSum, ve
ctor<int> ans) {
    if (index == right) {
        for (int i = 0; i < buckets.size(); i++) {
            if (buckets[i] != subSum) {
                return false;
            }
        }
        return true;
    }

    for (int i = 0; i < buckets.size(); i++) {
         // 剪枝 防止重复运算
        if (i > 0 && buckets[i] == buckets[i - 1]) {
            continue;
        }

        if (buckets[i] + ans[index] > subSum) {
            continue;
        }
        // 递归
        buckets[i] += ans[index];
        if (backtrack(index + 1, right, buckets, subSum, ans)) {
            return true;
        }
        // 回溯
        buckets[i] -= ans[index];

    }
    return false;
}

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;
```

```
            }
            n--;
        }
        cout << sum / n;
        return 0;
    }
```

JAVA

Plain Text

```
import java.util.*;

public class Main {

    // 比较函数，用于降序排序
    public static boolean cmp(int x, int y) {
        return x > y;
    }

    // 回溯函数
    public static boolean backtrack(int index, int right, int[] buckets, int subSum, List<Integer> ans) {
        if (index == right) {
            for (int bucket : buckets) {
                if (bucket != subSum) {
                    return false;
                }
            }
            return true;
        }

        for (int i = 0; i < buckets.length; i++) {
            // 剪枝：防止重复运算
            if (i > 0 && buckets[i] == buckets[i - 1]) {
                continue;
            }

            if (buckets[i] + ans.get(index) > subSum) {
                continue;
            }
            // 递归
            buckets[i] += ans.get(index);
            if (backtrack(index + 1, right, buckets, subSum, ans)) {
                return true;
            }
            // 回溯
            buckets[i] -= ans.get(index);
        }
        return false;
    }

    // 判断是否可以进行分配
    public static boolean canPartition(List<Integer> ans, int n, int sum) {
        if (sum % n != 0) {
```

```
            return false;
        }
        int subSum = sum / n;
        if (ans.get(0) > subSum) {
            return false;
        }

        int beginIndex = 0;
        // 预处理数据，减少数据量
        while (beginIndex < ans.size() && ans.get(beginIndex) == subSum) {
            n--;
            beginIndex++;
        }

        if (n == 0) {
            return true;
        }

        int[] buckets = new int[n];
        return backtrack(beginIndex, ans.size(), buckets, subSum, ans);
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        List<Integer> ans = new ArrayList<>();
        int sum = 0;
        for (int i = 0; i < n; i++) {
            int score = scanner.nextInt();
            ans.add(score);
            sum += score;
        }

        // 将分数按降序排列
        ans.sort((x, y) -> y - x);

        // 从大到小枚举获取MVP人数，范围为 1 - m
        while (n != 1) {
            if (canPartition(ans, n, sum)) {
                break;
            }
            n--;
        }
        System.out.println(sum / n);
    }
}
```

Python

Plain Text

```
def cmp(x, y):
    return x > y

def backtrack(index, right, buckets, sub_sum, ans, memo):
    if index == right:
        # 所有桶的值都已经达到 sub_sum
        return all(bucket == sub_sum for bucket in buckets)

    # 使用 memo 来缓存已经尝试过的状态
    state = tuple(buckets)
    if state in memo:
        return False
    memo.add(state)

    for i in range(len(buckets)):
        # 剪枝：避免重复运算
        if i > 0 and buckets[i] == buckets[i - 1]:
            continue

        if buckets[i] + ans[index] > sub_sum:
            continue

        # 递归
        buckets[i] += ans[index]
        if backtrack(index + 1, right, buckets, sub_sum, ans, memo):
            return True
        # 回溯
        buckets[i] -= ans[index]

    return False

def can_partition(ans, n, total_sum):
    if total_sum % n != 0:
        return False

    sub_sum = total_sum // n
    if ans[0] > sub_sum:
        return False

    begin_index = 0
    while begin_index < len(ans) and ans[begin_index] == sub_sum:
        n -= 1
        begin_index += 1

    if n == 0:
```

```
            return True

        buckets = [0] * n
        memo = set()  # 用于缓存已经遍历的桶状态
        return backtrack(begin_index, len(ans), buckets, sub_sum, ans, memo)

    def main():
        n = int(input())
        ans = list(map(int, input().split()))
        total_sum = sum(ans)

