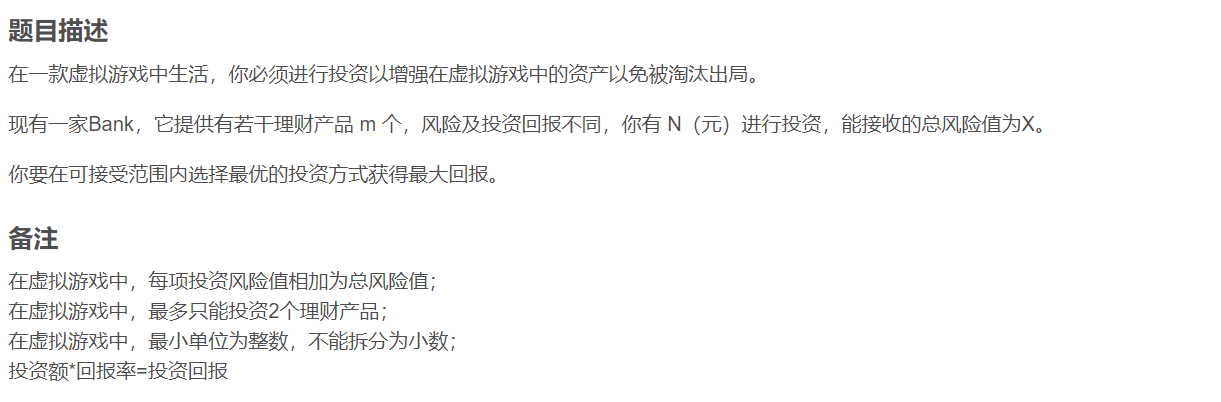
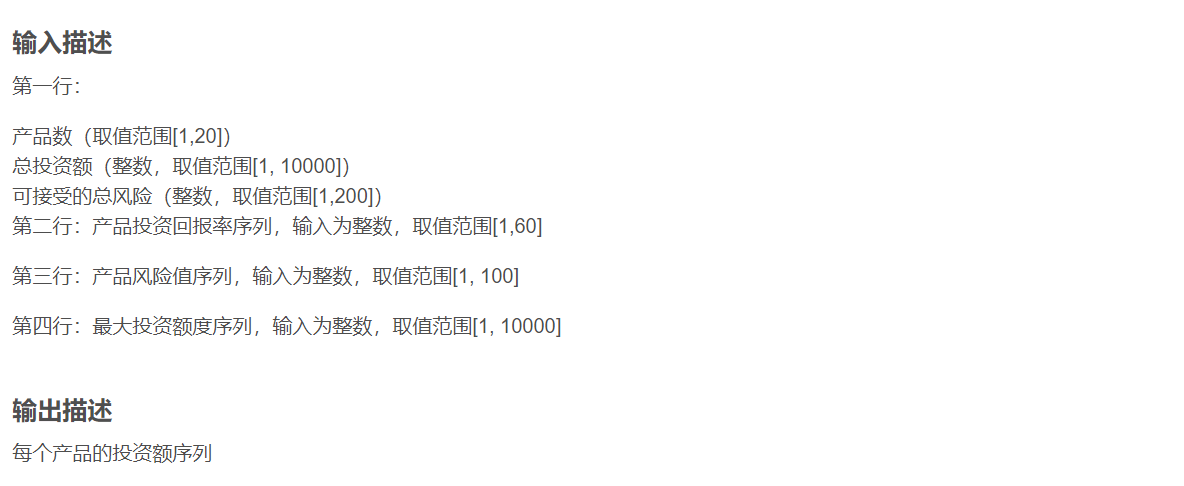
# **E卷-虚拟理财游戏[100分]（ Java | Python3 | C++ | C语言 | JsNode | Go）**









import java.util.Scanner;

public class Main {

static int[] qian = new int[500005]; // 产品投资回报率序列

static int[] danger = new int[500005]; // 产品风险值序列

static int[] cnt = new int[500005]; // 最大投资额度序列

static int[] res = new int[500005]; // 用于存储输出的投资额序列

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

int n = sc.nextInt(); // 产品数

int tot = sc.nextInt(); // 总投资额

int target = sc.nextInt(); // 可接受的总风险值

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

qian[i] = sc.nextInt(); // 输入产品投资回报率序列

}

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

danger[i] = sc.nextInt(); // 输入产品风险值序列

}

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

cnt[i] = sc.nextInt(); // 输入最大投资额度序列

}

int ans = 0; // 用于记录最大投资回报

int a = 0, b = 0; // 用于记录选择的两个产品的序号

int s1 = 0, s2 = 0; // 用于记录选择的两个产品的投资额

// 遍历每个产品 i

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

if (danger[i] > target)

