

# 【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率之和。
  - b. 更新最大回报率。
4. 返回最大回报率。

## C++

```
1  #include <iostream>
2  #include <vector>
3  #include <algorithm>
4  #include <sstream>
5  using namespace std;
6
7  vector<int> readIntArray() {
8      string line;
9      getline(cin, line); // 读取一行输入
10     istringstream iss(line);
11     vector<int> numbers;
12     int num;
13     while (iss >> num) { // 将字符串分割并转换为整数数组
14         numbers.push_back(num);
15     }
16     return numbers;
17 }
18
19 int main() {
20     // 读取投资项目数量m、总投资额N和风险容忍度X
21     vector<int> tmp = readIntArray();
22     int m = tmp[0], N = tmp[1], X = tmp[2];
23     // 读取每个项目的预期回报率
24     vector<int> returns = readIntArray();
25     // 读取每个项目的风险值
26     vector<int> risks = readIntArray();
27     // 读取每个项目的最大投资额
28     vector<int> maxInvestments = readIntArray();
29
30     // 初始化最大回报为0
31     int maxReturn = 0;
32     // 初始化最大回报对应的投资方案数组
33 }
```

```

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
            }
        }
    }
}

```

```

74         maxReturn = currentReturn;
75         // 重置最佳投资方案数组, 并为两个项目分配投资额
76         fill(bestInvestments.begin(), bestInvestments.end(), 0);
77         bestInvestments[i] = investmentForI;
78         bestInvestments[j] = investmentForJ;
79     }
80 }
81 }
82 }
83
84 // 输出最佳投资方案
85 for (int investment : bestInvestments) {
86     cout << investment << " ";
87 }
88 cout << endl;
89
90 return 0;
}

```

## Java

```

1 import java.util.Arrays;
2 import java.util.Scanner;
3 import java.util.StringJoiner;
4
5 public class Main {
6     public static void main(String[] args) {
7         // 创建Scanner对象用于获取用户输入
8         Scanner sc = new Scanner(System.in);
9         // 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
10        int[] tmp = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
11        // 获取投资项目数量m、总投资额N和风险容忍度X
12        int m = tmp[0];
13        int N = tmp[1];
14        int X = tmp[2];
15        // 读取每个项目的预期回报率
16        int[] returns = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
17        // 读取每个项目的风险值
18        int[] risks = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
19        // 读取每个项目的最大投资额
20
21

```

```

20 int[] maxInvestments = Arrays.stream(sc.nextLine().split(" ")).mapToInt(Integer::parseInt).toArray();
21
22 // 初始化最大回报为0
23 int maxReturn = 0;
24 // 初始化最大回报对应的投资方案数组
25 int[] bestInvestments = new int[m];
26
27 // 遍历所有项目
28 for (int i = 0; i < m; i++) {
29     // 如果单个项目的风险在容忍度范围内
30     if (risks[i] <= X) {
31         // 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
32         int investmentForI = Math.min(N, maxInvestments[i]);
33         // 计算当前回报
34         int currentReturn = investmentForI * returns[i];
35         // 如果当前回报大于已知的最大回报
36         if (currentReturn > maxReturn) {
37             // 更新最大回报
38             maxReturn = currentReturn;
39             // 重置最佳投资方案数组, 并为项目i分配投资额
40             bestInvestments = new int[m];
41             bestInvestments[i] = investmentForI;
42         }
43     }
44
45     // 遍历项目i之后的项目, 寻找两个项目的组合投资方案
46     for (int j = i + 1; j < m; j++) {
47         // 如果两个项目的风险总和在容忍度范围内
48         if (risks[i] + risks[j] <= X) {
49             int investmentForI, investmentForJ;
50             // 根据预期回报率决定投资额分配
51             if (returns[i] > returns[j]) {
52                 // 如果项目i的回报率高于项目j, 优先投资项目i
53                 investmentForI = Math.min(N, maxInvestments[i]);
54                 investmentForJ = Math.min(N - investmentForI, maxInvestments[j]);
55             } else {
56                 // 如果项目j的回报率高于项目i, 优先投资项目j
57                 investmentForJ = Math.min(N, maxInvestments[j]);
58                 investmentForI = Math.min(N - investmentForJ, maxInvestments[i]);
59             }
60

