【华为OD机考 统一考试机试C卷】虚拟游戏理财（C++ Java JavaScript Python 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

在一款虚拟游戏中生活，你必须进行投资以增强在虚拟游戏中的资产以免被淘汰出局。

现有一家Bank，它提供有若干理财产品 m 个，风险及投资回报不同，你有 N（元）进行投资，能接收的总风险值为X。

你要在可接受范围内选择最优的投资方式获得最大回报。

备注：

- 在虚拟游戏中，每项投资风险值相加为总风险值；
- 在虚拟游戏中，最多只能投资2个理财产品；
- 在虚拟游戏中，最小单位为整数，不能拆分为小数；
- 投资额*回报率=投资回报

输入描述

第一行：

- 产品数（取值范围[1,20]）
- 总投资额（整数，取值范围[1, 10000]）
- 可接受的总风险（整数，取值范围[1,200]）

第二行：产品投资回报率序列，输入为整数，取值范围[1,60]

第三行：产品风险值序列，输入为整数，取值范围[1, 100]

第四行：最大投资额度序列，输入为整数，取值范围[1, 10000]

输出描述

每个产品的投资额序列

用例

输入	**5 100 10** **10 20 30 40 50** **3 4 5 6 10** **20 30 20 40 30**
输出	0 30 0 40 0
说明	投资第二项30个单位，第四项40个单位，总的投资风险为两项相加为4+6=10

解题思路

在满足总风险不超过容忍度和总投资额不超过预算的前提下，通过遍历选择**单个**或**两个**理财产品的组合来最大化投资回报。

伪代码如下：

1. 初始化最大回报值为0。
2. 遍历所有理财产品：
   a. 如果单个产品的风险值和投资额均不超过限制，考虑其回报值。
   b. 更新最大回报值。

3. 再次遍历所有理财产品组合（两两配对）：
   a. 如果两个产品的风险值之和和投资额之和均不超过限制，考虑这两个产品的回报值之和。
   b. 更新最大回报值。
4. 返回最大回报值。

C++

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

vector<int> readIntArray() {
    string line;
    getline(cin, line); // 读取一行输入
    istringstream iss(line);
    vector<int> numbers;
    int num;
    while (iss >> num) { // 将字符串分割并转换为整数数组
        numbers.push_back(num);
    }
    return numbers;
}

int main() {
    // 读取投资项目数量m、总投资额N和风险容忍度X
    vector<int> tmp = readIntArray();
    int m = tmp[0], N = tmp[1], X = tmp[2];
    // 读取每个项目的预期回报率
    vector<int> returns = readIntArray();
    // 读取每个项目的风险值
    vector<int> risks = readIntArray();
    // 读取每个项目的最大投资额
    vector<int> maxInvestments = readIntArray();

    // 初始化最大回报为0
    int maxReturn = 0;
    // 初始化最大回报对应的投资方案数组

```

```cpp
    vector<int> bestInvestments(m, 0);

    // 遍历所有项目
    for (int i = 0; i < m; i++) {
        // 如果单个项目的风险在容忍度范围内
        if (risks[i] <= X) {
            // 计算对项目i的投资额，不超过总投资额N和项目i的最大投资额
            int investmentForI = min(N, maxInvestments[i]);
            // 计算当前回报
            int currentReturn = investmentForI * returns[i];
            // 如果当前回报大于已知的最大回报
            if (currentReturn > maxReturn) {
                // 更新最大回报
                maxReturn = currentReturn;
                // 重置最佳投资方案数组，并为项目i分配投资额
                fill(bestInvestments.begin(), bestInvestments.end(), 0);
                bestInvestments[i] = investmentForI;
            }
        }

        // 遍历项目i之后的项目，寻找两个项目的组合投资方案
        for (int j = i + 1; j < m; j++) {
            // 如果两个项目的风险总和在容忍度范围内
            if (risks[i] + risks[j] <= X) {
                int investmentForI, investmentForJ;
                // 根据预期回报率决定投资额分配
                if (returns[i] > returns[j]) {
                    // 如果项目i的回报率高于项目j，优先投资项目i
                    investmentForI = min(N, maxInvestments[i]);
                    investmentForJ = min(N - investmentForI, maxInvestments[j]);
                } else {
                    // 如果项目j的回报率高于项目i，优先投资项目j
                    investmentForJ = min(N, maxInvestments[j]);
                    investmentForI = min(N - investmentForJ, maxInvestments[i]);
                }
                // 计算当前两个项目组合的回报
                int currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
                // 如果当前回报大于已知的最大回报
                if (currentReturn > maxReturn) {
                    // 更新最大回报
```