continue; // 如果某个产品的风险超过可接受总风险，跳过该产品

// 若投资一个产品 i，计算其最大投资额带来的回报

int maxInvest = Math.min(tot,

cnt[i]); // 计算当前产品 i 的最大投资额

int x = maxInvest \* qian[i]; // 计算当前产品 i 的最大回报

// 如果仅投资一个产品 i，就能获得比当前最大回报更高的回报，就更新最大回报及相关变量

if (x > ans) {

ans = x;

a = 0; // 无第二个产品，设为 0

s1 = 0; // 无第一个产品的投资额，设为 0

b = i + 1; // 第一个产品的序号

s2 = maxInvest; // 第一个产品的投资额

}

// 遍历每对产品 i 和 j（j 从 i + 1 开始）

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

int riskSum = danger[i] + danger[j]; // 计算组合 i 和 j 的风险总和

if (riskSum > target)

continue; // 若组合 i 和 j 的风险总和超过可接受的总风险，跳过

// 遍历分配给产品 i 的投资额

for (int k = 1; k < tot; k++) {

int investI = qian[i] \* Math.min(k,

cnt[i]); // 计算分配给产品 i 的投资额与其回报

int left = tot - k; // 计算剩余的投资额

int investJ = qian[j] \* Math.min(left,

cnt[j]); // 计算分配给产品 j 的投资额与其回报

int totalInvestReturn = investI +

investJ; // 计算组合 i 和 j 的总投资回报

// 如果组合 i 和 j 能获得比当前最大回报更高的回报，更新最大投资回报及相关变量

if (totalInvestReturn > ans) {

ans = totalInvestReturn;

a = i + 1; // 第一个产品的序号

b = j + 1; // 第二个产品的序号

s1 = Math.min(k, cnt[i]); // 第一个产品的投资额

s2 = Math.min(left, cnt[j]); // 第二个产品的投资额

}

}

}

}

res[a] = s1; // 记录第一个产品的投资额

res[b] = s2; // 记录第二个产品的投资额

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

System.out.print(res[i] + " "); // 输出每个产品的投资额序列

}

}

}



# 输入和输出操作

n, tot, target = map(int, input().split()) # 输入产品数，总投资额及可接受的总风险值

qian = list(map(int, input().split())) # 输入产品投资回报率序列

danger = list(map(int, input().split())) # 输入产品风险值序列

cnt = list(map(int, input().split())) # 输入最大投资额度序列

ans = 0 # 用于记录最大投资回报

a, b = 0, 0 # 用于记录选择的两个产品的序号

s1, s2 = 0, 0 # 用于记录选择的两个产品的投资额

res = [0] \* (n + 1) # 用于存储输出的投资额序列

# 遍历每个产品 i

for i in range(n):

if danger[i] > target:

continue # 如果某个产品的风险超过可接受总风险，跳过该产品

# 若投资一个产品 i，计算其最大投资额带来的回报

max\_invest = min(tot, cnt[i]) # 计算当前产品 i 的最大投资额

x = max\_invest \* qian[i] # 计算当前产品 i 的最大回报

# 如果仅投资一个产品 i，就能获得比当前最大回报更高的回报，就更新最大回报及相关变量

if x > ans:

ans = x

a = 0 # 无第二个产品，设为 0

s1 = 0 # 无第一个产品的投资额，设为 0

b = i + 1 # 第一个产品的序号

s2 = max\_invest # 第一个产品的投资额

# 遍历每对产品 i 和 j（j 从 i + 1 开始）

for j in range(i + 1, n):

risk\_sum = danger[i] + danger[j] # 计算组合 i 和 j 的风险总和

if risk\_sum > target:

continue # 若组合 i 和 j 的风险总和超过可接受的总风险，跳过

# 遍历分配给产品 i 的投资额

for k in range(1, tot):

invest\_i = qian[i] \* min(k, cnt[i]) # 计算分配给产品 i 的投资额与其回报

left = tot - k # 计算剩余的投资额

invest\_j = qian[j] \* min(left, cnt[j]) # 计算分配给产品 j 的投资额与其回报

total\_invest\_return = invest\_i + invest\_j # 计算组合 i 和 j 的总投资回报

# 如果组合 i 和 j 能获得比当前最大回报更高的回报，更新最大投资回报及相关变量

if total\_invest\_return > ans:

