# 算法程序与设计分析

#### 多维背包问题用回溯法

1.多维背包问题是0-1背包的一个衍深，0-1背包的约束条件只有一个就是重量，但多维背包是多个约束条件，求最优解问题，本题目是给多个约束条件和多个物品属性值和物品的价值，求放入背包的最大价值。最初的想法是通过求出每一个约束条件下符合取值的集合，然后在这些对应每个约束条件下集合里求出他们的并集，然后发现这样的集合用一个表来装的话就很难实现动态读取数据并计算，在回想老师叫我们做的目的是用所学到的知识来解决问题，让我想到了动态规划，贪心算法，回溯法还有分支限界法。

2.以上代码是用c语言写的多维背包问题的代码，回溯法的思想是：它在问题的状态空间树中，从开始结点（根结点）出发，以深度优先搜索整个状态空间。这个开始结点成为活结点，同时也成为当前的扩展结点。在当前扩展结点处，搜索向纵深方向移至一个新结点。这个新结点成为新的活结点，并成为当前扩展结点。如果在当前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。此时，应往回移动（回溯）至最近的活结点处，并使这个活结点成为当前扩展结点。回溯法以这种工作方式递归地在状态空间中搜索，直到找到所要求的解或解空间中已无活结点时为止。

3.以文件式输入数据比较方便，将数据存入txt文本中保存到本地C盘对应位置，数据格式存放是物品个数+属性个数+最优值+物品价格+物品属性值+约束条件如下数据：

6 10 3800

100 600 1200 2400 500 2000

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

80 96 20 36 44 48 10 18 22 24

1. 通过Backtrack()函数递归得出最优值。
2. Backtrack()函数中通过f()函数递归判断物品当背包装入的物品的属性值总和加上当前i结点的物品属性值小于背包最大的属性值；通过bound()函数对右子树进行剪枝。
3. bound()函数中通过g()函数递归判断当背包中物品的属性值都小于背包剩余属性值

C语言：

#include<stdio.h>

int n;//物品的个数

int x[10000],bestx[10000];//x[i]暂存物品的选中情况,物品的选中情况

int a;//物品属性个数；

double p[10000],c[10000],w[10000][10000],bestp;//物品的价值,背包的属性限制,物品的属性对应值,最大价值

int g(int k, int j,double cleft[]){

if(k>a)

return 1;

else

if(w[k][j]<=cleft[k]){ // 当背包中物品的属性值都小于背包剩余属性值

return g(k+1,j,cleft);

}

else return 0;

}

//剪枝

double bound(int i,double cp,double cw[]){ //传入节点 i

int b;

//保存剩余的背包容量

double cleft[100000];

for(b=1;b<=a;b++)

cleft[b]=c[b]-cw[b]; //总属性容量 减去 当前已经装了的属性量

double tempv=cp,s=0; //保存当前的价值

int j;

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

if(g(1,j,cleft)){

tempv+=p[j];

for(b=1;b<=a;b++)

cleft[b]-=w[b][j];

} else{

for(b=1;b<=a;b++)

tempv+=p[j]/w[b][j]\*cleft[b] ;

break;

}

}

return tempv;

}

int f(int k,int j,int i,double cw[]){

if(k>a)

return 1;

else

if(cw[k]+j\*w[k][i]<=c[k]){//当背包装入的物品的属性值总和加上当前i结点的物品属性值小于背包最大的属性值时

return f(k+1,j,i,cw);

}

else return 0;

}

void Backtrack(int i,double cp,double cw[])

{ //cw当前包内物品重量，cp当前包内物品价值

int j,b,h;

if(i>n)//回溯结束

{

if(cp>bestp)

{

bestp=cp;

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

bestx[i]=x[i];

}

}

else

for(j=0;j<=1;j++)

{

x[i]=j;

if(f(1,j,i,cw)){

if(x[i]==1){//进入左子树l

for(b=1;b<=a;b++)

cw[b]+=w[b][i]\*x[i];

cp+=p[i]\*x[i];

