**描述：**

给定一个背包和n种物品，每种物品具有m（1<=m<=7）个属性。每种物品都有体积v，重量w，体积v，价值vi (v,w,v和vi都为整数),其余属性为可选属性，即某些属性为某些物品独有。限制体积和重量不超过背包容量。

**算法分析：**

多维背包问题，与0-1背包问题类似，都属于要寻找最优解的一类问题，用回溯法构造解空间树。对于该物品，只有选与不选两种决策，这个还要根据用户输入的要求以及整个算法计算出来的结果一起给出最优的方案。总共有n个物品，可以顺序的考虑每个物品，由此形成一颗解空间树；基本的思路就是：遍历这棵树，把各个情况都计算一遍，列举出所有的情况，最后进行判断。如果重量不超过背包容量，体积不超过背包所能承受的体积，并且符合用户所给出的属性值，该方案就是最后的答案，即最优方案。

**算法设计：**

利用回溯法试设计一个算法求出多维背包问题的解，即求出一个解向量X[t]。其中，X[t]=0或1，X[t]=0表示物品t不放入背包，X[t]=1表示物品t放入背包。

在函数Tract中，t>=n时，判断当前物品价值是否大于目前背包内物品的价值；目前背包内物品价值cm初始值为0；找出最大价值，第m个为最大价值，搜寻至叶子结点，给出最优方案。t<n时，当前扩展结点位于排列树的第（i-1）层，此时算法选择下一个要安排的物品，以深度优先方式递归的对相应的子树进行搜索，对不满足约束的结点，则剪去相应的子树。对于满足约束的节点，则继续加入可选队列。

**时间复杂度：**

因为物品有选与不选2个决策，且总共有n个物品，所以时间复杂度为O(2n)。

**伪代码：**

Begin

定义变量，数组

声明函数

遍历子树

If

当前重量cw加上目前背包内重量weight[t]>背包容量c

return false

停止循环 结束算法

else

return ture

继续算法

if t>=n

if cv>mv

初次将cv的值付给mv

遍历到子树

得到最佳路径bestx[m]

else 当t<n时

继续遍历下一个可行节点

if x[t]==0

不放入背包，则继续往下搜索

else if

放入背包，将当前价值与目前背包价值的和赋给cv

cw、cvo同上

继续往下一层搜索

x[t]=0

则为不放入，减去此时的价值、重量、体积

重新回溯

void main()

从屏幕显示需要输出内容的文字提示

给出最优方案

end