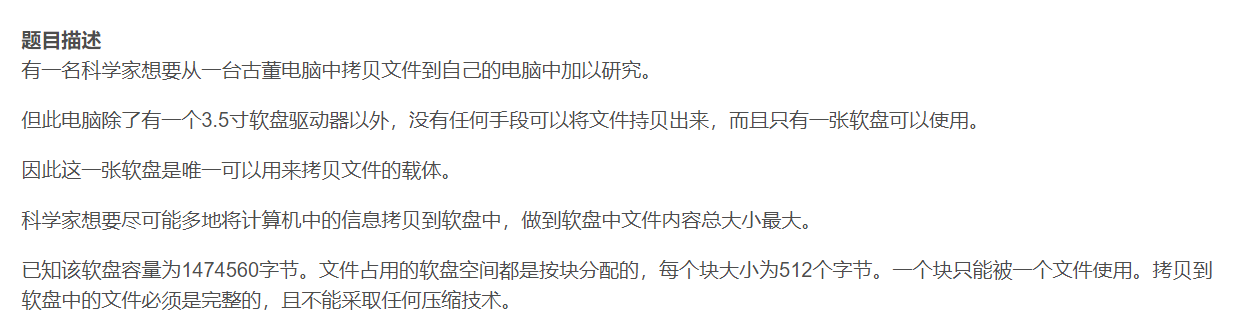
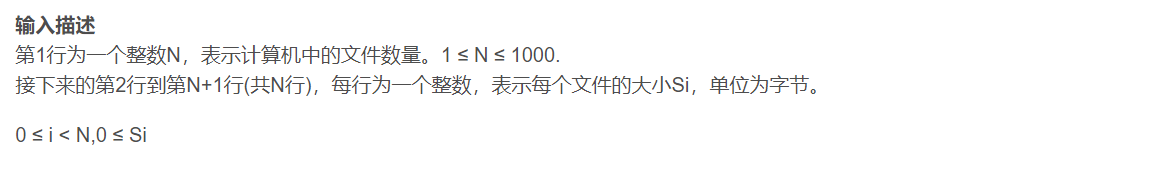
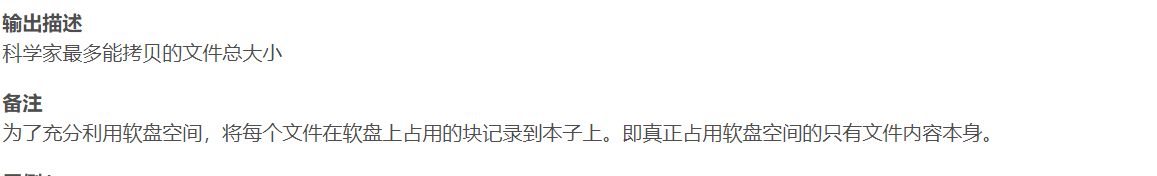
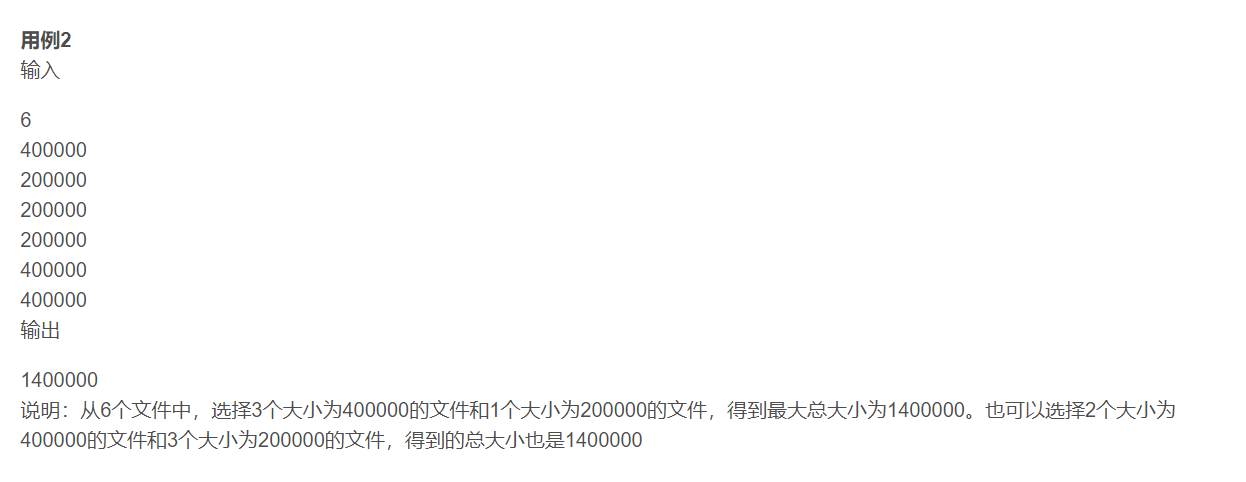
# **E卷-通过软盘拷贝文件[200分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**













import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int n = scanner.nextInt(); // 读取输入的物品数量

int Max = 2880; // 定义背包的容量为 2880

long[] dp = new long[Max + 1]; // 定义动态规划数组，长度为 Max + 1

// 遍历每个物品

for (int i = 1; i <= n; i++) {

int x = scanner.nextInt(); // 读取每个物品的体积

// 计算当前物品应该放入的最小块数（每块 512 单位大小）

int kuai = x / 512;

if (x % 512 != 0) {

kuai += 1; // 如果有余数则块数加 1

}

// 从最大容量到当前物品的块数，倒序更新 dp 数组

for (int j = Max; j >= kuai; j--) {

// 状态转移方程：选择当前物品或不选择当前物品

dp[j] = Math.max(dp[j], dp[j - kuai] + x);