Backtrack(i+1,cp,cw);

for(h=1;h<=a;h++)

cw[h]-=w[h][i]\*x[i];

cp-=p[i]\*x[i];}

else

if (bound(i+1,cp,cw)>bestp){

//后续节点的价值上界大于当前最优价值，则可以进入右结点

//否则最优的都小于或等于当前的 就没必要再进入右节点

Backtrack(i+1,cp,cw); }

//进入右节点 因为不加入到背包 故 当前的价值 属性值 都不发生改变

}

}

}

void main()

{

int i,j;

bestp=0;

double v[10000];//开一个足够大的数组。

int l=0,s;

FILE \*fp;//文件指针

fp = fopen("C://222.txt", "r");//以文本方式打开文件。

if(fp == NULL) //打开文件出错。

{

perror("fail to read");}

while(fscanf(fp, "%lf", &v[l]) != EOF) //读取数据到数组，直到文件结尾(返回EOF)

l++;

fclose(fp);//关闭文件

n=(int)v[0];//物品的个数

a=(int)v[1];//物品属性个数

s=0;

for(i=3;i<=n+2;i++) {

p[++s]=v[i];}

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

printf("%.1lf ",p[i]);

//s=6;

s=s+2;

printf("\n");

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

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

w[j][i]=v[++s];

//s=68

printf("%.lf ",w[j][i]);}

printf("\n");}

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

c[i]=v[++s];

printf("%.lf ",c[i]);}

printf("\n");

double cw[10000];

for(i=1;i<=a;i++)

cw[i]=0;

Backtrack(1,0,cw);

printf("最大价值为:\n");

printf("%.1lf\n",bestp);

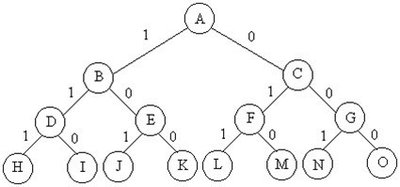
printf("被选中的物品依次是(0表示未选中，1表示选中)\n");

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

printf("%d ",bestx[i]);

printf("\n");

}



**剪枝**：就是对右子树进行剪枝，这里用到的是g()函数，情况1:假如当前结点的价值加上需要进入右子树的剩下节点物品的所有结点下的价格之和（在剩下结点都能装入背包的情况下）他们的价值总和都小于当前最优值就无需进入右子树，情况2：当g()函数中已经取了多个结点，但是当前的结点（指g()函数里的j变化）不能满足当前剩余量，就需要执行tempv+=p[j]/w[b][j]\*cleft[b]语句，意思是取他的单位价值来填满价值缺口。当当前j放不进当前背包剩余约束时，因为不知道当前j物品在没有放i~j时，当前j能否放得进去，所以需要去单位剩余价值来补这个价值缺口，直到j等于n时跳出循环返回子树的价值上界。

**算法的复杂度分析**：**Backtrack()**函数实现回溯，物品的个数**n**就是**Backtrack()**函数的递归次数，当进入**Backtrack()**函数时递归的次数没有达到最后一次，（也就是没有处理完最后一个物品的时候）接下来就是让他进入左子树和右子树当进入左子树时**X[i]=1**,当进入右子树时**x[i]=0**;假如给定**n**个物品所以此时用回溯法的算法的时间复杂度是**O(2^n)**;但因为进入了多个约束条件所以要判断当所以约束条件都满足的条件下进行**Backtrack()**函数的递归调用，这些约束条件也是用递归来实现，当递归到最后一个约束条件成立后返回参数值1，否则返回0。上述**f()**函数实现这个功能，只有所以当前属性值都满足约束条件时返回值1，否者返回0。