```
                    maxReturn = currentReturn;
                    // 重置最佳投资方案数组，并为两个项目分配投资额
                    fill(bestInvestments.begin(), bestInvestments.end(), 0);
                    bestInvestments[i] = investmentForI;
                    bestInvestments[j] = investmentForJ;
                }
            }
        }
    }

    // 输出最佳投资方案
    for (int investment : bestInvestments) {
        cout << investment << " ";
    }
    cout << endl;

    return 0;
}
```

Java

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

public class Main {
    public static void main(String[] args) {
        // 创建Scanner对象用于获取用户输入
        Scanner sc = new Scanner(System.in);
        // 读取一行输入并将其分割为字符串数组，然后转换为整数数组
        int[] tmp = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        // 获取投资项目数量m、总投资额N和风险容忍度X
        int m = tmp[0];
        int N = tmp[1];
        int X = tmp[2];
        // 读取每个项目的预期回报率
        int[] returns = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        // 读取每个项目的风险值
        int[] risks = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
        // 读取每个项目的最大投资额
```

```java
        int[] maxInvestments = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();

        // 初始化最大回报为0
        int maxReturn = 0;
        // 初始化最大回报对应的投资方案数组
        int[] bestInvestments = new int[m];

        // 遍历所有项目
        for (int i = 0; i < m; i++) {
            // 如果单个项目的风险在容忍度范围内
            if (risks[i] <= X) {
                // 计算对项目i的投资额，不超过总投资额N和项目i的最大投资额
                int investmentForI = Math.min(N, maxInvestments[i]);
                // 计算当前回报
                int currentReturn = investmentForI * returns[i];
                // 如果当前回报大于已知的最大回报
                if (currentReturn > maxReturn) {
                    // 更新最大回报
                    maxReturn = currentReturn;
                    // 重置最佳投资方案数组，并为项目i分配投资额
                    bestInvestments = new int[m];
                    bestInvestments[i] = investmentForI;
                }
            }

            // 遍历项目i之后的项目，寻找两个项目的组合投资方案
            for (int j = i + 1; j < m; j++) {
                // 如果两个项目的风险总和在容忍度范围内
                if (risks[i] + risks[j] <= X) {
                    int investmentForI, investmentForJ;
                    // 根据预期回报率决定投资额分配
                    if (returns[i] > returns[j]) {
                        // 如果项目i的回报率高于项目j，优先投资项目i
                        investmentForI = Math.min(N, maxInvestments[i]);
                        investmentForJ = Math.min(N - investmentForI, maxInvestments[j]);
                    } else {
                        // 如果项目j的回报率高于项目i，优先投资项目j
                        investmentForJ = Math.min(N, maxInvestments[j]);
                        investmentForI = Math.min(N - investmentForJ, maxInvestments[i]);
                    }
```

```
                    // 计算当前两个项目组合的回报
                    int currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
                    // 如果当前回报大于已知的最大回报
                    if (currentReturn > maxReturn) {
                        // 更新最大回报
                        maxReturn = currentReturn;
                        // 重置最佳投资方案数组，并为两个项目分配投资额
                        bestInvestments = new int[m];
                        bestInvestments[i] = investmentForI;
                        bestInvestments[j] = investmentForJ;
                    }
                }
            }
        }

        // 创建StringJoiner对象，用于构建输出格式
        StringJoiner sj = new StringJoiner(" ");
        // 遍历最佳投资方案数组，将投资额添加到StringJoiner中
        for (int investment : bestInvestments) {
            sj.add(String.valueOf(investment));
        }

        // 输出最佳投资方案
        System.out.println(sj.toString());
        // 关闭Scanner对象
        sc.close();
    }
}
```

javaScript

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

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

// 用于存储输入行的数组
```

```
const lines = [];

// 读取输入行的事件监听
rl.on('line', (line) => {
  lines.push(line);
}).on('close', () => {
  // 当输入完成时，开始处理数据

  // 读取一行输入并将其转换为整数数组的函数
  const readIntArray = (line) => line.split(' ').map(Number);

  // 读取投资项目数量m、总投资额N和风险容忍度X
  const [m, N, X] = readIntArray(lines[0]);
  // 读取每个项目的预期回报率
  const returns = readIntArray(lines[1]);
  // 读取每个项目的风险值
  const risks = readIntArray(lines[2]);
  // 读取每个项目的最大投资额
  const maxInvestments = readIntArray(lines[3]);

  // 初始化最大回报为0
  let maxReturn = 0;
  // 初始化最大回报对应的投资方案数组
  let bestInvestments = new Array(m).fill(0);

