# 【贪心】-虚拟游戏理财

## 题目描述与示例

### 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 ,风险及投资回报不同,你有 N (元)进行投资,能接受的总风险 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。

#### 说明:

- 1、在虚拟游戏中,每项投资风险值相加为总风险值;
- 2、在虚拟游戏中,最多只能投资 2 个理财产品;
- 3、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

## 补充说明

- 1、在虚拟游戏中,每项投资风险值相加为总风险值;
- 2、在虚拟游戏中,最多只能投资2个理财产品;
- 3、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

## 示例

#### 输入

```
1 5 100 10
2 10 20 30 40 50
3 3 4 5 6 10
4 20 30 20 40 30
```

### 输出

```
1 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

本题难度不高,但是背景陌生以及条件较多,需要耐心读题完成。

首先观察本题的数据范围,产品数 n 最大只有 20 。由于最多只会取 2 个产品,所以枚举所有的产品对需要  $0(n^2)$  的时间复杂度,在题目所给的数据范围下是该复杂度是可以接受的。

枚举所有产品对非常容易实现,直接使用两个for循环即可实现。即

当我们得到一个产品对的编号 (i, j) 时,需要考虑以下若干事情:

- 1. 判断两个产品的风险值的情况。如果两个产品风险值相加超过了 X ,即 risk\_lst[i] + risk\_lst[j] > X ,则必然不会选择这对二元组,直接跳过。
- 2. 两个产品的最大可投资额的相加结果 max\_amount\_lst[i] + max\_amount\_lst[j] ,是否 超过最大总投资额 total amount 。若
  - o 否。则产品 i 和 j 的选择份额 i \_ amount 和 j \_ amount 即为它们各自的最大可投资额 max \_ amount \_ lst[i] 和 max \_ amount \_ lst[j] 。这属于一种贪心策略。
  - 是。则进行后续的判断。
- 3. 产品 i 和 j 中选择单份份额回报率更高的产品,将其尽可能地选满。以如果单份产品 i 的投资回报率更高为例(j 回报率更高的情况也是类似的),即 return\_rate\_lst[i] >= return\_rate\_lst[j] 成立时,存在
  - 产品 i 的选择份额,为产品 i 的最大可投资额 max\_amount\_lst[i] 和总最大额 total\_amount 之间的较小值,即 i\_amount = min(max\_amount\_lst[i], total\_amount)
  - 产品j的份额,为总最大额减 total\_amount 去产品 i 的选择份额 i\_amount ,即j\_amount = total\_amount i\_amount
  - 。 这也属于一种贪心策略
- 4. 根据选择份额 i\_amount 和 j\_amount ,计算当前回报值 cur\_return ,更新全局的最大回报值和答案对。
  - 可以用以下格式储存具有最大回报值的产品编号以及选择份额
  - pairs = [[i, i\_amount], [j, j\_amount]]

上述过程可能会错误地掉一些可能的只选择单个产品的情况。当 risk\_lst[i] 和 risk\_lst[j] 的和大于 X ,但 risk\_lst[i] 或 risk\_lst[j] 本身并不大于 X 时,无法获得选择单个产品的情况。

因此还需要再加一个for循环判断只选择一种产品的情况。

```
1 for i in range(n):
2    if risk_lst[i] > X:
3         continue
4    i_amount = min(max_amount_lst[i], total_amount)
5    cur_return = i_amount * return_rate_lst[i]
6    if cur_return > max_return:
7         max_return = cur_return
8         pairs = [[0, 0], [i, i_amount]]
```

## 代码

## **Python**

```
1 # 题目:【贪心】2024E-虚拟游戏理财
2 # 分值: 200
3 # 作者: 闭着眼睛学数理化
4 # 算法: 贪心
5 # 代码看不懂的地方,请直接在群上提问
6
7
8 # 输入产品数量n,最大总投资额total amount,最高风险系数X
9 n, total_amount, X = map(int, input().split())
10
11 # 输入长度为n的回报率列表
12 return_rate_lst = list(map(int, input().split()))
13
14 # 输入长度为n的风险值列表
15 risk_lst = list(map(int, input().split()))
16
17 # 输入长度为n的最大投资额序列
18 max_amount_lst = list(map(int, input().split()))
19
20 # 初始化总的最大回报值为0
21 max_return = 0
22 # 储存当前最大会值对应的产品编号以及所选取的份额数
23 pairs = [[0, 0], [0, 0]]
24
25 # 双重循环,遍历所有产品对(二元组)
26 for i in range(n):
      for j in range(i+1, n):
27
         # 考虑风险值的影响,如果两个产品的风险值加起来超过了X,则这组二元组不能选择
28
         # 直接跳过
29
         if risk_lst[i] + risk_lst[j] > X:
30
31
32
         # 如果两个产品的最大可投资额加起来,也不超过最大总投资额total amount
         # 那么会贪心地将两个产品都选满
33
         # 即产品i选择max_amount_lst[i]份,产品j选择max_amount_lst[j]份
34
         if max_amount_lst[i] + max_amount_lst[j] <= total_amount:</pre>
35
             i_amount, j_amount = max_amount_lst[i], max_amount_lst[j]
36
         # 否则,我们必须在两个产品之间选择【单份产品的投资回报率】更高的产品
37
         # 贪心地尽可能选择它
38
39
         else:
```

