# 1292：宠物小精灵之收服

## 题目描述

宠物小精灵是一部讲述小智和他的搭档皮卡丘一起冒险的故事。

一天，小智和皮卡丘来到了小精灵狩猎场，里面有很多珍贵的野生宠物小精灵。小智也想收服其中的一些小精灵。然而，野生的小精灵并不那么容易被收服。对于每一个野生小精灵而言，小智可能需要使用很多个精灵球才能收服它，而在收服过程中，野生小精灵也会对皮卡丘造成一定的伤害（从而减少皮卡丘的体力）。当皮卡丘的体力小于等于0时，小智就必须结束狩猎（因为他需要给皮卡丘疗伤），而使得皮卡丘体力小于等于0的野生小精灵也不会被小智收服。当小智的精灵球用完时，狩猎也宣告结束。

我们假设小智遇到野生小精灵时有两个选择：收服它，或者离开它。如果小智选择了收服，那么一定会扔出能够收服该小精灵的精灵球，而皮卡丘也一定会受到相应的伤害；如果选择离开它，那么小智不会损失精灵球，皮卡丘也不会损失体力。

小智的目标有两个：主要目标是收服尽可能多的野生小精灵；如果可以收服的小精灵数量一样，小智希望皮卡丘受到的伤害越小（剩余体力越大），因为他们还要继续冒险。

现在已知小智的精灵球数量和皮卡丘的初始体力，已知每一个小精灵需要的用于收服的精灵球数目和它在被收服过程中会对皮卡丘造成的伤害数目。请问，小智该如何选择收服哪些小精灵以达到他的目标呢？

## 输入

输入数据的第一行包含三个整数：N(0<N<1000)，M(0<M<500)，K(0<K<100)，分别代表小智的精灵球数量、皮卡丘初始的体力值、野生小精灵的数量。

之后的K行，每一行代表一个野生小精灵，包括两个整数：收服该小精灵需要的精灵球的数量，以及收服过程中对皮卡丘造成的伤害。

## 输出

输出为一行，包含两个整数：C，R，分别表示最多收服C个小精灵，以及收服C个小精灵时皮卡丘的剩余体力值最多为R。

## 输入样例

```
10 100 5
7 10
2 40
2 50
1 20
4 20
```

**输出样例**

3 30

**解析**

二维费用背包其实就是在01背包的基础上多了一维，因此，我们只需要仿照01背包的写法，在dp上再增加一个纬度，判断时再增加一重循环即可。

现在，我们将本题中的关键信息与01背包中的关键信息进行一一映射：

物品->野生小精灵

物品价值->每个小精灵的价值是1

物品重量->精灵球数量、皮卡丘体力值。因为每向背包中装入一个野生小精灵，都会消耗精灵球的数量和皮卡丘的体力值。

背包容量上限->精灵球初始数量、皮卡丘初始体力值。

很明显，我们需要使用尽可能少的消耗，来捕捉更多的野生小精灵。是不是与01背包的思路一模一样呢？

对于题目中的第二个问，我们可以这样思考：

通过整个动态规划过程，我们能够计算出捕获到的最多小精灵的数量。然后，我们只需要在dp表中去寻找当我们捕获到最多的小精灵所花费的最少生命值，然后再用最初的生命值减掉这个最小消耗，就是我们索要求的结果了。

**朴素算法**

```
#include<bits/stdc++.h>

using namespace std;

int ball, health, n;  //拥有的精灵球，初始体力值，野生小精灵的数量

int consume[1001];      //精灵球限制
int blood[501];         //血量限制
//不需要建立价值数组


//动态规划表 第一维：野生小精灵的数量 K(0<K<100)
//第二维：球的数量  N(0<N<1000)
//第三维：体力值  M(0<M<500)
int dp[101][1001][501] = {{0}};


int main() {
    //记录最大承重和物品数量
   cin >> ball >> health >> n;
```

```
    //记录每个物品的重量1，重量2
    for (int i = 1; i <= n; i++) {
        cin >> consume[i] >> blood[i];
    }

    //从第一个小精灵开始
    for (int i = 1; i <= n; i++) {
        //从第一个精灵球开始
        for (int j = 1; j <= ball; j++) {
            //从第一滴血开始
            for (int k = 1; k <= health; k++) {
                //当前所需要的精灵球数小于能够提供的精灵球数
                //或者血量不够
                if (j < consume[i] || k < blood[i]) {
                    //取前一个物品的最大价值
                    dp[i][j][k] = dp[i - 1][j][k];
                }
                    //二维费用背包只是多了一维约束
                else {
                    //读取上一行的数据
                    int lastValue = dp[i - 1][j][k];
                    //计算当前行数据
                    //每个小精灵的价值永远为1
                    int curValue = dp[i - 1][j - consume[i]][k - blo
                    //判断能否捕获当前野生小精灵
                    dp[i][j][k] = max(lastValue, curValue);
                    //01背包的公式
                    //dp[i][j] = max(dp[i-1][j],dp[i][j-w[i] + v[i]])
                }

            }
        }
    }
    //输出捕获的数量
    cout << dp[n][ball][health] << " ";
    int max = dp[n][ball][health];
    //查找捕获到指定数量的小精灵时的最小血量消耗
    for (int i = 0; i <= health; i++) {
        //第一次达到最大价值时，所消耗的血量
        if (dp[n][ball][i] == max) {
            //此刻即为最大的剩余血量值
            cout << health - i;
            break;
        }
    }
    return 0;
}
```

当然，以上的算法是无法AC的，因为时间复杂度过高，我们需要对其进行优化。仿照01背包的一维优化，我们也将上面的代码进行降纬，对于本题来说，我们只能将其从三维降低至二维。

**降维优化**

```
#include<bits/stdc++.h>

using namespace std;

int ball, health, n;  //拥有的精灵球，初始体力值，小精灵的数量
int consume[1001];       //精灵球限制
int blood[501];          //血量限制
// 第一维：球的数量  N(0<N<1000)
// 第二维：体力值  M(0<M<500)
int dp[1001][501] = {{0}};


int main() {
    //记录最大承重和物品数量
    cin >> ball >> health >> n;
    //记录每个物品的重量和价值
    for (int i = 1; i <= n; i++) {
        cin >> consume[i] >> blood[i];
    }
    //从第一个小精灵开始
    for (int i = 1; i <= n; i++) {
        //从第一个精灵球开始
       for (int j = ball; j >= 1; j--) {
            //从第一滴血开始
          for (int k = health; k >= 1; k--) {
                //如果当前的精灵球小于目标的精灵球
             if (j < consume[i] || k < blood[i]) {
                    dp[j][k] = dp[j][k];
                } else {
                    //每个小精灵的价值永远为1
                    dp[j][k] = max(dp[j][k], dp[j - consume[i]][k -
                }
            }
        }
    }
    //01背包的最大值在最后一个格子中
    cout << dp[ball][health] << " ";
    //从第一个小精灵开始
    for (int i = 0; i <= health; i++) {
        if (dp[ball][i] == dp[ball][health]) {
            cout << health - i;
            break;
        }
```

```
    }
    return 0;
}
```

逻辑航线培优教育，信息学奥赛培训专家。

扫码添加作者获取更多内容。

```
od[i]] + 1;
```

```
blood[i]]+1);
```