  // 遍历所有项目
  for (let i = 0; i < m; i++) {
    // 检查项目i的风险是否在容忍度X以内
    if (risks[i] <= X) {
      // 计算项目i的投资额，不超过总投资额N和项目i的最大投资额
      const investmentForI = Math.min(N, maxInvestments[i]);
      // 计算当前项目的回报
      const currentReturn = investmentForI * returns[i];
      // 如果当前回报大于已知的最大回报
      if (currentReturn > maxReturn) {
        // 更新最大回报
        maxReturn = currentReturn;
        // 重置最佳投资方案数组，并为项目i分配投资额
        bestInvestments = new Array(m).fill(0);
        bestInvestments[i] = investmentForI;

```

```
      }
    }

    // 遍历项目i之后的项目，寻找两个项目的组合投资方案
    for (let j = i + 1; j < m; j++) {
      // 如果两个项目的风险总和在容忍度范围内
      if (risks[i] + risks[j] <= X) {
        let investmentForI, investmentForJ;
        // 根据预期回报率决定投资额分配
        if (returns[i] > returns[j]) {
          // 如果项目i的回报率高于项目j，优先投资项目i
          investmentForI = Math.min(N, maxInvestments[i]);
          // 计算项目j的剩余可投资额
          investmentForJ = Math.min(N - investmentForI, maxInvestments[j]);
        } else {
          // 如果项目j的回报率高于项目i，优先投资项目j
          investmentForJ = Math.min(N, maxInvestments[j]);
          // 计算项目i的剩余可投资额
          investmentForI = Math.min(N - investmentForJ, maxInvestments[i]);
        }
        // 计算两个项目组合的当前回报
        const currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
        // 如果当前回报大于已知的最大回报
        if (currentReturn > maxReturn) {
          // 更新最大回报
          maxReturn = currentReturn;
          // 重置最佳投资方案数组，并为两个项目分配投资额
          bestInvestments = new Array(m).fill(0);
          bestInvestments[i] = investmentForI;
          bestInvestments[j] = investmentForJ;
        }
      }
    }
  }

  // 输出最佳投资方案
  console.log(bestInvestments.join(' '));
});
```

Python

```python
import sys

# 读取一行输入并将其转换为整数列表的函数
def read_int_array():
    # 使用input()替换sys.stdin.readline()以适应在线编译器
    return list(map(int, input().split()))

# 读取投资项目数量m、总投资额N和风险容忍度X
m, N, X = read_int_array()
# 读取每个项目的预期回报率
returns = read_int_array()
# 读取每个项目的风险值
risks = read_int_array()
# 读取每个项目的最大投资额
max_investments = read_int_array()

# 初始化最大回报为0
max_return = 0
# 初始化最大回报对应的投资方案列表
best_investments = [0] * m

# 遍历所有项目
for i in range(m):
    # 检查项目i的风险是否在容忍度X以内
    if risks[i] <= X:
        # 计算项目i的投资额，不超过总投资额N和项目i的最大投资额
        investment_for_i = min(N, max_investments[i])
        # 计算当前项目的回报
        current_return = investment_for_i * returns[i]
        # 如果当前回报大于已知的最大回报
        if current_return > max_return:
            # 更新最大回报
            max_return = current_return
            # 重置最佳投资方案列表，并为项目i分配投资额
            best_investments = [0] * m
            best_investments[i] = investment_for_i

    # 遍历项目i之后的项目，寻找两个项目的组合投资方案
    for j in range(i + 1, m):
        # 如果两个项目的风险总和在容忍度范围内
```

```
        if risks[i] + risks[j] <= X:
            # 根据预期回报率决定投资额分配
            if returns[i] > returns[j]:
                # 如果项目i的回报率高于项目j，优先投资项目i
                investment_for_i = min(N, max_investments[i])
                # 计算项目j的剩余可投资额
                investment_for_j = min(N - investment_for_i, max_investments[j])
            else:
                # 如果项目j的回报率高于项目i，优先投资项目j
                investment_for_j = min(N, max_investments[j])
                # 计算项目i的剩余可投资额
                investment_for_i = min(N - investment_for_j, max_investments[i])
            # 计算两个项目组合的当前回报
            current_return = investment_for_i * returns[i] + investment_for_j * returns[j]
            # 如果当前回报大于已知的最大回报
            if current_return > max_return:
                # 更新最大回报
                max_return = current_return
                # 重置最佳投资方案列表，并为两个项目分配投资额
                best_investments = [0] * m
                best_investments[i] = investment_for_i
                best_investments[j] = investment_for_j

# 输出最佳投资方案
print(' '.join(map(str, best_investments)))
```

C语言

```
#include <stdio.h>  // 导入标准输入输出库，用于输入输出
#include <string.h> // 导入字符串操作库，用于处理字符串