```
# 如果单份产品i的投资回报率更高
40
             if return_rate_lst[i] >= return_rate_lst[j]:
41
                 # 产品i的份额,为产品i的最大额和总最大额之间的较小值
42
                 i_amount = min(max_amount_lst[i], total_amount)
43
                 # 产品i的份额,为总最大额减去产品i的份额
44
45
                 j_amount = total_amount - i_amount
             # 如果单份产品i的投资回报率更高
46
47
             else:
48
                 # 产品i的份额,为产品i的最大额和总最大额之间的较小值
                 j_amount = min(max_amount_lst[j], total_amount)
49
                 # 产品i的份额,为总最大额减去产品i的份额
50
                 i_amount = total_amount - j_amount
51
52
          # 计算得到对应的当前回报值cur_return
53
         cur_return = i_amount * return_rate_lst[i] + j_amount *
54
  return_rate_lst[j]
         # 如果当前计算得到的回报值比之前记录过的最大回报值更大
55
         # 则更新最大回报值以及pairs
56
         if cur_return > max_return:
57
             max_return = cur_return
58
59
             pairs = [[i, i_amount], [j, j_amount]]
60
61
62 # 考虑只选择1种产品i的情况
63 for i in range(n):
      if risk_lst[i] > X:
64
          continue
65
      # 产品i的份额,为产品i的最大额和总最大额之间的较小值
66
      i_amount = min(max_amount_lst[i], total_amount)
67
      cur_return = i_amount * return_rate_lst[i]
68
      # 如果当前计算得到的回报值比之前记录过的最大回报值更大
69
      # 则更新最大回报值以及pairs
70
      if cur_return > max_return:
71
         max_return = cur_return
72
73
         pairs = [[0, 0], [i, i_amount]]
74
75
76 # 构建答案序列,除了最终的i和i位置需要调整为最优的i amount和i amount,
77 # 其他位置所选取的份额都是0
78 ans = [0] * n
79 ans[pairs[0][0]] = pairs[0][1]
80 ans[pairs[1][0]] = pairs[1][1]
81
82 print(" ".join(str(num) for num in ans))
```

```
1 import java.util.*;
 2
3 public class Main {
       public static void main(String[] args) {
 5
           Scanner scanner = new Scanner(System.in);
 6
7
           int n = scanner.nextInt();
8
           int totalAmount = scanner.nextInt();
9
           int X = scanner.nextInt();
10
           int[] returnRateLst = new int[n];
11
           int[] riskLst = new int[n];
12
           int[] maxAmountLst = new int[n];
13
14
           for (int i = 0; i < n; i++) {
15
16
                returnRateLst[i] = scanner.nextInt();
17
           }
18
           for (int i = 0; i < n; i++) {
19
                riskLst[i] = scanner.nextInt();
20
           }
21
22
23
           for (int i = 0; i < n; i++) {
               maxAmountLst[i] = scanner.nextInt();
24
25
           }
26
           int maxReturn = 0;
27
           int[][] pairs = new int[2][2];
28
29
           for (int i = 0; i < n; i++) {
30
31
                for (int j = i + 1; j < n; j++) {
                    if (riskLst[i] + riskLst[j] > X) {
32
                        continue;
33
34
                    }
35
                    int iAmount, jAmount;
36
                    if (maxAmountLst[i] + maxAmountLst[j] <= totalAmount) {</pre>
37
                        iAmount = maxAmountLst[i];
38
                        jAmount = maxAmountLst[j];
39
                    } else {
40
41
                        if (returnRateLst[i] >= returnRateLst[j]) {
                            iAmount = Math.min(maxAmountLst[i], totalAmount);
42
43
                            jAmount = totalAmount - iAmount;
                        } else {
44
                            jAmount = Math.min(maxAmountLst[j], totalAmount);
45
                            iAmount = totalAmount - jAmount;
46
```

