要求：1、提交算法设计与分析文档；2、录屏讲解代码；

3、提交实现算法源代码；所有提交都需push到github上；4、录屏调试测试过程，测试用MKP算例mknapinfo.html；

# 题目：

## Multidimensional knapsack problem

There are 11 data files.

The first data file is mknap1.

This data file contains 7 test problems which are

the test problems from C.C.Petersen "Computational experience

with variants of the Balas algorithm applied to the selection

of R&D projects" Management Science 13(9) (1967) 736-750.

The problem to be solved is:

Max sum{j=1,...,n} p(j)x(j)

st sum{j=1,...,n} r(i,j)x(j) <= b(i) i=1,...,m

x(j)=0 or 1

The format of this data file is:

number of test problems (K)

then for each test problem k (k=1,...,K) in turn:

number of variables (n), number of constraints (m), optimal

solution value (zero if unavailable)

the coefficients p(j); j=1,...,n

for each constraint i (i=1,...,m): the coefficients r(i,j); j=1,...,n

the constraint right-hand sides b(i); i=1,...,m

The second data file is mknap2.

This data file contains 48 test problems taken

from the literature. The format of these problems

is described within the file.

The remaining data files are the problems solved in P.C.Chu and

J.E.Beasley "A genetic algorithm for the multidimensional knapsack

problem", Journal of Heuristics, vol. 4, 1998, pp63-86.

These data files are mknapcb1, mknapcb2, ..., mknapcb9

The format of these data files is the same as the format of mknap1

These data files each contain 30 test problems, the first ten problems

have a tightness ratio of 0.25, the second ten problems have a tightness

ratio of 0.50 and the last ten problems have a tightness ratio of 0.75 (see

the above paper).

The best feasible solution values found and the value of the LP

relaxation for these problems are given in the file mkcbres

The largest file is mknapcb9 of size 2000Kb (approximately)

The entire set of files are of size 5400KB (approximately).

github链接：

https://github.com/dajinglingpake/suanfayufenxi

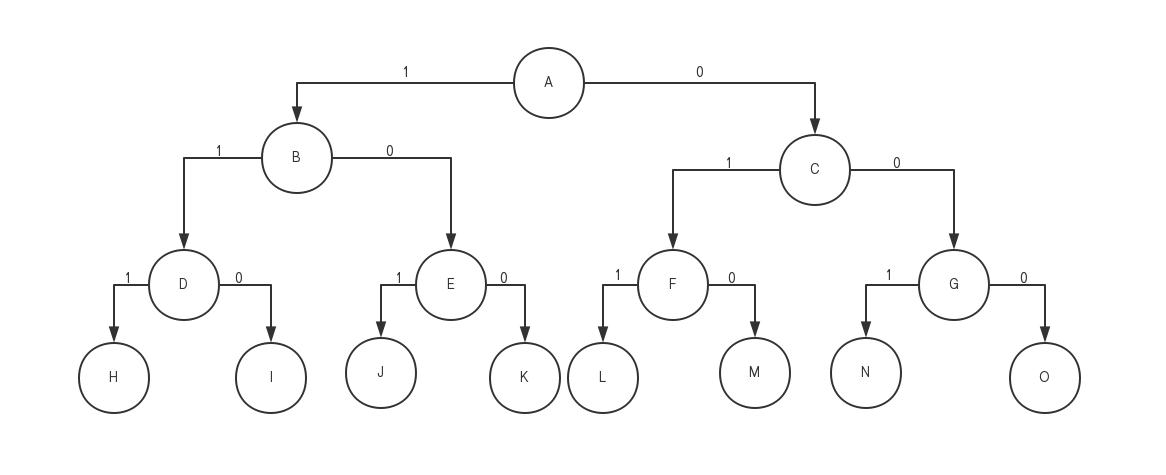
##### 问题要求：

给定n个物品和一个约束限制为c的背包，物品i的约束限制为Wi,其价值为Vi,根据判断如何选择符合所有约束条件而装入背包的物品（物品不可分割），使得装入背包的物品的价值为最大。

##### 题目分析：

当我们身在一棵搜索空间树中，站在一个i点举棋不定的时候。我们用它估算假设我们继续向下走，我们走完本段路会获得的总价值。假设我们如今有一个最大解，算出一个价值后和当前的最大解比較。假设能获得更大价值，我们就继续向下走，不能，果断放弃，这就是回溯。

考虑到每种物品只有2 种选择，即装入背包或不装入背包，并且物品数和约束条件已给定，要计算装入背包物品的最大价值和最优装入方案，可用回溯法搜索子集树的算法（回溯法解空间树）进行求解。  
问题是n个物品中选择部分物品，可知，问题的解空间是子集树。比如物品数目n=3时，其解空间树如下图，边为1代表选择该物品，边为0代表不选择该物品。使用x[i]表示物品i是否放入背包，x[i]=0表示不放，x[i]=1表示放入。回溯搜索过程，如果来到了叶子节点，表示一条搜索路径结束，如果该路径上存在更优的解，则保存下来。如果不是叶子节点，是中点的节点（如B），就遍历其子节点（D和E）。



