**算法设计与分析——多维背包问题**

问题描述：

多维背包给定N种物品，每种物品有M个属性，每种属性有对应的约束条件，要求物品的所有属性在满足对应约束条件的情况的同时，获取到背包的最大价值。简单来说，就是在经典的0-1背包问题上多增加了一个维度进行分析。

回溯法解决多维背包问题：

利用回溯法解决多维背包问题是一个子集选取问题，适合于用子集树表示多维背包问题的解空间。在搜索解空间树时，只要其左儿子节点是一个可行节点，搜索就进入左子树，在右子树中可能含有最优解时才进入右子树搜索。否则将右子树剪去。回溯法解决多维背包问题的时间复杂度和空间复杂度均为O(n)。

多维背包问题源代码：

#include<stdio.h>

//回溯法解决多维背包问题

int m; //物品的数量

int n; //物品属性的数量

int i;

double value[1000]; //物品的价值

int shuxing[1000][1000]; //物品的属性值

int yueshu[1000]; //各属性的约束值

int now[1000]; //当前的选取状态

int best[1000]; //最优的选取状态

double nowValue = 0; //当前背包的最大价值

double maxValue = 0; //最大价值

double nowShuxing[1000]; //当前所选物品的属性值总和

bool Bound(int k){ //布尔型函数，判断界限(即当前背包约束加上新物品的约束是否会超出背包约束上限)

for (i = 0; i < n; i++) //利用for循环遍历当前物品的所有属性

{

if (nowShuxing[i] + shuxing[i][k] <= yueshu[i]) //如果当前所选物品的属性值总和小于属性的约束值跳出for循环判断下一步i==n这个条件

continue;

else break;

}

if (i == n) //都符合才返回真值

return true;

else

return false;

}

//回溯函数

void BackTrack(int k)

{

//已搜索到根节点

if (k > m) //如果搜索到叶子节点

{

if (nowValue >= maxValue) //如果当前背包的总价值大于上一次的最大价值

{

maxValue = nowValue; //最大价值变成当前背包得总价值

for (i = 0; i < m; i++) //保存最优选取方案

{

best[i] = now[i];

}

}

return;

}

//搜索左子树

if ( Bound(k)) //如果当前所选物品的属性值总和+下一个物品的属性值<=背包的该属性约束值

{

for (i = 0; i < n; i++)

{

nowShuxing[i] += shuxing[i][k]; //表示当前属性值总和加上下一个属性值

}

nowValue += value[k]; //更新当前背包的最大价值

BackTrack(k + 1); // 调用回溯法

for (i = 0; i < n; i++) //不选择则回溯恢复原来的约束情况

{

nowShuxing[i] -= shuxing[i][k]; //当前属性值总和减去上一个属性值

}

nowValue -= value[k]; //当前背包的最大价值恢复为原来的价值

}

//不装入这个物品，直接搜索右边的节点

//遍历右子树

BackTrack(k + 1);

}

int main()

{

int j,x;

printf("请输入物品的数量、物品属性的数量：");

scanf("%d%d", &m,&n);

printf("物品的价值:");

for (j = 0; j < m; j++)

{

scanf("%lf", &value[j]); //读取物品价值数组

}

for (j = 0; j < n; j++)

{

printf("第%d个物品属性:",j+1);

for (x = 0; x < m; x++)

scanf("%d", &shuxing[j][x]); //读取物品属性的二维数组

}

printf("物品属性的约束值:");

for (j = 0; j < n; j++)

scanf("%d", &yueshu[j]); //读取属性约束值数组

BackTrack(0);

printf("最大价值为：%0.1f\n", maxValue);

return 0;

}