算法设计与分析文档

给定n中物品和一个限制条件为m的背包，物品i的约束条件为其价值为Vi,

mkp问题实质上也是0-1背包问题。0-1背包问题是如何选择装入背包的物品（物品不可分割），使得装入背包的物品的价值为最大。

1.题目分析：

考虑到每种物品只有2种选择，即装入背包或不装入背包，并且物品数、约束条件和限制已给定，

要计算装入背包物品的最大价值可用回溯法搜索子集树的算法进行求解。

2.算法设计：

a.我们可以首先设置一个物品类，定义变量n和m，n代表物品个数，m代表限制条件，用一维数组v来存放价值，一维数组a来存放限制条件，二维数组w用来存放约束条件，一维数组cx存放当前的解，一维数组bestx存放最终解。

定义变量cv代表当前的价值，bestv代表目前最优装载的价值，r代表剩余物品价值

b.设置一个YS函数初始化bestv，cx，bestx在这里会调用一次Backtrace(0,cv)

c.用递归函数Backtrace(i,cv)来实现回溯法搜索子集树（形式参数i表示递归深度）：

①若i==n，则算法搜索到一个叶结点，判断当前价值是否最优：

若cv>bestv，更新装载方案（即bestx[y]=cx[y]( 0≤y< n)）更新当前最优解（即bestv=cv）；输出当前最优的价值bestv,返回。

更新剩余的价值r(r-=v[i])，更新当前解（cx[i]=1）。

② 若a[j]-w[j][i]<0(0≤j< m)，cx[i]=0，跳出循环。

③ 如果cx[i]==1，更新当前价值（cv+=v[i]），更新限制条件(即a[s]-=w[s][i]( 0≤s< m))，然后调用一次BackTrack(i+1,cv)进行回溯，然后更新当前价值(即cv-=v[i])，再然后恢复现场(即a[p]+=w[p][i](0≤p< m))

④如果cv+r>bestv那就进行剪枝操作，搜索右子树，更新当前解(cx[i]=0)，然后调用BackTrack(i+1,cv)进行回溯。恢复现场(即r+=v[i])

d.定义主函数，在这里设置输入的输入值及提示语句，调用YS（w, v, a）函数，输出最大值。

import java.util.Scanner;

public class MKP {

private static int n, m;// n物品个数,m限制条件个数

private static double[] v; // 定义了一个一维数组v,用来存放价值

private static double[][] w; // 定义了一个二维数组w,用来存放约束条件

private static double[] a; // 定义了一个一维数组a,用来存放限制条件

private static double cv;// 当前的价值

private static double bestv;// 目前最优的价值

private static double r;// 剩余物品的价值

private static int[] cx;// 存放当前解

private static int[] bestx;// 存放最终解

public static double YS(double[][] w, double[] v, double[] a)

{

bestv = 0; //初始化,即目前最优的价值

cx = new int[n];

bestx = new int[n]; // 初始化r，即剩余最大价格

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

{

r += v[i]; //r=r+v[i]

}

// 调用回溯法计算

BackTrack(0,cv);

return bestv; //返回当前最优解

}

public static void BackTrack(int i, double cv)

{

if (i == n)

{// 到达叶结点

if (cv > bestv) //如果当前价值大于当前最优的价值

{

for (int y = 0; y <n; y++)

{

bestx[y] = cx[y]; //当前最优的价值等于当前解

}

bestv = cv; //当前最优的价值等于当前价值

System.out.println(bestv); //输出当前最优的价值

}

return; //返回

}

r -= v[i]; //r=r-v[i]

cx[i]=1; //当前解等于1

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

{

if (a[j]-w[j][i]<0) //如果限制条件减去约束条件的值小于0

{// 搜索左子树

cx[i] = 0; //当前解等于0

break; //跳出循环

}}

if(cx[i]==1) //如果当前解等于1

{cv += v[i]; //当前价值等于当前价值加上价值

for(int s=0;s<m;s++)

{a[s]-=w[s][i]; //限制条件等于限制条件减约束条件

}

BackTrack(i + 1,cv); //回溯

cv -= v[i]; //当前价值等于当前价值减价值

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

{a[p]+=w[p][i];}// 恢复现场

// 恢复现场

}

if (cv + r > bestv) //如果当前价值与剩余价值之和大于当前最优的价值

{// 剪枝操作

cx[i] = 0;// 搜索右子树

BackTrack(i + 1,cv); //回溯

}

r += v[i];

} // 恢复现场

public static void main(String[] args)

{

Scanner input = new Scanner(System.in);

try {

n = input.nextInt(); //输入物品个数n的值

m = input.nextInt(); //输入限制条件m的值

v = new double[n]; //开辟了行为n的双精度型一维数组v

w = new double[m][n]; //开辟了行为m,列为n的双精度型二维数组w

a = new double[m]; //开辟了行为m的双精度型一维数组a

System.out.println("请输入n个物品的价值:");

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

{

v[i] = input.nextDouble(); //输入价值数组的值

}

System.out.println("请输入约束条件:");

for (int i = 0; i < m; i++)

{

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

w[i][j] = input.nextInt(); //输入约束条件数组的值

}

System.out.println("请输入m个属性限制条件:");

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

{

a[j] = input.nextInt(); //输入限制条件数组的值

}

YS(w, v, a); //调用YS(w,v,a)方法

System.out.println("最大价值为：" + bestv); //输出最大值

}finally {

input.close();

}

}

}

5.调试代码，测试数据是mknapinfo.html里的内容。在调试代码的过程中我遇到了一些问题。我在测试第二个数据的时候，因为它是带有小数的，一开始我测试出来的结果是8688而实际结果是8706.1，在思考、查阅资料和请教别人之后我终于改正了错误，原来是我将一些数据类型设置成了整型，在我将它们改成双精度型之后终于得到了结果。

其实这个问题和0-1背包问题真的差不多，只不过在我理解的时候遇到了很多问题，后来我在网上查了很多的资料也请教了很多的人，我才弄懂了很多的东西。刚开始的时候我连题目都不懂，后来我才理解了。才明白了它的要求，那些数据是什么意思。我懂得了第一行数据是约束条件的个数，限制条件的个数等。我知道了我应该构造一维数组去存放价值，限制条件，构造二维数组存放约束条件等。代码的旁边我也有比较明确的注释。通过做这次作业我明白了什么都得去做才可以，不亲手去做，空抱怨是不行的，得自己花时间去弄懂，去学习。