```

```

61         // 计算当前两个项目组合的回报
62         int currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
63         // 如果当前回报大于已知的最大回报
64         if (currentReturn > maxReturn) {
65             // 更新最大回报
66             maxReturn = currentReturn;
67             // 重置最佳投资方案数组, 并为两个项目分配投资额
68             bestInvestments = new int[m];
69             bestInvestments[i] = investmentForI;
70             bestInvestments[j] = investmentForJ;
71         }
72     }
73 }
74 }
75
76 // 创建StringJoiner对象, 用于构建输出格式
77 StringJoiner sj = new StringJoiner(" ");
78 // 遍历最佳投资方案数组, 将投资额添加到StringJoiner中
79 for (int investment : bestInvestments) {
80     sj.add(String.valueOf(investment));
81 }
82
83 // 输出最佳投资方案
84 System.out.println(sj.toString());
85 // 关闭Scanner对象
86 sc.close();
87 }
}

```

## JavaScript

```

1  const readline = require('readline');
2
3  // 创建readLine接口实例
4  const rl = readline.createInterface({
5      input: process.stdin,
6      output: process.stdout
7  });
8
9  // 用于存储输入行的数组
10

```

```
10 const lines = [];
11
12 // 读取输入行的事件监听
13 rl.on('line', (line) => {
14     lines.push(line);
15 }).on('close', () => {
16     // 当输入完成时, 开始处理数据
17
18     // 读取一行输入并将其转换为整数数组的函数
19     const readIntArray = (line) => line.split(' ').map(Number);
20
21     // 读取投资项目数量m、总投资额N和风险容忍度X
22     const [m, N, X] = readIntArray(lines[0]);
23     // 读取每个项目的预期回报率
24     const returns = readIntArray(lines[1]);
25     // 读取每个项目的风险值
26     const risks = readIntArray(lines[2]);
27     // 读取每个项目的最大投资额
28     const maxInvestments = readIntArray(lines[3]);
29
30     // 初始化最大回报为0
31     let maxReturn = 0;
32     // 初始化最大回报对应的投资方案数组
33     let bestInvestments = new Array(m).fill(0);
34
35     // 遍历所有项目
36     for (let i = 0; i < m; i++) {
37         // 检查项目i的风险是否在容忍度X以内
38         if (risks[i] <= X) {
39             // 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
40             const investmentForI = Math.min(N, maxInvestments[i]);
41             // 计算当前项目的回报
42             const currentReturn = investmentForI * returns[i];
43             // 如果当前回报大于已知的最大回报
44             if (currentReturn > maxReturn) {
45                 // 更新最大回报
46                 maxReturn = currentReturn;
47                 // 重置最佳投资方案数组, 并为项目i分配投资额
48                 bestInvestments = new Array(m).fill(0);
49                 bestInvestments[i] = investmentForI;
50             }
51         }
52     }
53 }
```



```

51     }
52 }
53
54 // 遍历项目i之后的项目，寻找两个项目的组合投资方案
55 for (let j = i + 1; j < m; j++) {
56     // 如果两个项目的风险总和在容忍度范围内
57     if (risks[i] + risks[j] <= X) {
58         let investmentForI, investmentForJ;
59         // 根据预期回报率决定投资额分配
60         if (returns[i] > returns[j]) {
61             // 如果项目i的回报率高于项目j，优先投资项目i
62             investmentForI = Math.min(N, maxInvestments[i]);
63             // 计算项目j的剩余可投资额
64             investmentForJ = Math.min(N - investmentForI, maxInvestments[j]);
65         } else {
66             // 如果项目j的回报率高于项目i，优先投资项目j
67             investmentForJ = Math.min(N, maxInvestments[j]);
68             // 计算项目i的剩余可投资额
69             investmentForI = Math.min(N - investmentForJ, maxInvestments[i]);
70         }
71         // 计算两个项目组合的当前回报
72         const currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
73         // 如果当前回报大于已知的最大回报
74         if (currentReturn > maxReturn) {
75             // 更新最大回报
76             maxReturn = currentReturn;
77             // 重置最佳投资方案数组，并为两个项目分配投资额
78             bestInvestments = new Array(m).fill(0);
79             bestInvestments[i] = investmentForI;
80             bestInvestments[j] = investmentForJ;
81         }
82     }
83 }
84 }
85
86 // 输出最佳投资方案
87 console.log(bestInvestments.join(' '));
});

