对于n件商品先以实际最大价值val来从大到小排序，然后如果val相等，那么再以积分b从小到大排序，如果实际最大价值val和积分b均相等，再以价钱a从小到大排序，这样就得到一个商品信息数组；如果积分和价钱为0就可以买下来的话，那么我们就先统计这个异常行为的实际价值，并从商品信息数组里去掉这部分；之后将排好序的商品属性，直接去掉k件免费，并统计拿掉的k件免费商品的价值；之后对剩余的商品根据积分来统计能购买到的最大价值并统计；再之后对剩余的商品根据价钱来统计能购买到的最大价值并统计；

品有多种属性价钱a、积分b及免费拿k名额，并非仅仅存在一种属性求最大价值，显然，难度比较大，但是我们依然可以使用动态规划来解这个问题！我们定义value[i][j][m]来表示第item件商品拥有i现金j积分及m免费拿名额时能购买到的最大价值。此时到item，i，j，m时，面临三种选择：

**第一种：**

第item件商品用钱买，此时简化为0/1背包问题，要么买要么不买，取最大值a。

a = max(value[i-1][j][k][m], value[i - 1][j-goods.v1][k][m] + goods.val);

**第二种：**

用积分兑换，要么兑换要么不兑换，取最大值b。

b = max(value[i-1][j][k][m], value[i - 1][j][k - goods.v2][m] + goods.val);

**第三种：**

免费拿，要么拿要么不拿，取最大值c。

c = max(value[i-1][j][k][m], value[i - 1][j][k][m - 1] + goods.val);

那么第item，j，k，m时的最大价值就是max（a，b，c），其转移方程如下：

value[i][j][m] = max(a, b, c)

**定义商品的属性**

a代表该商品的价钱，b代表该商品的积分，val代表该商品的实际价值

struct GoodsAttr

{

int a;

int b;

int val;

}stGoodsAttr;

**定义max函数**

该函数用于比较两个数并返回最大值

int max(int a, int b)

{

return (a >= b) ? a : b;

}

**定义多维背包算法**

int GetMaxvalue(vector<struct GoodsAttr> & goodset, const int & v1, const int & v2, const int & k)

{

int i, j, m;

int temp = 0;

vector<vector<vector<int>>> value(v1+1, vector<vector<int>>(v2+1, vector<int>(k+1, 0)));

for (auto & goods : goodset)

{

for (i = v1; i >= 0; i--)

{

for(j = v2; j >= 0; j--)

{

for (m = k; m >= 0; m--)

{

if (m >= 1)

{

temp = max(temp, value[i][j][m - 1] + goods.val);

}

if (j >= goods.b)

{

temp = max(temp, value[i][j - goods.b][m] + goods.val);

}

if (i >= goods.a)

{

temp = max(temp, value[i - goods.a][j][m] + goods.val);

}

value[i][j][m] = max(temp, value[i][j][m]);

temp = 0;

}

}

}

}

return value[v1][v2][k];

}

**最后带入数据算即可**