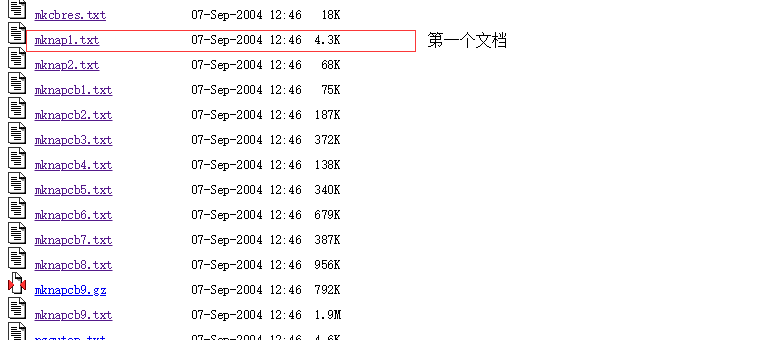
### 多维背包问题算法设与分析——解说版

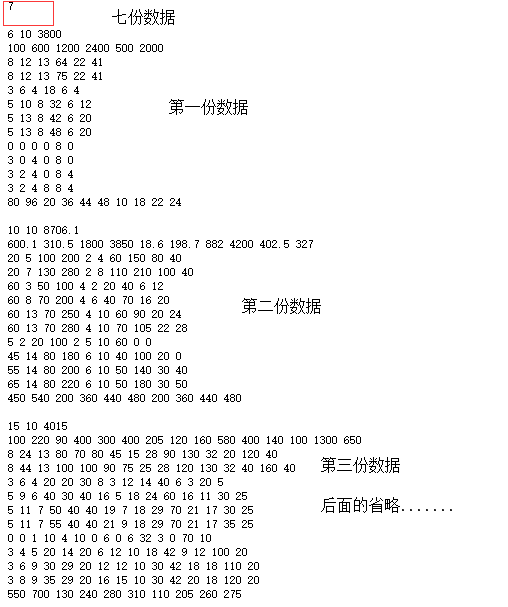
**一、题目解读**

题目给出了11个数据文档，第一个数据文件是mknap1，如图一。

该数据文件包含7个测试问题，如图二。



【图一】



【图二】

每份数据的含义：有6个背包或者说6个物品，每个背包有10种属性，（属性值可以为：物品的重量、价值、体积、数量等其他属性）第二行且数值较大的为每种属性的价值，最后一行对应10种属性的约束值，最大价值是3800。开始时数据形式陈列如下：

物品最大价值

背包有多少种属性

背包的个数

6 10 3800

100 600 1200 2400 500 2000

每一种物品的价值

8 12 13 64 22 41

8 12 13 75 22 4

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

80 96 20 36 44 48 10 18 22 24

对应10种物品价值的约束值

我们将其表示为：

③物品最大价值

①背包的个数

②背包有多少种属性

6

3800

10

④每一种物品的价值

100 600 1200 2400 500 2000

80

96

20

36

44

48

10

18

22

24

8 12 13 64 22 41

8 12 13 75 22 4

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

⑥对应10种物品价值的约束值

⑤每行物品约束条件的属性值

根据以上的图形解说，可以知道，每份数据都可以用这样的结构图形呈现出来，来表达各种属性关系。因为数据较多，在这里我们只举一个例子来说明。

1. **设置变量**

查看每个数据文档的数据我们发现，每份数据的背包个数、数据属性种类数、最大价值、每种属性价值和每种属性的约束值都是可变，

这可延伸为多维背包问题。我们将其表示为：

①背包的个数为n即物品数为n

②背包有多少种属性,设为约束数m

③每一种物品的价值：设有n个物品的价值

④每行物品属性的数值：约束条件的属性值a[i][j]

⑤对应物品价值的约束值：各个物品的价值的约束条件v[i]

⑥物品最大价值是我们所要求的值：设为最大价值为bestp元

**三、算法分析：**

【整体思路】01背包属于找最优解问题，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，可以顺序依次考虑每个物品，这样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。