```

```

1 import sys
2
3 # 读取一行输入并将其转换为整数列表的函数
4 def read_int_array():
5     # 使用input()替换sys.stdin.readline()以适应在线编译器
6     return list(map(int, input().split()))
7
8 # 读取投资项目数量m、总投资额N和风险容忍度X
9 m, N, X = read_int_array()
10 # 读取每个项目的预期回报率
11 returns = read_int_array()
12 # 读取每个项目的风险值
13 risks = read_int_array()
14 # 读取每个项目的最大投资额
15 max_investments = read_int_array()
16
17 # 初始化最大回报为0
18 max_return = 0
19 # 初始化最大回报对应的投资方案列表
20 best_investments = [0] * m
21
22 # 遍历所有项目
23 for i in range(m):
24     # 检查项目i的风险是否在容忍度X以内
25     if risks[i] <= X:
26         # 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
27         investment_for_i = min(N, max_investments[i])
28         # 计算当前项目的回报
29         current_return = investment_for_i * returns[i]
30         # 如果当前回报大于已知的最大回报
31         if current_return > max_return:
32             # 更新最大回报
33             max_return = current_return
34             # 重置最佳投资方案列表, 并为项目i分配投资额
35             best_investments = [0] * m
36             best_investments[i] = investment_for_i
37
38 # 遍历项目i之后的项目, 寻找两个项目的组合投资方案
39 for j in range(i + 1, m):
40     # 如果两个项目的风险总和在容忍度范围内
41

```

```

41     if risks[i] + risks[j] <= X:
42         # 根据预期回报率决定投资额分配
43         if returns[i] > returns[j]:
44             # 如果项目i的回报率高于项目j, 优先投资项目i
45             investment_for_i = min(N, max_investments[i])
46             # 计算项目j的剩余可投资额
47             investment_for_j = min(N - investment_for_i, max_investments[j])
48         else:
49             # 如果项目j的回报率高于项目i, 优先投资项目j
50             investment_for_j = min(N, max_investments[j])
51             # 计算项目i的剩余可投资额
52             investment_for_i = min(N - investment_for_j, max_investments[i])
53         # 计算两个项目组合的当前回报
54         current_return = investment_for_i * returns[i] + investment_for_j * returns[j]
55         # 如果当前回报大于已知的最大回报
56         if current_return > max_return:
57             # 更新最大回报
58             max_return = current_return
59             # 重置最佳投资方案列表, 并为两个项目分配投资额
60             best_investments = [0] * m
61             best_investments[i] = investment_for_i
62             best_investments[j] = investment_for_j
63
64 # 输出最佳投资方案
65 print(' '.join(map(str, best_investments)))

```

## C语言

```

1  #include <stdio.h> // 导入标准输入输出库, 用于输入输出
2  #include <string.h> // 导入字符串操作库, 用于处理字符串
3
4  #define MAX_PRODUCTS 20 // 定义最大产品数量常量
5
6  // 用于读取整数数组的函数
7  void readIntArray(int *arr, int size) {
8      for (int i = 0; i < size; i++) {
9          scanf("%d", &arr[i]); // 循环读取输入的整数并存储到数组中
10     }
11 }
12
13