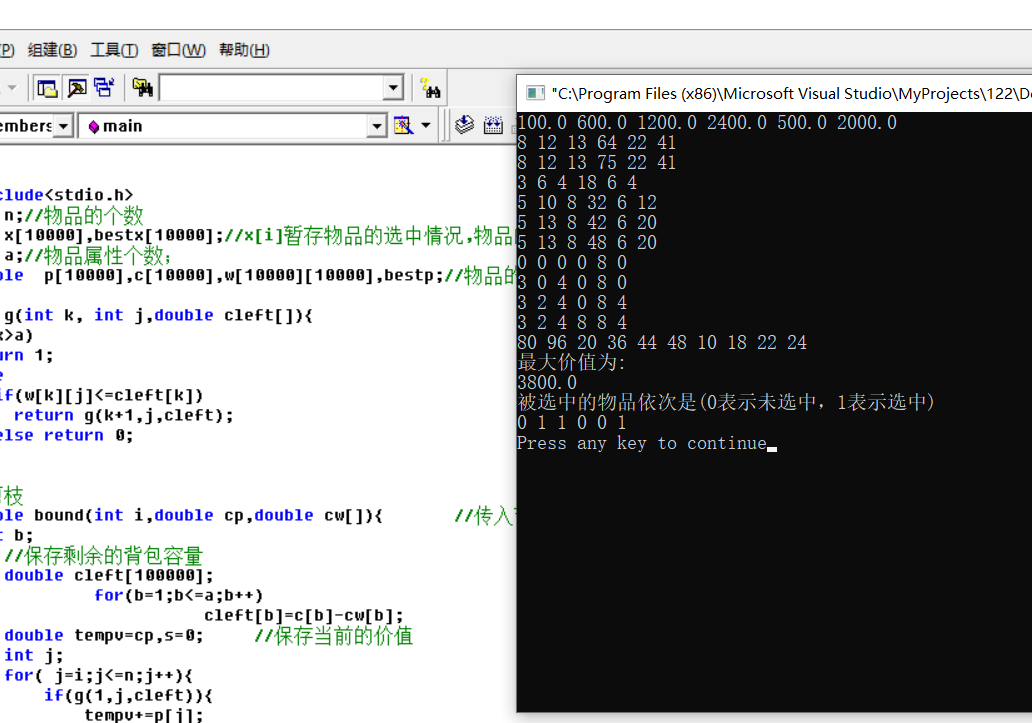
若有**a**个约束条件，**f()**函数的时间复杂度是**O(a/2)**,最坏情况下是**O(a)**;在进入左子树后需要对当前背包加入装入的**i**物品属性值，用到**for**语言，计算时间是**O(a)**;当需要进入右子树时将加入的**i**物品属性值减去，其计算时间是**O(a)**;

单个约束条件剪枝时需要用的计算时间是**O(n)**,最坏的情况下是**O(2n)**现在引入了**bound()**函数对右子树进行剪枝而在**bound()**函数里引入了**g()**函数进行判断，**g()**函数的功能是当前物品j需要装入背包是否都满足当前的剩余约束条件。每一次**g()**函数需要计算的时间是**O(a/2)**;最坏情况下是**O(a)**;当进入for语句进行计算的时候需要减少当前属性剩余量用到的时间是**O(a);**最终**bound()**函数的计算时间是**O(n(a^2)/2)**;最坏情况下是**O((a^2)n)**；

