【华为OD机考 统一考试机试C卷】MELON的难题（C++ Java Python javaScript C语言）

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷

目前在考C卷，经过两个月的收集整理，C卷真题已基本整理完毕
抽到原题的概率为2/3到3/3，也就是最少抽到两道原题。请注意：大家刷完C卷真题，最好要把B卷的真题刷一下，因为C卷的部分真题来自B卷。
另外订阅专栏还可以联系笔者开通在线 OJ 进行刷题，提高刷题效率。
真题目录：华为OD机考机试 真题目录（C卷 + D卷 + B卷 + A卷） + 考点说明
专栏：2023华为OD机试(B卷+C卷+D卷)（C++JavaJSPy）
华为OD面试真题精选：华为OD面试真题精选
在线OJ：点击立即刷题，模拟真实机考环境

题目描述：MELON的难题（本题200分）

MELON有一堆精美的雨花石（数量为n，重量各异），准备送给S和W。MELON希望送给俩人的雨花石重量一致，请你设计一个程序，帮MELON确认是否能将雨花石平均分配。

输入描述

第1行输入为雨花石个数: n，0 < n < 31.
第2行输入为空格分割的各雨花石重量: m[0] m[1] … m[n - 1]， 0 < m[k] < 1001

不需要考虑异常输入的情况。

输出描述

如果可以均分，从当前雨花石中最少拿出几块，可以使两堆的重量相等:如果不能均分，则输出-1。

用例1

输入

```
4
1 1 2 2
```

输出

```
2
```

说明

> 输入第一行代表共4颗雨花石，第二行代表4颗雨花石重量分别为1、1、2、2。均分时只能分别为1,2，需要拿出重量为1和2的两块雨花石，所以输出2。

用例2

输入

```
10
1 1 1 1 1 9 8 3 7 10
```

输出

```
3
```

说明

> 输入第一行代表共10颗雨花石，第二行代表4颗雨花石重量分别为1、1、1、1、1、9、8、3、7、10 。
> 均分时可以1,1,1,1,1,9,7和10,8,3，也可以1,1,1,1,9.8和10,7,3,1，或者其他均分方式，但第一种只需要拿出重量为10.8,3的3块雨花石，第二种需要拿出4块，所以输出3(块数最少)。

01背包问题的思路：

题目要求将雨花石平均分配，即找到一种方法，使得从雨花石中拿出最少的数量，使得两堆雨花石的重量相等。这个问题可以转化为一个01背包问题：从给定的雨花石中选取一些，使得它们的重量之和等于总重量的一半，且选取的雨花石数量最少。

01背包问题的核心思路是使用动态规划。具体步骤如下：

计算所有雨花石的总重量。如果总重量为奇数，那么无法将雨花石平均分配，直接输出-1。

如果总重量为偶数，我们的目标是找到一些雨花石，使得它们的重量之和等于总重量的一半。定义一个动态规划数组dp，其中dp[j]表示从雨花石中选取一些，使得它们的重量之和为j时，所需的最少雨花石数量。

初始化dp数组，将除了dp之外的其他元素设置为n，表示最坏情况下需要拿出所有雨花石。

遍历每一块雨花石，对于每一块雨花石，从targetWeight开始递减，更新dp数组。如果使用当前雨花石能够减少所需雨花石数量，则更新dp[j]。

最后，检查dp[targetWeight]的值。如果它等于n，表示无法找到满足条件的雨花石组合，输出-1。否则，输出dp[targetWeight]，表示从当前雨花石中最少拿出几块，可以使两堆的重量相等。

C++

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
    int n;
    cin >> n;  // 输入雨花石个数
    vector<int> stones(n);
    for (int i = 0; i < n; i++) {
        cin >> stones[i];  // 输入雨花石重量
    }

    int totalWeight = 0;
    for (int stone : stones) {
        totalWeight += stone;  // 计算雨花石总重量
    }

    if (totalWeight % 2 != 0) {  // 如果总重量为奇数，无法均分
        cout << -1 << endl;
    } else {
        int targetWeight = totalWeight / 2;  // 目标重量为总重量的一半

