**多维背包问题算法与设计分析**

1. 【回溯法】——用回溯法来解决多维背包问题

回溯法定义：回溯法是一个既带有系统性又带有跳跃性的搜索算法。它在包含问题的所有解的解空间树中按照深度优先的策略，从根节点出发搜索解空间树。算法搜索至解空间树的任一节点时，总是先判断该节点是否肯定不包含问题的解。如果肯定不包含，则跳过对以该节点为根的子树的系统搜索，逐层向其原先节点回溯。否则，进入该子树，继续按深度优先的策略进行搜索。

运用回溯法解题通常包含以下三个步骤：

1、针对所给问题，定义问题的解空间；

2、确定易于搜索的解空间结构；

3、以深度优先的方式搜索解空间，并且在搜索过程中用剪枝函数避免无效搜索；

二、【回溯法】－－多维背包问题算法设计分析

【整体思路】

多维背包问题与01背包问题性质差不多，应该是属于找最优解问题，只不过01背包是关于价值和重量，只要其重量不超过背包容量，就可以放进背包中。而有关于多维背包，放入的物品必须满足其所有约束条件，与01背包问题一样，我们也可以用回溯法需要构造解的子集树。对于每一个物品i，对于该物品只有选与不选2个决策，总共有n个物品，如果满足其所有约束条件，就可以放进背包中，可以顺序依次考虑每个物品，这样也就形成了一棵解空间树：

基本思想就是遍历这棵树，以枚举所有情况，最后进行判断，如果重量不超过背包容量，且价值最大的话，该方案就是最后的答案。多维背包问题使用回溯法的大概思路应该是：把物品看成是一棵树，0、1就是树枝，从一开始一直往下搜索下一个节点，满足其所有约束条件就可以放入背包中，只要其其中一个属性不满足它的约束条件，就不能放入背包中，继续遍历下一层，然后，递归回溯到上一个，比较上一种取或者不取的相对情况，规划出另一种方案，知道把整棵树遍历完，然后把每一种方案的所有解进行比较，最大的解就是其最优的价值了。 （剪枝函数）在搜索状态空间树时，只要左子节点是可一个可行结点，搜索就进入其左子树。对于右子树时，先计算上界函数，以判断是否将其减去（剪枝）。

多维背包问题应该和01背包问题的性质差不多，在搜索解空间树时，只要左儿子节点是一个可行节点，搜索就进入左子树，在右子树中有可能包含最优解才进入右子树搜索，否则将右子树剪去。

　　上界函数bound():当前价值cw+剩余容量可容纳的最大价值<=当前最优价值bestp。　在运用上界函数剪枝之前，选择先将物品按照其单位重量价值从大到小排序，此后就按照顺序考虑各个物品。本题中，我没有运用到上界函数，在使用上街函数前必须先使用冒泡排序把它的价值属性排好序，而我直接用递归回溯法遍历求出最优解，然后，对于不符合约束条件的物品则用剪枝函数进行剪枝。

第一份数据的运行数值和运行结果如下：

6 10

//6表示物品数量，10表示其数量的限制条件（维度）

100 600 1200 2400 500 2000

//这一行表示物品价值

8 12 13 64 22 41

8 12 13 75 22 41

3 6 4 18 6 4

5 10 8 32 6 12

5 13 8 42 6 20

5 13 8 48 6 20

0 0 0 0 8 0

3 0 4 0 8 0

3 2 4 0 8 4

3 2 4 8 8 4

//这一部分则表示属性，约束条件的值

80 96 20 36 44 48 10 18 22 24

//限制条件的约束

3800.0 //运行结果，最优解

1. 【回溯法】——算法伪代码如下：

回溯法，调用递归回溯，自己调用自己的一个方法{

如果i大于n(n表示物品的数量)则表示它搜索到叶子节点{

如果cv(物品价值)大于bestcp{

直接把cv赋值给bestcp

}

}

否则，则表示这个活节点不是叶子节点的情况{

假设把第i物品装入到背包中x[i]=1

循环判断物品i是否满足所有的约束条件{

如果y[j]减去shuxing[j][i]小于0

则表示不满足条件的情况{

则x[i]=0，不把物品放入背包中

Break，提前结束循环

}

}

}

否则，如果y[j]减去shuxing[j][i]大于0

则表示满足所有约束条件，物品i可以放入背包中{

则更新cv(当前价值)cv=cv(原来价值) +v[i]（物品价值）

}

装入了物品之后{

要把第P个的剩余装载空间要减去物品i所对应的消耗

当变量P大于等于c维度时，跳出循环 }

然后用回溯法回backtrack(i+1,cv)

物品i装入背包时，搜索下一个节点 搜索完后回溯

回溯时当前价值cv 减去物品i的价值 v[i]， cv-=v[i]

回溯时，用for循环

把消耗掉的装载空间加回来 {

y[p]+=shuxing[p][i]

如果变量p大于或等于cw(维度)，则跳出循环

}

}

用递归回溯backtrack(i+1,cv)

当物品不装入背包时，再继续搜索下一个节点

}

}

}

回溯法伪代码如下：

/\*

\* @param i：查找到了第i个物品（i从1开始索引）

\* @param cv：当前包中的价值

\*/

private static void backtrack(int i,double cv){ //这里用的是递归回溯

if(i>=n){ //如果i大于物品数量（到达叶子节点）

if(cv>bestcp){ //如果当前物品的价值大于最优解，则把它赋值给最优解bestcp

bestcp=cv; //更新最大价值

}

}

else{ //不是叶子节点的情况

x[i] = 1; //将物品i装入背包

for(int j=0;j<c;j++) { //循环判断是否满足所有约束条件

if (y[j]-shuxing[j][i]<0）//不满足约束条件时候的情况 （备注：这里的数组y[i]是表示第i个约束条件剩余的装载空间）

{

x[i]=0; //不将物品装入背包

break; //有一个条件不满足了就可以提前结束循环了

}

}

if(x[i]==1) { //如果物品i装入了背包

cv+=v[i]; //装入了背包肯定要更新当前价值 就是 原来的价值cv 加上物品i的价值 v[i] 变成新的价值cv

for(int p=0;p<c;p++)

{

y[p]-=shuxing[p][i]; //装入了物品，第p个约束条件的剩余装载空间要减去物品i所对应的消耗

}

backtrack(i+1,cv); //物品i装入背包时，搜索下一个节点

//搜索完后回溯

cv-=v[i]; //回溯时当前价值cv 减去物品i的价值 v[i]

for(int p=0;p<c;p++)

{

y[p]+=yueshu[p][i]; //回溯时 把刚刚物品i消耗的装载空间加回来

}

}

backtrack(i+1,cv); //物品i不装入背包时，搜索下一个节点

}

}

}