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）。搜索的具体方法：  
**（解空间树）** 选不选

**物品1**

**物品2**

**物品2**

选 不选 选 不选

**物品3**

**物品3**

**物品3**

**物品3**

对于每一一个物品i,对于该物品只有选与不选2个决策，总共有n个物品，可以顺序依次考虑每个物品，这样就形成了一棵解空间树。

**四、算法思想设计**

我采用的是回溯法思想，利用回溯法设计一个算法求出0-1背包问题的解，即多维背包问题的解，也就是求出一个解向量ｘi （即对n个物品放或不放的一种的方案）其中， (ｘi = 0 或１，ｘi = 0表示物体ｉ不放入背包，ｘi ＝1表示把物体ｉ放入背包）。

在递归函数Backtrack中，当i>n时，算法搜索至叶子结点，得到一个新的物品装包方案。此时算法适时更新当前的最优价值，当i<n时，当前扩展结点位于排列树的第（i-1）层，此时算法选择下一个要安排的物品，以深度优先方式递归的对相应的子树进行搜索，对不满足上界约束的结点，则剪去相应的子树。

**五、时间复杂度以及优化**

　　因为物品只有选与不选2个决策，而总共有n个物品，所以时间复杂度为IMG_256。

因为递归栈最多达到n层，而且存储所有物品的信息也只需要常数个一维数组，所以最终的空间复杂度为O(n)。

**六、代码分析**

#include <stdio.h>

int n,m;//物品数量n，物品属性个数m

int i,j,s,d;

double c[100];//各个属性的约束

double v[100];//各个物品的价值　value

double w[100][100];//各个物品的重量　weight

double cw[100];//当前背包重量　current weight

double cp = 0.0;//当前背包中物品总价值　current value

double bestp = 0.0;//当前最大价值best price

//回溯函数

void backtrack(int x)

{ //i用来指示到达的层数（第几步，从0开始），同时也指示当前选择完了几个物品

double bound(int x);

if(x>n) //递归结束的判定条件

{

bestp = cp;

return;

}

//如若左子节点可行，则直接搜索左子树;

//对于右子树，先计算上界函数，以判断是否将其减去

for(i=1;i<=n;i++){

for(j=1;j<=m;j++){

if(cw[s]+w[i][j]<=c[d])//将物品i放入背包,搜索左子树

{

cw[s]+=w[i][j];//同步更新当前背包的重量

cp+=v[i];//同步更新当前背包的总价值

d++;

}

}

backtrack(x+1);//深度搜索进入下一层

cw[s]-=w[i][j];//回溯复原

cp-=v[i];//回溯复原

}

if(bound(x+1)>bestp)//如若符合条件则搜索右子树

backtrack(x+1);

}

//计算上界函数，功能为剪枝

double bound(int x)

{ //判断当前背包的总价值cp＋剩余容量可容纳的最大价值<=当前最优价值

for(d=1;d<=m;d++){

s+=1;

double leftw= c[d]-cw[s];//剩余背包容量

double b = cp;//记录当前背包的总价值cp,最后求上界

//以物品单位重量价值递减次序装入物品

while(x<=n && w[i][j]<=leftw)

{

leftw-=w[i][j];

b+=v[i];

x++;

}//装满背包

if(x<=n)

b+=v[i]/w[i][j]\*leftw;

return b;//返回计算出的上界

}

}

int main(){

int x;

int a[100][100];

printf("请输入物品数和约束数的值：\n");

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

printf("请输入每个物品的价值:\n");

for(i=1;i<=n;i++){

scanf("%d",&v[i]);

}

printf("请输入约束条件的属性值：\n");

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

for(j=1;j<=n;j++)

scanf("%d",&a[i][j]);

}

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

for(j=1;j<=n;j++);

}

printf("请输入每个约束条件的约束值：\n");

for(d=1;d<=n;d++){

scanf("%d",&c[d]);

}

backtrack(1);

printf("最大价值为：%lf\n",bestp);

return;

}