#define MAX_PRODUCTS 20 // 定义最大产品数量常量

// 用于读取整数数组的函数
void readIntArray(int *arr, int size) {
    for (int i = 0; i < size; i++) {
        scanf("%d", &arr[i]); // 循环读取输入的整数并存储到数组中
    }
}

```

```
int min(int a, int b) {
    return a < b ? a : b; // 返回两个整数中较小的一个
}

int main() {
    int m, N, X; // 分别代表产品数、总投资额和可接受的总风险
    scanf("%d %d %d", &m, &N, &X); // 读取这三个值

    int returns[MAX_PRODUCTS]; // 存储每个产品的投资回报率
    readIntArray(returns, m); // 读取投资回报率

    int risks[MAX_PRODUCTS]; // 存储每个产品的风险值
    readIntArray(risks, m); // 读取风险值

    int maxInvestments[MAX_PRODUCTS]; // 存储每个产品的最大投资额度
    readIntArray(maxInvestments, m); // 读取最大投资额度

    int maxReturn = 0; // 初始化最大回报为0
    int bestInvestments[MAX_PRODUCTS] = {0}; // 初始化最佳投资方案数组，初始值全为0

    // 遍历所有产品，尝试找到最佳的投资组合
    for (int i = 0; i < m; i++) {
        // 如果单个产品的风险在可接受范围内
        if (risks[i] <= X) {
            // 计算对产品i的投资额，不超过总投资额N和产品i的最大投资额
            int investmentForI = min(N, maxInvestments[i]);
            // 计算当前投资的回报
            int currentReturn = investmentForI * returns[i];
            // 如果当前回报大于已知的最大回报
            if (currentReturn > maxReturn) {
                maxReturn = currentReturn; // 更新最大回报
                memset(bestInvestments, 0, sizeof(bestInvestments)); // 重置最佳投资方案数组
                bestInvestments[i] = investmentForI; // 为产品i分配投资额
            }
        }

        // 遍历产品i之后的产品，寻找两个产品的组合投资方案
        for (int j = i + 1; j < m; j++) {
            // 如果两个产品的风险总和在可接受范围内
            if (risks[i] + risks[j] <= X) {

```

```
                int investmentForI, investmentForJ;
                // 根据回报率决定投资额分配
                if (returns[i] > returns[j]) {
                    // 如果产品i的回报率高于产品j，优先投资产品i
                    investmentForI = min(N, maxInvestments[i]);
                    investmentForJ = min(N - investmentForI, maxInvestments[j]);
                } else {
                    // 如果产品j的回报率高于产品i，优先投资产品j
                    investmentForJ = min(N, maxInvestments[j]);
                    investmentForI = min(N - investmentForJ, maxInvestments[i]);
                }
                // 计算当前两个产品组合的回报
                int currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
                // 如果当前回报大于已知的最大回报
                if (currentReturn > maxReturn) {
                    maxReturn = currentReturn; // 更新最大回报
                    memset(bestInvestments, 0, sizeof(bestInvestments)); // 重置最佳投资方案数组
                    bestInvestments[i] = investmentForI; // 为产品i分配投资额
                    bestInvestments[j] = investmentForJ; // 为产品j分配投资额
                }
            }
        }
    }

    // 输出最佳投资方案
    for (int i = 0; i < m; i++) {
        printf("%d ", bestInvestments[i]); // 打印每个产品的投资额
    }
    printf("\n"); // 打印换行符

    return 0; // 程序结束
}
```

完整用例

用例1

```
1 1000 10
10

```

```
5
1000
```

用例2

```
2 500 15
10 20
7 8
300 400
```

用例3

```
3 2000 50
15 25 35
10 20 30
500 1000 1500
```

用例4

```
4 300 20
5 10 15 20
2 4 6 8
100 100 100 100
```

用例5

```
5 800 30
8 16 24 32 40
3 6 9 12 15
160 320 480 640 800
```

用例6

```
6 1500 40
20 19 18 17 16 15
10 9 8 7 6 5
250 300 350 400 450 500
```

用例7

```
7 10000 100
1 2 3 4 5 6 7
10 20 30 40 50 60 70
1000 2000 3000 4000 5000 6000 7000
```

用例8

```
8 600 60
10 20 30 40 50 60 70 80
5 10 15 20 25 30 35 40
100 200 300 400 500 600 700 800
```

用例9

```
9 2000 150
60 50 40 30 20 10 5 3 1
90 80 70 60 50 40 30 20 10
200 400 600 800 1000 1200 1400 1600 1800
```

用例10

```
10 1000 80
20 18 16 14 12 10 8 6 4 2
15 14 13 12 11 10 9 8 7 6
100 200 300 400 500 600 700 800 900 1000
```

文章目录