ans = total\_invest\_return

a = i + 1 # 第一个产品的序号

b = j + 1 # 第二个产品的序号

s1 = min(k, cnt[i]) # 第一个产品的投资额

s2 = min(left, cnt[j]) # 第二个产品的投资额

res[a] = s1 # 记录第一个产品的投资额

res[b] = s2 # 记录第二个产品的投资额

print(" ".join(map(str, res[1 : n + 1]))) # 输出每个产品的投资额序列



#include <iostream>

#include <algorithm>

using namespace std;

int qian[500005]; // 产品投资回报率序列

int danger[500005]; // 产品风险值序列

int cnt[500005]; // 最大投资额度序列

int res[500005]; // 用于存储输出的投资额序列

int main() {

int n, tot, target;

cin >> n >> tot >>

target; // 输入产品数，总投资额以及可接受的总风险值

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

cin >> qian[i]; // 输入产品投资回报率序列

}

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

cin >> danger[i]; // 输入产品风险值序列

}

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

cin >> cnt[i]; // 输入最大投资额度序列

}

int ans = 0; // 用于记录最大投资回报

int a = 0, b = 0; // 用于记录选择的两个产品的序号

int s1 = 0, s2 = 0; // 用于记录选择的两个产品的投资额

// 遍历每个产品 i

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

if (danger[i] > target)

continue; // 如果某个产品的风险超过可接受总风险，跳过该产品

// 若投资一个产品 i，计算其最大投资额带来的回报

int maxInvest = min(tot, cnt[i]); // 计算当前产品 i 的最大投资额

int x = maxInvest \* qian[i]; // 计算当前产品 i 的最大回报

// 如果仅投资一个产品 i，就能获得比当前最大回报更高的回报，就更新最大回报及相关变量

if (x > ans) {

ans = x;

a = 0; // 无第二个产品，设为 0

s1 = 0; // 无第一个产品的投资额，设为 0

b = i + 1; // 第一个产品的序号

s2 = maxInvest; // 第一个产品的投资额

}

// 遍历每对产品 i 和 j（j 从 i + 1 开始）

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

int riskSum = danger[i] + danger[j]; // 计算组合 i 和 j 的风险总和

if (riskSum > target)

continue; // 若组合 i 和 j 的风险总和超过可接受的总风险，跳过

// 遍历分配给产品 i 的投资额

for (int k = 1; k < tot; k++) {

int investI = qian[i] \* min(k,

cnt[i]); // 计算分配给产品 i 的投资额与其回报

int left = tot - k; // 计算剩余的投资额

int investJ = qian[j] \* min(left,

cnt[j]); // 计算分配给产品 j 的投资额与其回报

int totalInvestReturn = investI +

investJ; // 计算组合 i 和 j 的总投资回报

// 如果组合 i 和 j 能获得比当前最大回报更高的回报，更新最大投资回报及相关变量

if (totalInvestReturn > ans) {

ans = totalInvestReturn;

a = i + 1; // 第一个产品的序号

b = j + 1; // 第二个产品的序号

s1 = min(k, cnt[i]); // 第一个产品的投资额

s2 = min(left, cnt[j]); // 第二个产品的投资额

}

}

}

}

res[a] = s1; // 记录第一个产品的投资额

res[b] = s2; // 记录第二个产品的投资额

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

cout << res[i] << " "; // 输出每个产品的投资额序列

}

return 0;

}



#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int qian[500005]; // 产品投资回报率序列

int danger[500005]; // 产品风险值序列

int cnt[500005]; // 最大投资额度序列

int res[500005]; // 用于存储输出的投资额序列

int main() {

int n, tot, target;

scanf("%d %d %d", &n, &tot,

&target); // 输入产品数，总投资额以及可接受的总风险值

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

scanf("%d", &qian[i]); // 输入产品投资回报率序列

}

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

scanf("%d", &danger[i]); // 输入产品风险值序列

}

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

scanf("%d", &cnt[i]); // 输入最大投资额度序列

}

int ans = 0; // 用于记录最大投资回报

int a = 0, b = 0; // 用于记录选择的两个产品的序号

int s1 = 0, s2 = 0; // 用于记录选择的两个产品的投资额

// 遍历每个产品 i

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

if (danger[i] > target)

continue; // 如果某个产品的风险超过可接受总风险，跳过该产品

// 若投资一个产品 i，计算其最大投资额带来的回报

int max\_invest = tot < cnt[i] ? tot :