}

}

// 输出在容量为 Max 的情况下所能放入的物品的最大体积

System.out.println(dp[Max]);

scanner.close();

}

}



# 读取输入的物品数量

n = int(input())

Max = 2880 # 定义背包的容量为 2880

dp = [0] \* (Max + 1) # 定义动态规划数组，长度为 Max + 1

# 遍历每个物品

for i in range(n):

x = int(input()) # 读取每个物品的体积

# 计算当前物品应该放入的最小块数（每块 512 单位大小）

kuai = x // 512

if x % 512 != 0:

kuai += 1 # 如果有余数则块数加 1

# 从最大容量到当前物品的块数，倒序更新 dp 数组

for j in range(Max, kuai - 1, -1):

# 状态转移方程：选择当前物品或不选择当前物品

dp[j] = max(dp[j], dp[j - kuai] + x)

# 输出在容量为 Max 的情况下所能放入的物品的最大体积

print(dp[Max])



#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

int main() {

int n;

cin >> n; // 读取输入的物品数量

int Max = 2880; // 定义背包的容量为 2880

vector<ll> dp(Max + 1); // 定义动态规划数组，长度为 Max + 1

// 遍历每个物品

for (int i = 1; i <= n; i++) {

int x;

cin >> x; // 读取每个物品的体积

// 计算当前物品应该放入的最小块数（每块 512 单位大小）

int kuai = x / 512;

if (x % 512 != 0) kuai += 1; // 如果有余数则块数加 1

// 从最大容量到当前物品的块数，倒序更新 dp 数组

for (int j = Max; j >= kuai; j--) {

// 状态转移方程：选择当前物品或不选择当前物品

dp[j] = max(dp[j], dp[j - kuai] + x);

}

}

// 输出在容量为 Max 的情况下所能放入的物品的最大体积

cout << dp[Max] << endl;

}



#include <stdio.h>

#include <string.h>

int main() {

int n;

scanf("%d", &n); // 读取输入的物品数量

int Max = 2880; // 定义背包的容量为 2880

int dp[Max + 1]; // 定义动态规划数组，长度为 Max + 1

memset(dp, 0, sizeof(dp)); // 初始化动态规划数组为 0

// 遍历每个物品

for (int i = 0; i < n; i++) {

int x;

scanf("%d", &x); // 读取每个物品的体积

// 计算当前物品应该放入的最小块数（每块 512 单位大小）

int kuai = x / 512;

if (x % 512 != 0) {

kuai += 1; // 如果有余数则块数加 1

}

// 从最大容量到当前物品的块数，倒序更新 dp 数组

for (int j = Max; j >= kuai; j--) {

// 状态转移方程：选择当前物品或不选择当前物品

if (dp[j] < dp[j - kuai] + x) {

dp[j] = dp[j - kuai] + x;

}

}

}

// 输出在容量为 Max 的情况下所能放入的物品的最大体积

printf("%d\n", dp[Max]);

return 0;

}



const readline = require('readline');

// 设置输入输出接口

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout

});

let inputLines = [];

let n, Max = 2880; // 定义背包的容量为 2880

let dp = Array(Max + 1).fill(0); // 定义动态规划数组，长度为 Max + 1

// 读取输入

rl.on('line', (line) => {

inputLines.push(line);

if (inputLines.length > 1 && inputLines.length === Number(inputLines[0]) + 1) {

n = Number(inputLines[0]);

for (let i = 1; i <= n; i++) {

let x = Number(inputLines[i]); // 读取每个物品的体积

// 计算当前物品应该放入的最小块数（每块 512 单位大小）

let kuai = Math.ceil(x / 512);

// 从最大容量到当前物品的块数，倒序更新 dp 数组

for (let j = Max; j >= kuai; j--) {

// 状态转移方程：选择当前物品或不选择当前物品

dp[j] = Math.max(dp[j], dp[j - kuai] + x);

}

}

// 输出在容量为 Max 的情况下所能放入的物品的最大体积

console.log(dp[Max]);

rl.close();

}

});



package main

import (

"fmt"

)

func main() {

var n int

fmt.Scan(&n) // 读取输入的物品数量

Max := 2880 // 定义背包的容量为 2880

dp := make([]int, Max+1) // 定义动态规划数组，长度为 Max + 1

// 遍历每个物品

for i := 0; i < n; i++ {

var x int

fmt.Scan(&x) // 读取每个物品的体积

// 计算当前物品应该放入的最小块数（每块 512 单位大小）

kuai := x / 512

if x%512 != 0 {

kuai += 1 // 如果有余数则块数加 1

}

// 从最大容量到当前物品的块数，倒序更新 dp 数组

for j := Max; j >= kuai; j-- {

// 状态转移方程：选择当前物品或不选择当前物品

if dp[j] < dp[j-kuai]+x {

dp[j] = dp[j-kuai] + x

}

}

}

// 输出在容量为 Max 的情况下所能放入的物品的最大体积

fmt.Println(dp[Max])

}