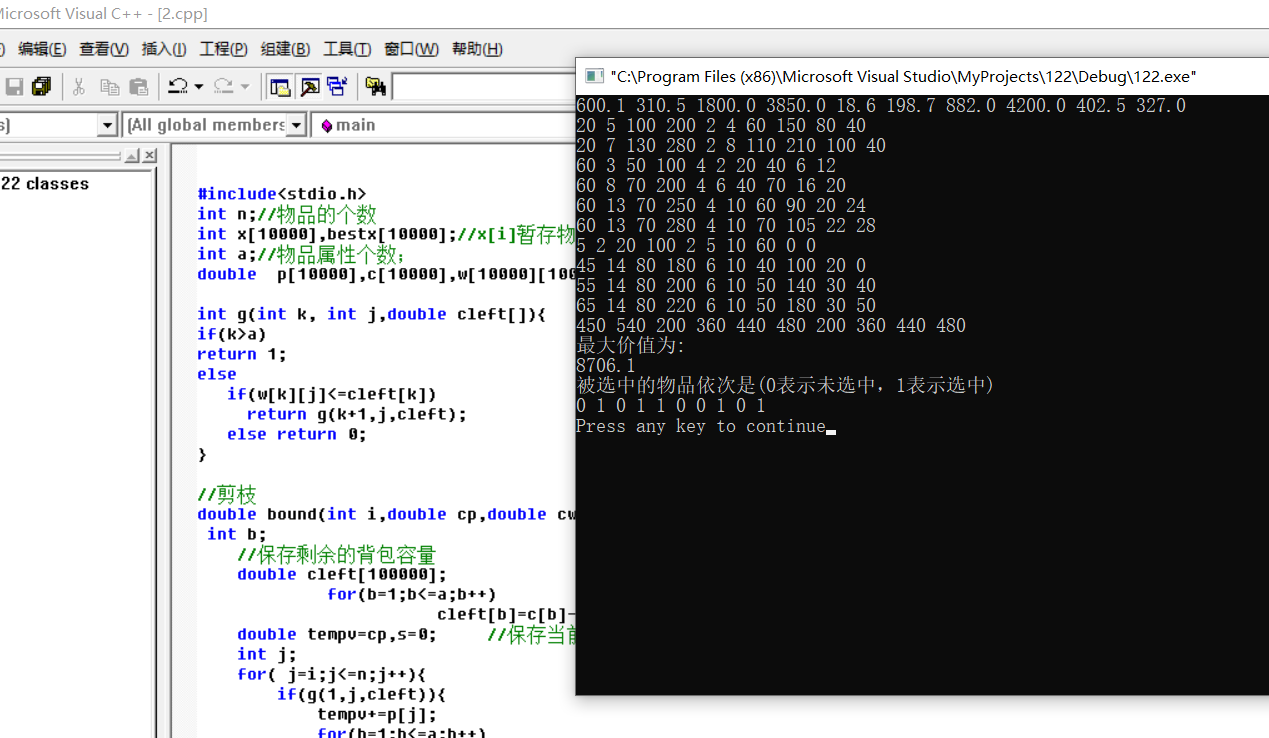
所以函数**Backtrack()**的时间复杂度是**O(n(2^n)(a^4)/2)**;