```

```

13 int min(int a, int b) {
14     return a < b ? a : b; // 返回两个整数中较小的一个
15 }
16
17 int main() {
18     int m, N, X; // 分别代表产品数、总投资额和可接受的总风险
19     scanf("%d %d %d", &m, &N, &X); // 读取这三个值
20
21     int returns[MAX_PRODUCTS]; // 存储每个产品的投资回报率
22     readIntArray(returns, m); // 读取投资回报率
23
24     int risks[MAX_PRODUCTS]; // 存储每个产品的风险值
25     readIntArray(risks, m); // 读取风险值
26
27     int maxInvestments[MAX_PRODUCTS]; // 存储每个产品的最大投资额度
28     readIntArray(maxInvestments, m); // 读取最大投资额度
29
30     int maxReturn = 0; // 初始化最大回报为0
31     int bestInvestments[MAX_PRODUCTS] = {0}; // 初始化最佳投资方案数组, 初始值全为0
32
33     // 遍历所有产品, 尝试找到最佳的投资组合
34     for (int i = 0; i < m; i++) {
35         // 如果单个产品的风险在可接受范围内
36         if (risks[i] <= X) {
37             // 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
38             int investmentForI = min(N, maxInvestments[i]);
39             // 计算当前投资的回报
40             int currentReturn = investmentForI * returns[i];
41             // 如果当前回报大于已知的最大回报
42             if (currentReturn > maxReturn) {
43                 maxReturn = currentReturn; // 更新最大回报
44                 memset(bestInvestments, 0, sizeof(bestInvestments)); // 重置最佳投资方案数组
45                 bestInvestments[i] = investmentForI; // 为产品i分配投资额
46             }
47         }
48
49         // 遍历产品i之后的产品, 寻找两个产品的组合投资方案
50         for (int j = i + 1; j < m; j++) {
51             // 如果两个产品的风险总和在可接受范围内
52             if (risks[i] + risks[j] <= X) {
53

```

```

54         int investmentForI, investmentForJ;
55         // 根据回报率决定投资额分配
56         if (returns[i] > returns[j]) {
57             // 如果产品i的回报率高于产品j, 优先投资产品i
58             investmentForI = min(N, maxInvestments[i]);
59             investmentForJ = min(N - investmentForI, maxInvestments[j]);
60         } else {
61             // 如果产品j的回报率高于产品i, 优先投资产品j
62             investmentForJ = min(N, maxInvestments[j]);
63             investmentForI = min(N - investmentForJ, maxInvestments[i]);
64         }
65         // 计算当前两个产品组合的回报
66         int currentReturn = investmentForI * returns[i] + investmentForJ * returns[j];
67         // 如果当前回报大于已知的最大回报
68         if (currentReturn > maxReturn) {
69             maxReturn = currentReturn; // 更新最大回报
70             memset(bestInvestments, 0, sizeof(bestInvestments)); // 重置最佳投资方案数组
71             bestInvestments[i] = investmentForI; // 为产品i分配投资额
72             bestInvestments[j] = investmentForJ; // 为产品j分配投资额
73         }
74     }
75 }
76 }
77
78 // 输出最佳投资方案
79 for (int i = 0; i < m; i++) {
80     printf("%d ", bestInvestments[i]); // 打印每个产品的投资额
81 }
82 printf("\n"); // 打印换行符
83
84 return 0; // 程序结束
}

```

## 完整用例

### 用例1

```

1 | 1 1000 10
2 | 10
3 |

```

4	5
	1000

用例2

1	2	500	15
2	10	20	
3	7	8	
4	300	400	

用例3

1	3	2000	50
2	15	25	35
3	10	20	30
4	500	1000	1500

用例4

1	4	300	20	
2	5	10	15	20
3	2	4	6	8
4	100	100	100	100

用例5

1	5	800	30		
2	8	16	24	32	40
3	3	6	9	12	15
4	160	320	480	640	800

用例6

1	6	1500	40			
2	20	19	18	17	16	15
3	10	9	8	7	6	5
4	250	300	350	400	450	500

用例7

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

用例8

1	8	600	60
2	10	20	30 40 50 60 70 80
3	5	10	15 20 25 30 35 40
4	100	200	300 400 500 600 700 800

用例9

1	9	2000	150
2	60	50	40 30 20 10 5 3 1
3	90	80	70 60 50 40 30 20 10
4	200	400	600 800 1000 1200 1400 1600 1800

用例10

1	10	1000	80
2	20	18	16 14 12 10 8 6 4 2
3	15	14	13 12 11 10 9 8 7 6
4	100	200	300 400 500 600 700 800 900 1000

文章目录

华为OD机考:统一考试 C卷 + D卷 + B卷 +A卷	
题目描述	
输入描述	
输出描述	
用例	
解题思路	
C++	

Java

JavaScript

Python

C语言

完整用例

用例1

用例2

用例3

用例4

用例5

用例6

用例7

用例8

用例9

用例10

# 机考真题 华为OD



CSDN @算法大师