**多维背包问题算法设计与分析**

**算法设计思路:**我用的是回溯法来求解。分别输入n个物品和m个属性，可以顺序依次考虑每个物品，这样就形成了解空间树：基本思想就是先判断level是否超过物品总数，也就是到了叶子节点。如果是，如果当前价值大于最大价值，则把当前价值设为最大价值。否则遍历这棵树，对于每一个物品，对于该物品只有放与不放2个选择，用flag来标志，一个属性的可用容量小于物品容量，则不能装载该物品，则把flag设为0，反正设为1；递归调用，继续往下遍历，直到不能再往下了，则进入回溯阶段，如果后续物品的价值没有能超过当前最大价值的话，就进行剪枝操作；在满足约束的情况下，且价值最大的话，该方案就是最后的答案，并输出哪几个物品放进了背包中。

**时间复杂度分析：T(n)=O(n\*2^n)**  
**核心代码：**

void DFS(所需参数)

{

if(level>=n+1) //level超过物品总数，也就是到了叶子节点。

{

if(current\_value>maxValue) //如果当前价值大于最大价值

{

maxValue = current\_value;//把当前价值设为最大价值

for(i=1;i<=n;i++) //打印出选择的物品

bestSelect[i] = select[i];

}

}

else

{

bool flag=1;

for(i=1;i<=m;i++)

{

if(currentTrait[i]<wp[level+1].trait[i])

flag=0;

//又一个属性的可用容量小于物品容量，则不能装载该物品

}

if(flag)

{

//可以下一个节点装载，容量减，价值加，记录数组赋值为：1

for(i=1;i<=m;i++)

{

currentTrait[i]-=wp[level+1].trait[i];

}

current\_value = current\_value + value[level+1];

select[level+1] = 1;

//递归调用，继续往下遍历

DFS(level+1,currentTrait,current\_value);

//不能再往下遍历了，则进入回溯阶段

select[level+1] = 0;

for(i=1;i<=m;i++)

{

currentTrait[i]+=wp[level+1].trait[i];

}

current\_value = current\_value - value[level+1];

}

//如果不能装载level的下一个物品，跳过level的下一个（记录数组对应值为0不动）

//不选择level+1物品，遍历level+1的子节点

DFS(level+1,currentTrait,current\_value);

}

}