        # 降序排序
        ans.sort(reverse=True)

        # 从大到小枚举获取MVP人数
        while n != 1:
            if can_partition(ans, n, total_sum):
                break
            n -= 1

        print(total_sum // n)

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

JavaScript

Plain Text

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

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

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

  if (lines.length === 2) {
    const n = parseInt(lines[0], 10);
    const arr = lines[1].split(" ").map(Number);

    console.log(findMaxAverage(arr, n));
    lines.length = 0;
  }
});

function findMaxAverage(arr, n) {
  // 计算总和并从大到小排序
  const sum = arr.sort((a, b) => b - a).reduce((acc, val) => acc + val,
0);

  let count = n;
  // 从n开始枚举MVP人数
  while (count >= 1) {
    if (isPartitionPossible([...arr], sum, count)) {
      return sum / count;
    }
    count--;
  }
}

function isPartitionPossible(arr, sum, count) {
  // 如果总和不能均分
  if (sum % count !== 0) return false;

  const subSum = sum / count;
  // 如果最大的分数超过每个人应该得到的分数，直接返回false
  if (subSum < arr[0]) return false;

  // 剔除已经等于subSum的值，减少数据量
  while (arr[0] === subSum) {
```

```
    arr.shift();
    count--;
  }

  const buckets = new Array(count).fill(0);
  return canPartitionRecursively(0, arr, subSum, buckets);
}

function canPartitionRecursively(index, arr, subSum, buckets) {
  if (index === arr.length) {
    // 所有元素都已经分配完
    return true;
  }

  const current = arr[index];

  for (let i = 0; i < buckets.length; i++) {
    // 剪枝：避免重复运算
    if (i > 0 && buckets[i] === buckets[i - 1]) continue;

    // 如果当前桶加入当前元素没有超过目标分数
    if (buckets[i] + current <= subSum) {
      buckets[i] += current;
      if (canPartitionRecursively(index + 1, arr, subSum, buckets)) retur
n true;
      // 回溯
      buckets[i] -= current;
    }
  }

  return false;
}
```

Go

Plain Text

```
package main

import (
  "fmt"
  "sort"
)

func cmp(x, y int) bool {
  return x > y
}

func backtrack(index, right int, buckets []int, subSum int, ans []int) bool {
  if index == right {
    for _, bucket := range buckets {
      if bucket != subSum {
        return false
      }
    }
    return true
  }

  for i := 0; i < len(buckets); i++ {
    // 剪枝，避免重复运算
    if i > 0 && buckets[i] == buckets[i-1] {
      continue
    }

    if buckets[i]+ans[index] > subSum {
      continue
    }

    // 递归
    buckets[i] += ans[index]
    if backtrack(index+1, right, buckets, subSum, ans) {
      return true
    }
    // 回溯
    buckets[i] -= ans[index]
  }

  return false
}

func canPartition(ans []int, n, totalSum int) bool {
```

```
    // 如果不能整除，直接返回 false
    if totalSum%n != 0 {
      return false
    }

    // 每个人应该得到的分数
    subSum := totalSum / n
    size := len(ans)

    // 如果最大的得分大于 subSum，直接返回 false
    if ans[0] > subSum {
      return false
    }

    beginIndex := 0

    // 预处理数据，减少计算量
    for beginIndex < size && ans[beginIndex] == subSum {
      n--
      beginIndex++
    }

    if n == 0 {
      return true
    }

    buckets := make([]int, n)
    return backtrack(beginIndex, size, buckets, subSum, ans)
  }

  func main() {
    var n int
    fmt.Scan(&n) // 输入人数
    ans := make([]int, n)
    totalSum := 0

    // 输入所有的得分并计算总分
    for i := 0; i < n; i++ {
      fmt.Scan(&ans[i])
      totalSum += ans[i]
    }

    // 将得分按降序排序
    sort.Slice(ans, func(i, j int) bool {
      return cmp(ans[i], ans[j])
    })

    // 从大到小枚举获取 MVP 的人数
```

```
    for n != 1 {
      if canPartition(ans, n, totalSum) {
        break
      }
      n--
    }

    // 输出每个人的分数
    fmt.Println(totalSum / n)
}
```

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