```
47
                    }
48
49
                    int curReturn = iAmount * returnRateLst[i] + jAmount *
50
   returnRateLst[i];
51
52
                    if (curReturn > maxReturn) {
                        maxReturn = curReturn;
53
54
                        pairs[0][0] = i;
                        pairs[0][1] = iAmount;
55
                        pairs[1][0] = j;
56
                        pairs[1][1] = jAmount;
57
                    }
58
                }
59
           }
60
61
62
63
           for (int i = 0; i < n; i++){
64
                if (riskLst[i] > X) {
                        continue;
65
66
                }
                int iAmount = Math.min(maxAmountLst[i], totalAmount);
67
                int curReturn = iAmount * returnRateLst[i];
68
69
                if (curReturn > maxReturn) {
70
                    maxReturn = curReturn;
                    pairs[0][0] = 0;
71
72
                    pairs[0][1] = 0;
                    pairs[1][0] = i;
73
74
                    pairs[1][1] = iAmount;
                }
75
           }
76
77
78
79
           int[] ans = new int[n];
80
           ans[pairs[0][0]] = pairs[0][1];
81
           ans[pairs[1][0]] = pairs[1][1];
82
           for (int i = 0; i < n; i++) {
83
                System.out.print(ans[i] + " ");
84
           }
85
       }
86
87 }
88
```

```
1 #include <iostream>
 2 #include <vector>
3 using namespace std;
 4
 5 int main() {
       int n, totalAmount, X;
 6
       cin >> n >> totalAmount >> X;
 7
 8
9
       vector<int> returnRateLst(n);
       vector<int> riskLst(n);
10
       vector<int> maxAmountLst(n);
11
12
       for (int i = 0; i < n; i++) {
13
14
           cin >> returnRateLst[i];
       }
15
16
       for (int i = 0; i < n; i++) {
17
18
           cin >> riskLst[i];
19
       }
20
21
       for (int i = 0; i < n; i++) {
           cin >> maxAmountLst[i];
22
23
       }
24
25
       int maxReturn = 0;
       vector<vector<int>> pairs(2, vector<int>(2, 0));
26
27
       for (int i = 0; i < n; i++) {
28
           for (int j = i + 1; j < n; j++) {
29
               if (riskLst[i] + riskLst[j] > X) {
30
31
                    continue;
               }
32
33
               int iAmount, jAmount;
34
35
               if (maxAmountLst[i] + maxAmountLst[j] <= totalAmount) {</pre>
36
                    iAmount = maxAmountLst[i];
                    jAmount = maxAmountLst[j];
37
               } else {
38
                   if (returnRateLst[i] >= returnRateLst[j]) {
39
                        iAmount = min(maxAmountLst[i], totalAmount);
40
                        jAmount = totalAmount - iAmount;
41
                    } else {
42
                        jAmount = min(maxAmountLst[j], totalAmount);
43
                        iAmount = totalAmount - jAmount;
44
45
                    }
46
               }
47
```

```
int curReturn = iAmount * returnRateLst[i] + jAmount *
48
   returnRateLst[j];
49
               if (curReturn > maxReturn) {
50
                    maxReturn = curReturn;
51
                    pairs[0][0] = i;
52
53
                    pairs[0][1] = iAmount;
54
                    pairs[1][0] = j;
55
                    pairs[1][1] = jAmount;
56
               }
           }
57
       }
58
59
60
       for (int i = 0; i < n; i++){
           if (riskLst[i] > X) {
61
62
                   continue;
           }
63
           int iAmount = min(maxAmountLst[i], totalAmount);
64
65
           int curReturn = iAmount * returnRateLst[i];
           if (curReturn > maxReturn) {
66
67
               maxReturn = curReturn;
               pairs[0][0] = 0;
68
               pairs[0][1] = 0;
69
70
               pairs[1][0] = i;
               pairs[1][1] = iAmount;
71
72
           }
73
       }
74
75
76
       vector<int> ans(n, 0);
77
       ans[pairs[0][0]] = pairs[0][1];
78
79
       ans[pairs[1][0]] = pairs[1][1];
80
81
       for (int i = 0; i < n; i++) {
           cout << ans[i] << " ";
82
       }
83
84
85
       return 0;
86 }
87
```

## 时空复杂度

时间复杂度: O(N^2)。双重循环所需时间复杂度

空间复杂度: 0(1)。除了输入的序列,仅需若干常数变量维护遍历过程。