cnt[i]; // 计算当前产品 i 的最大投资额

int x = max\_invest \* qian[i]; // 计算当前产品 i 的最大回报

// 如果仅投资一个产品 i，就能获得比当前最大回报更高的回报，就更新最大回报及相关变量

if (x > ans) {

ans = x;

a = 0; // 无第二个产品，设为 0

s1 = 0; // 无第一个产品的投资额，设为 0

b = i + 1; // 第一个产品的序号

s2 = max\_invest; // 第一个产品的投资额

}

// 遍历每对产品 i 和 j（j 从 i + 1 开始）

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

int risk\_sum = danger[i] + danger[j]; // 计算组合 i 和 j 的风险总和

if (risk\_sum > target)

continue; // 若组合 i 和 j 的风险总和超过可接受的总风险，跳过

// 遍历分配给产品 i 的投资额

for (int k = 1; k < tot; k++) {

int invest\_i = qian[i] \* (k < cnt[i] ? k :

cnt[i]); // 计算分配给产品 i 的投资额与其回报

int left = tot - k; // 计算剩余的投资额

int invest\_j = qian[j] \* (left < cnt[j] ? left :

cnt[j]); // 计算分配给产品 j 的投资额与其回报

int total\_invest\_return = invest\_i +

invest\_j; // 计算组合 i 和 j 的总投资回报

// 如果组合 i 和 j 能获得比当前最大回报更高的回报，更新最大投资回报及相关变量

if (total\_invest\_return > ans) {

ans = total\_invest\_return;

a = i + 1; // 第一个产品的序号

b = j + 1; // 第二个产品的序号

s1 = (k < cnt[i] ? k : cnt[i]); // 第一个产品的投资额

s2 = (left < cnt[j] ? left : cnt[j]); // 第二个产品的投资额

}

}

}

}

res[a] = s1; // 记录第一个产品的投资额

res[b] = s2; // 记录第二个产品的投资额

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

printf("%d ", res[i]); // 输出每个产品的投资额序列

}

return 0;

}



const readline = require("readline");

// 创建接口以便于读取输入

const rl = readline.createInterface({

input: process.stdin,

output: process.stdout,

});

let input = []; // 用于存储输入

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

input.push(line);

});

rl.on("close", () => {

let [n, tot, target] = input[0].split(" ").map(Number); // 输入产品数，总投资额及可接受的总风险值

let qian = input[1].split(" ").map(Number); // 输入产品投资回报率序列

let danger = input[2].split(" ").map(Number); // 输入产品风险值序列

let cnt = input[3].split(" ").map(Number); // 输入最大投资额度序列

let ans = 0; // 用于记录最大投资回报

let a = 0,

b = 0; // 用于记录选择的两个产品的序号

let s1 = 0,

s2 = 0; // 用于记录选择的两个产品的投资额

let res = new Array(n + 1).fill(0); // 用于存储输出的投资额序列

// 遍历每个产品 i

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

if (danger[i] > target) continue; // 如果某个产品的风险超过可接受总风险，跳过该产品

// 若投资一个产品 i，计算其最大投资额带来的回报

let maxInvest = Math.min(tot, cnt[i]); // 计算当前产品 i 的最大投资额

let x = maxInvest \* qian[i]; // 计算当前产品 i 的最大回报

// 如果仅投资一个产品 i，就能获得比当前最大回报更高的回报，就更新最大回报及相关变量

if (x > ans) {

ans = x;

a = 0; // 无第二个产品，设为 0

s1 = 0; // 无第一个产品的投资额，设为 0

b = i + 1; // 第一个产品的序号

s2 = maxInvest; // 第一个产品的投资额

}

// 遍历每对产品 i 和 j（j 从 i + 1 开始）

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

let riskSum = danger[i] + danger[j]; // 计算组合 i 和 j 的风险总和

if (riskSum > target) continue; // 若组合 i 和 j 的风险总和超过可接受的总风险，跳过

// 遍历分配给产品 i 的投资额

for (let k = 1; k < tot; k++) {

let investI = qian[i] \* Math.min(k, cnt[i]); // 计算分配给产品 i 的投资额与其回报

let left = tot - k; // 计算剩余的投资额

let investJ = qian[j] \* Math.min(left, cnt[j]); // 计算分配给产品 j 的投资额与其回报

let totalInvestReturn = investI + investJ; // 计算组合 i 和 j 的总投资回报

// 如果组合 i 和 j 能获得比当前最大回报更高的回报，更新最大投资回报及相关变量

if (totalInvestReturn > ans) {

ans = totalInvestReturn;

a = i + 1; // 第一个产品的序号

b = j + 1; // 第二个产品的序号

s1 = Math.min(k, cnt[i]); // 第一个产品的投资额

s2 = Math.min(left, cnt[j]); // 第二个产品的投资额

}

}

}

}

res[a] = s1; // 记录第一个产品的投资额

res[b] = s2; // 记录第二个产品的投资额

console.log(res.slice(1, n + 1).join(" ")); // 输出每个产品的投资额序列

});



package main

import (

"bufio"