#### 如下截图:

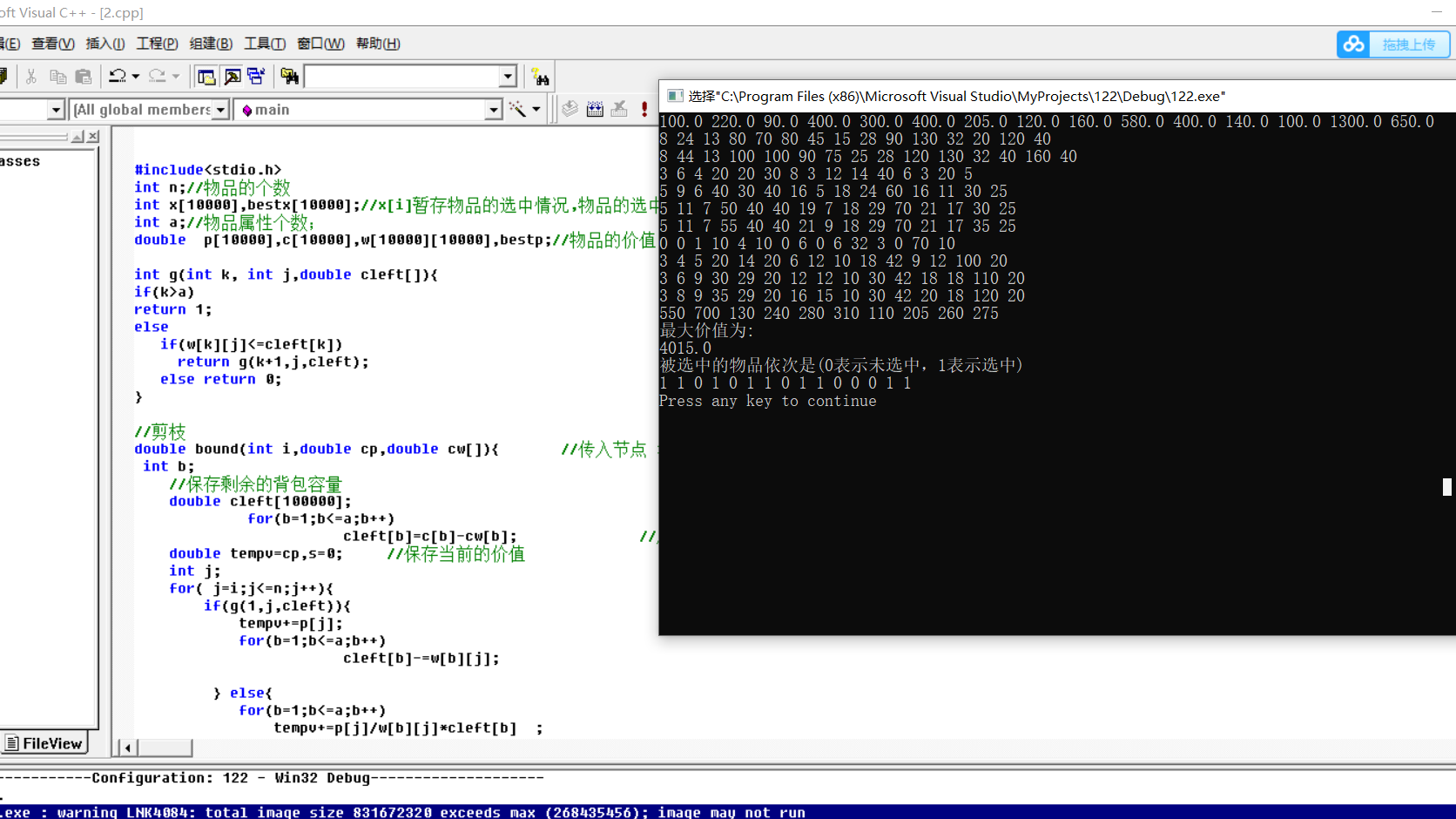
6个物品，10个约束条件，可以跑



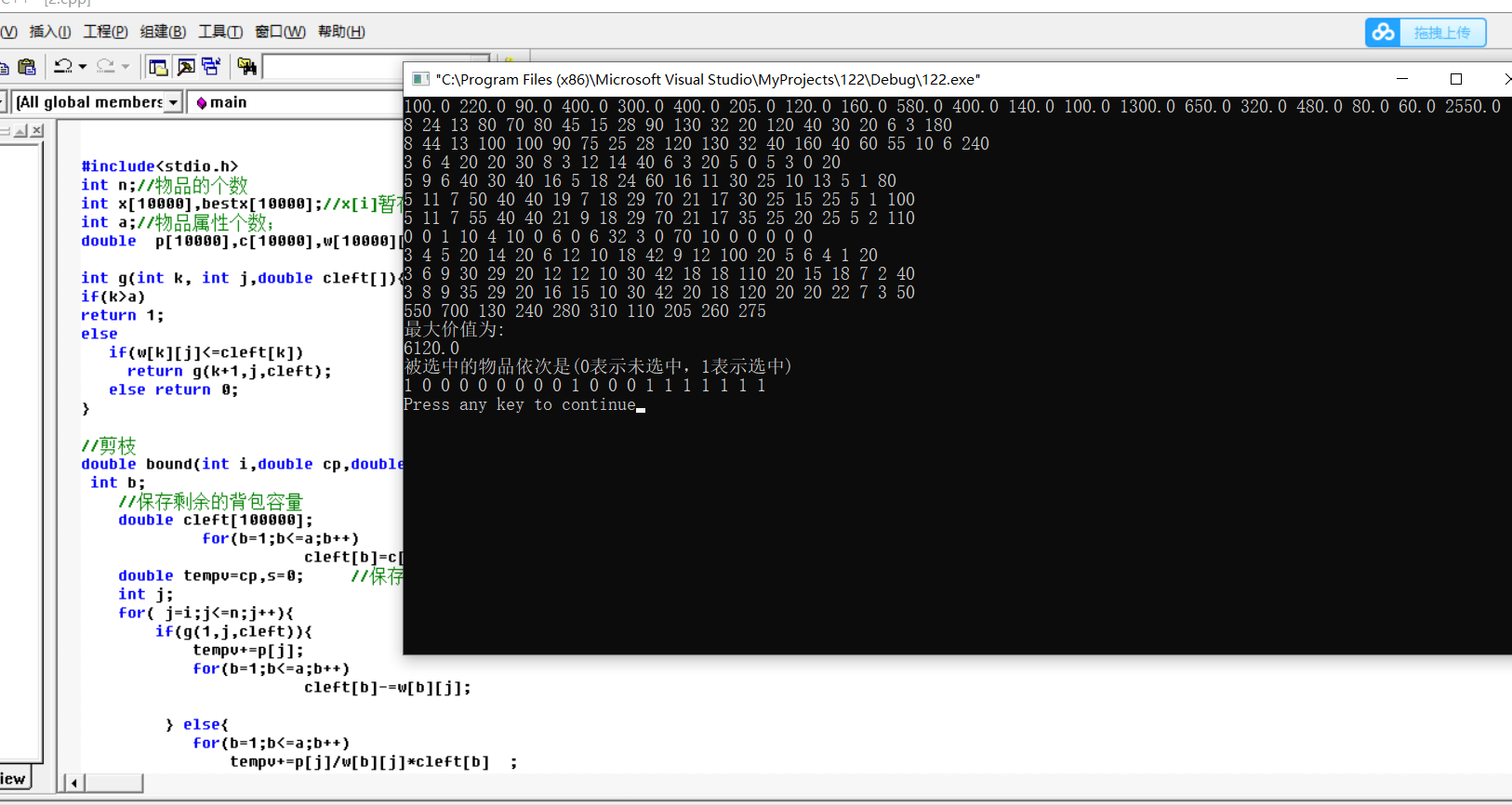
