**问题描述:**

有N个物品，它的价值分别为A1,A2…..An，有M个属性，每个属性均有约束条件，如何让背包里装入的物品具有最大的价值总和？

**回溯算法的基本思想是：**

从一条路往前走，能进则进，不能进则退回来，换一条路再试。回溯法从开始结点（根结点）出发，以深度优先的方式搜索整个解空间。这个开始结点就成为一个活结点，同时也成为当前的扩展结点。在当前的扩展结点处，搜索向纵深方向移至一个新结点。这个新结点就成为一个新的活结点，并成为当前扩展结点。如果在当前的扩展结点处不能再向纵深方向移动，则当前扩展结点就成为死结点。换句话说，这个结点不再是一个活结点。此时，应往回移动（回溯）至最近的一个活结点处，并使这个活结点成为当前的扩展结点。回溯法即以这种工作方式递归地在解空间中搜索，直至找到所要求的解或解空间中已没有活结点时为止。

**回溯法核心算法伪代码形式:**

/\*回溯法\*/

void Backtracking(int m) {

/\*当m > num 时，说明已经到达边界，进行边界处理。否则，进行计算最大价值 注意：这是递归结束的条件\*/

if (m > num) {

/\*边界处理：当如果当前物品价值current\_value大于最大价值best\_value，则更新最大价值，否则什么也不做。\*/

if (current\_value > best\_value)

best\_value = current\_value;

} else {

/\*----将物品m预装入背包---\*/

/\* 首先假设物品m装入背包，然后通过循环计算物品m装入背包后是否超过上界，

\*如果超过上界，则将物品m退出背包，恢复装入物品m之前的现场，并且结束循环。

\*/

load[m] = 1;

for (int j = 0; j < limit\_num; j++) {

if (curreen\_limit[j] + characteristic[j][m] > limit[j]) {

load[m] = 0; //如果产生越界，则不装入物品m，将物品退出，进行现场恢复。

break;

}

}

/\*----这里才是真正判断是否将物品装入背包，如果装入了执行if语句，在if语句里面递归搜索，否则在if外部递归搜索---\*/

if (load[m] == 1) {

/\*更新没有搜索过的各个节点的约束大小和价值，无论是否装入，这个节点都已经考虑过了，剩下的r\_limit和surplus\_value都不能再包含它\*/

for (int i = 1; i < limit\_num + 1; i++)

r\_limit[i] -= characteristic[i][m];

surplus\_value -= value[m];

/\*----将物品m实际装入背包，计算当前占用的约束条件空间以及当前价值---\*/

if (load[m] == 1) {

for (int i = 1; i < limit\_num + 1; i++)

curreen\_limit[i] += characteristic[i][m];

current\_value += value[m];

}

Backtracking(m + 1); /\*在物品m装入的条件下，计算下一个物品的情况\*/

/\*向上层回溯\*/

for (int i = 1; i < limit\_num + 1; i++)

r\_limit[i] += characteristic[i][m];

surplus\_value += value[m];

/\*当物品装入了背包时，还原当前占用的约束条件空间以及当前价值，上面加了就要减去\*/

if (load[m] == 1) {

for (int i = 1; i < limit\_num + 1; i++)

curreen\_limit[i] -= characteristic[i][m];

current\_value -= value[m];

}

}

Backtracking(m + 1); /\*在物品m不满足条件即 load[m] = 0 没有被装入时，计算下一个物品的情况\*/

}

}

**算法复杂度分析:**

1.时间复杂度分析

该算法的时间复杂度为：

T(n) = 2n\*m

2.空间复杂度分析

本算法需要开辟n个空间进行存放n个商品的各个属性，因此本算法的空间复杂度为：

S(n) = O(n)