        // 创建动态规划数组，dp[i]表示前i块雨花石中是否能够取出一些雨花石使得重量和为j
        vector<int> dp(targetWeight + 1, 0);

        // 初始化dp数组，将除了dp[0]之外的所有值设为n，表示最大需要拿出n块雨花石
        for (int i = 1; i <= targetWeight; i++) {
            dp[i] = n;
        }

```

```
        // 遍历每一块雨花石
        for (int i = 1; i <= n; i++) {
            int weight = stones[i - 1];
            // 更新dp数组，从后往前更新，避免重复使用同一块雨花石
            for (int j = targetWeight; j >= weight; j--) {
                // 如果当前重量可以由前面的雨花石组成，更新dp[j]为最小需要拿出的雨花石数量
                dp[j] = min(dp[j], dp[j - weight] + 1);
            }
        }

        // 如果dp[targetWeight]仍然等于n，表示无法均分雨花石
        if (dp[targetWeight] == n) {
            cout << -1 << endl;
        } else {
            // 输出最少需要拿出的雨花石数量
            cout << dp[targetWeight] << endl;
        }
    }

    return 0;
}
```

Java

```
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();  // 输入雨花石个数
        int[] stones = new int[n];
        for (int i = 0; i < n; i++) {
            stones[i] = scanner.nextInt();  // 输入雨花石重量
        }

        int totalWeight = 0;
        for (int stone : stones) {
            totalWeight += stone;  // 计算雨花石总重量
        }
```

```
        if (totalWeight % 2 != 0) {  // 如果总重量为奇数，无法均分
            System.out.println(-1);
        } else {
            int targetWeight = totalWeight / 2;  // 目标重量为总重量的一半

            // 创建动态规划数组，dp[i]表示前i块雨花石中是否能够取出一些雨花石使得重量和为j
            int[] dp = new int[targetWeight + 1];

            // 初始化dp数组，将除了dp[0]之外的其他元素设置为n，表示最坏情况下需要拿出所有雨花石
            for (int i = 1; i <= targetWeight; i++) {
                dp[i] = n;
            }

            // 遍历每一块雨花石
            for (int i = 1; i <= n; i++) {
                int weight = stones[i - 1];  // 当前雨花石的重量
                // 从目标重量开始递减，更新dp数组
                for (int j = targetWeight; j >= weight; j--) {
                    // 如果当前重量可以由前面的雨花石组成，更新dp[j]为最小需要拿出的雨花石数量
                    dp[j] = Math.min(dp[j], dp[j - weight] + 1);
                }
            }

            // 如果dp[targetWeight]仍然等于n，表示无法找到满足条件的雨花石组合
            if (dp[targetWeight] == n) {
                System.out.println(-1);
            } else {
                // 输出最少需要拿出的雨花石数量
                System.out.println(dp[targetWeight]);
            }
        }
    }
}
```

JavaScript

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

```

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

const inputLines = [];
// 当接收到一行输入时，将其添加到inputLines数组
rl.on('line', (line) => {
  inputLines.push(line);
  // 当接收到两行输入时，处理输入并关闭readline接口
  if (inputLines.length === 2) {
    processInput();
    rl.close();
  }
});

function processInput() {
  // 解析输入的雨花石数量和重量
  const n = parseInt(inputLines[0]);
  const stones = inputLines[1].split(' ').map(Number);

  // 计算雨花石总重量
  let totalWeight = 0;
  for (const stone of stones) {
    totalWeight += stone;
  }

  // 如果总重量不能被2整除，则无法平分
  if (totalWeight % 2 !== 0) {
    console.log(-1);
  } else {
    // 目标重量为总重量的一半
    const targetWeight = totalWeight / 2;

    // 初始化动态规划数组
    const dp = new Array(targetWeight + 1).fill(0);