10个物品，10个约束条件，可以跑



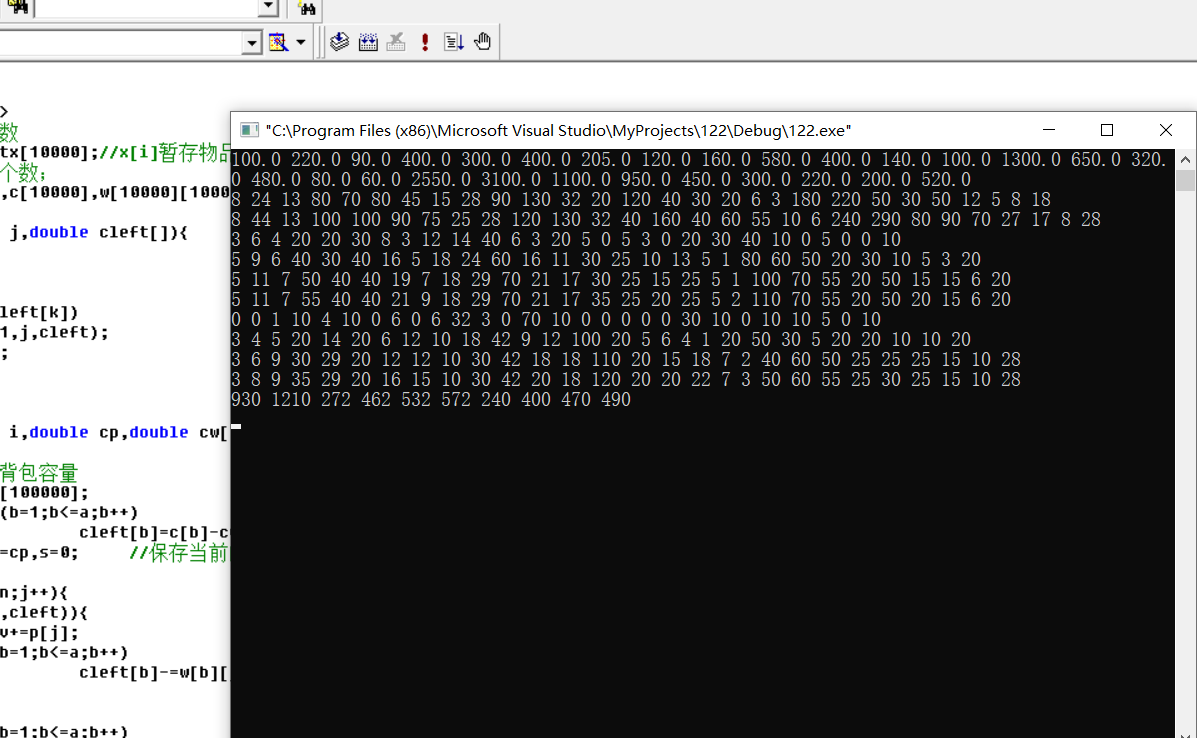
15个物品，10个约束条件，可以跑