"fmt"

"os"

"strconv"

"strings"

)

func main() {

reader := bufio.NewReader(os.Stdin) // 创建一个读取器以从标准输入读取数据

// 读取第一行输入

line, \_ := reader.ReadString('\n') // 从标准输入读取一行数据

parts := strings.Fields(line) // 将这一行数据按空格分割成字符串字段切片

n, \_ := strconv.Atoi(parts[0]) // 将第一个字段转为整数，表示产品数

tot, \_ := strconv.Atoi(parts[1]) // 将第二个字段转为整数，表示总投资额

target, \_ := strconv.Atoi(parts[2]) // 将第三个字段转为整数，表示可接受的总风险值

// 读取第二行输入

line, \_ = reader.ReadString('\n') // 从标准输入读取第二行数据

qian := make([]int, n) // 创建一个长度为n的切片用于存储产品投资回报率

for i, v := range strings.Fields(line) {

qian[i], \_ = strconv.Atoi(v) // 将每个字段转为整数并存储在qian切片中

}

// 读取第三行输入

line, \_ = reader.ReadString('\n') // 从标准输入读取第三行数据

danger := make([]int, n) // 创建一个长度为n的切片用于存储产品风险值

for i, v := range strings.Fields(line) {

danger[i], \_ = strconv.Atoi(v) // 将每个字段转为整数并存储在danger切片中

}

// 读取第四行输入

line, \_ = reader.ReadString('\n') // 从标准输入读取第四行数据

cnt := make([]int, n) // 创建一个长度为n的切片用于存储最大投资额度

for i, v := range strings.Fields(line) {

cnt[i], \_ = strconv.Atoi(v) // 将每个字段转为整数并存储在cnt切片中

}

// 初始化变量

ans := 0 // 用于记录最大投资回报

a, b := 0, 0 // 用于记录选择的两个产品的序号

s1, s2 := 0, 0 // 用于记录选择的两个产品的投资额

res := make([]int, n+1) // 创建一个长度为n+1的切片用于存储输出的投资额序列

// 遍历每个产品 i

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

if danger[i] > target { // 如果某产品的风险值超过可接受的总风险值

continue // 跳过该产品

}

// 若投资一个产品 i，计算其最大投资额带来的回报

maxInvest := min(tot, cnt[i]) // 计算单一产品的最大可投资金额（不超过总投资额或该产品的最大投资额度）

x := maxInvest \* qian[i] // 计算单一产品的最大投资回报

if x > ans { // 如果当前产品的最大投资回报超过记录的最大回报

ans = x // 更新最大回报

a, b = 0, i+1 // 更新产品序号

s1, s2 = 0, maxInvest // 更新投资金额

}

// 遍历每对产品 i 和 j

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

riskSum := danger[i] + danger[j] // 计算组合 i 和 j 的风险总和

if riskSum > target { // 如果组合 i 和 j 的风险总和超过可接受的总风险值

continue // 跳过该组合

}

// 遍历分配给产品 i 的投资额

for k := 1; k < tot; k++ {

investI := qian[i] \* min(k, cnt[i]) // 计算分配给产品 i 的投资额与其回报

left := tot - k // 计算剩余的投资额

investJ := qian[j] \* min(left, cnt[j]) // 计算分配给产品 j 的投资额与其回报

totalInvestReturn := investI + investJ // 计算组合 i 和 j 的总投资回报

if totalInvestReturn > ans { // 如果组合 i 和 j 的总投资回报超过当前最大回报

ans = totalInvestReturn // 更新最大投资回报

a, b = i+1, j+1 // 更新产品序号

s1, s2 = min(k, cnt[i]), min(left, cnt[j]) // 更新投资金额

}

}

}

}

res[a] = s1 // 记录第一个产品的投资额

res[b] = s2 // 记录第二个产品的投资额

// 输出每个产品的投资额序列

fmt.Println(strings.Trim(fmt.Sprint(res[1:]), "[]"))

}

// 辅助函数，用于返回两个整数中的较小值

func min(a, b int) int {

if a < b {

return a

}

return b

}