    // 将除第一个元素外的其他元素设置为n
    for (let i = 1; i <= targetWeight; i++) {
```

```
      dp[i] = n;
    }

    // 遍历每个雨花石
    for (let i = 1; i <= n; i++) {
      const weight = stones[i - 1];
      // 更新动态规划数组
      for (let j = targetWeight; j >= weight; j--) {
        // 如果当前重量可以由前面的雨花石组成，更新dp[j]为最小需要拿出的雨花石数量
        dp[j] = Math.min(dp[j], dp[j - weight] + 1);
      }
    }

    // 如果dp[targetWeight]等于n，说明无法平分
    if (dp[targetWeight] === n) {
      console.log(-1);
    } else {
      // 输出最少需要拿出的雨花石数量
      console.log(dp[targetWeight]);
    }
  }
}

```

Python

```
# 输入雨花石个数
n = int(input())

# 输入雨花石重量，将输入的字符串转换为整数列表
stones = list(map(int, input().split()))

# 计算所有雨花石的总重量
totalWeight = 0
for stone in stones:
    totalWeight += stone

# 如果总重量为奇数，则无法平均分配，输出 -1
if totalWeight % 2 != 0:

```

```
    print(-1)
else:
    # 计算目标重量，即总重量的一半
    targetWeight = totalWeight // 2

    # 初始化动态规划数组 dp，长度为目标重量加 1
    dp = [0] * (targetWeight + 1)

    # 将 dp 数组的值从索引 1 开始设置为 n
    for i in range(1, targetWeight + 1):
        dp[i] = n

    # 遍历所有雨花石
    for i in range(1, n + 1):
        weight = stones[i - 1]
        # 更新 dp 数组的值
        for j in range(targetWeight, weight - 1, -1):
            # 如果当前重量可以由前面的雨花石组成，更新dp[j]为最小需要拿出的雨花石数量
            dp[j] = min(dp[j], dp[j - weight] + 1)

    # 如果 dp[targetWeight] 等于 n，说明无法平均分配，输出 -1
    if dp[targetWeight] == n:
        print(-1)
    else:
        # 输出最少需要拿出的雨花石数量，使两堆的重量相等
        print(dp[targetWeight])
```

C语言

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_N 31
#define MAX_WEIGHT 1001

int stones[MAX_N]; // 存储每块雨花石的重量
int dp[MAX_WEIGHT]; // 动态规划数组，用于记录达到某个重量的最小雨花石数量

// 求两个数中的较小值
```

```
int min(int a, int b) {
    return a < b ? a : b;
}

int main() {
    int n;
    scanf("%d", &n); // 输入雨花石个数

    int totalWeight = 0; // 雨花石总重量
    for (int i = 0; i < n; i++) {
        scanf("%d", &stones[i]); // 输入每块雨花石的重量
        totalWeight += stones[i]; // 累加总重量
    }

    // 如果总重量为奇数，无法均分
    if (totalWeight % 2 != 0) {
        printf("-1\n");
        return 0;
    }

    int targetWeight = totalWeight / 2; // 目标重量为总重量的一半

    // 初始化动态规划数组，dp[0]为0，其余为最大值n
    dp[0] = 0;
    for (int i = 1; i <= targetWeight; i++) {
        dp[i] = n;
    }

    // 动态规划求解
    for (int i = 0; i < n; i++) {
        for (int j = targetWeight; j >= stones[i]; j--) {
            // 更新dp数组，求取最小需要拿出的雨花石数量
            dp[j] = min(dp[j], dp[j - stones[i]] + 1);
        }
    }

    // 如果dp[targetWeight]仍然等于n，表示无法均分雨花石
    if (dp[targetWeight] == n) {
        printf("-1\n");
    } else {
```

```
        // 输出最少需要拿出的雨花石数量
        printf("%d\n", dp[targetWeight]);
    }

    return 0;
}
```

文章目录


机考真题
华为OD
No No No
CSDN @算法大师