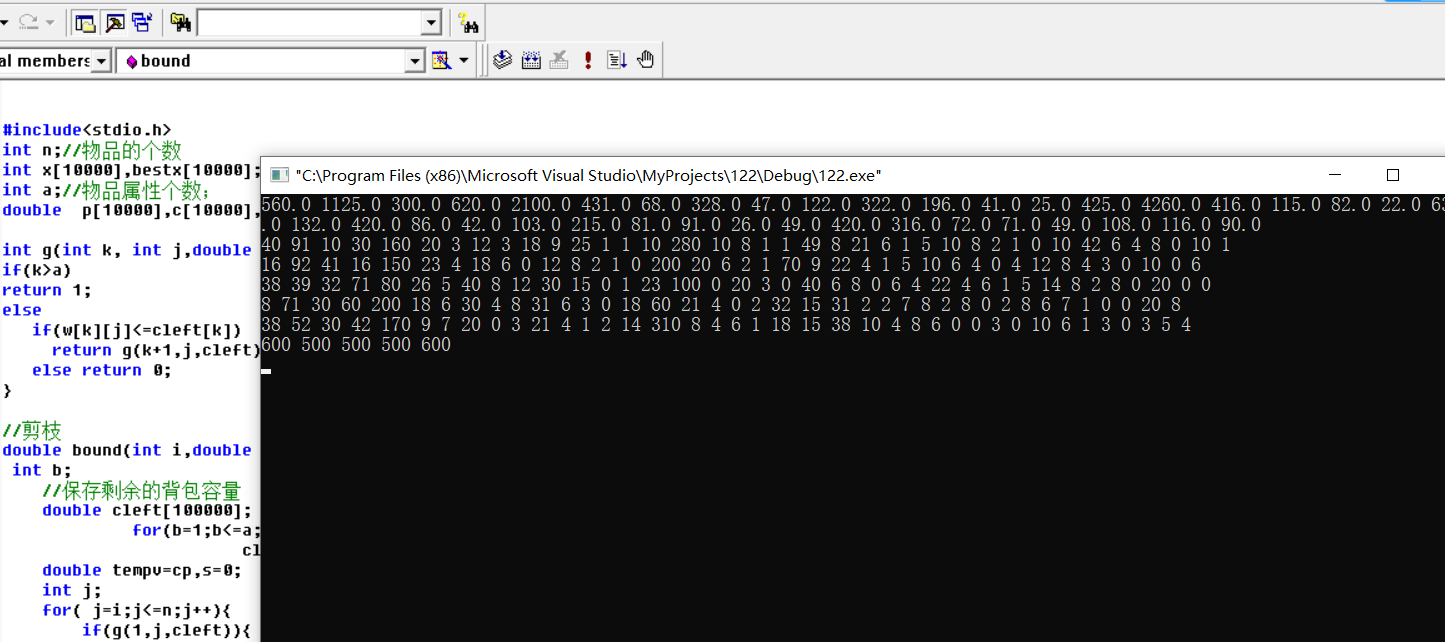
20个物品，10个约束条件，可以跑



28个物品，10个约束条件，跑不动



39个物品，5个约束条件，跑不动



50个物品，5个约束条件，跑不动