算法设计（代码部分）：  
//**1. 定义物品有n种，背包容量为C，用v[i]和w[i]存储第i种物品的价值和重量（因为value有小数所以用双精度），最后用x[i]标记第i种物品是否装入背包，用bestx[i]来存储第i个物品的最优装载方案：**

#include<stdio.h>

int n;

int x[10000],bestx[10000],c[10000],w[10000][10000];

double bestv;

double v[10000];

**//2.用递归函数Backtrack (i,cv,cw)来实现回溯法搜索子集树（形式参数i表示递归深  
度，n用来控制递归深度，形式参数cv和cw表示当前总价值和总重量，bestv表示当前最优总价值）：**

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

**//3.如果i >n，则算法搜索到一个叶结点，判断当前总价值是否最优：**

{

if(i>n)

{

if(cv>bestv)

{

**//如果cv>bestv，更新当前最优总价值为当前总价值：**

bestv=cv;

**//存储第i个物品的装载方案( 1≤i≤n)）：**

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

bestx[i]=x[i];

}

}  
//**4. 用for循环讨论第i个物品装与不装，j>1时循环结束：**

else

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

{

x[i]=j;

**//如果当前总重量不大于背包容量（即当前所有限制均<=所有约束限制）：**

if(cw[1]+x[i]\*w[1][i]<=c[1]&&

cw[2]+x[i]\*w[2][i]<=c[2]&&

cw[3]+x[i]\*w[3][i]<=c[3]&&

cw[4]+x[i]\*w[4][i]<=c[4]&&

cw[5]+x[i]\*w[5][i]<=c[5]&&

cw[6]+x[i]\*w[6][i]<=c[6]&&

cw[7]+x[i]\*w[7][i]<=c[7]&&

cw[8]+x[i]\*w[8][i]<=c[8]&&

cw[9]+x[i]\*w[9][i]<=c[9]&&

cw[10]+x[i]\*w[10][i]<=c[10])

**//就更新当前总价值和当前总重量（即约束条件）：**

{

for(int k=1;k<=10;k++)

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

cv+=v[i]\*x[i];

**//对物品i+1调用递归函数Backtrack继续进行装载：**

Backtrack(i+1,cv,cw);

//**函数Backtrack调用结束后更新当前总价值和当前总重量：**

for(k=1;k<=10;k++)

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

cv-=v[i]\*x[i];

}

}

}

**//6.主函数部分：定义当前约束、价值初始值。输入物品个数与约束限制，从文本读取所有约束限制的字符串赋值到数组：**

int main()

{

int i,j;

int cw[100];

for(i=1;i<=10;i++)

cw[i]=0;

bestv=0.0;

scanf("%d",&n);

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

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

for(i=1;i<=10;i++)

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

FILE \*fr;

fr = fopen("1.txt", "r");

while(!feof(fr))

{

for(j=1; j<=10; j++)

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

fscanf(fr, "%d", &w[j][i]);

}

**//调用一次backtrack完成整个回溯搜索过程，打印最优总价值和最优装载方案。**

Backtrack(1,0.0,cw);

printf("%lf\n",bestv);

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

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

printf("\n");

return 0;

}

## 总结：

时间复杂度较大，无法判断计算物品数量超过34的数据，会不断循环至死。

时间复杂度大概是O（n2^n）

**完整代码：**

#include<stdio.h>

int n;//物品个数

int x[10000],bestx[10000],c[10000],w[10000][10000];//x[i]暂存物品的选中情况,最优装载，约束限制背包容量，约束条件

double bestv;//最优价值

double v[10000];//物品价值

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

{ //cw当前包内物品重量，cv当前包内物品价值

if(i>n)//回溯结束

{

if(cv>bestv)//如果当前价值＞最优价值

{

bestv=cv;//最优价值=当前价值

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

bestx[i]=x[i];//第i个最优装载是否装入

}

}

else

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

{

x[i]=j;//x[i]=1或x[i]=0

if(cw[1]+x[i]\*w[1][i]<=c[1]&&

cw[2]+x[i]\*w[2][i]<=c[2]&&

cw[3]+x[i]\*w[3][i]<=c[3]&&

cw[4]+x[i]\*w[4][i]<=c[4]&&

cw[5]+x[i]\*w[5][i]<=c[5]&&

cw[6]+x[i]\*w[6][i]<=c[6]&&

cw[7]+x[i]\*w[7][i]<=c[7]&&

cw[8]+x[i]\*w[8][i]<=c[8]&&

cw[9]+x[i]\*w[9][i]<=c[9]&&

cw[10]+x[i]\*w[10][i]<=c[10]) //如果当前包内总重＋第i个物品重量约束<=背包容量

{

for(int k=1;k<=10;k++)

cw[k]+=w[k][i]\*x[i];//当前总重=当前总重＋第i个物品重量

cv+=v[i]\*x[i];//当前总价=当前总价＋第i个物品价值

Backtrack(i+1,cv,cw);

for(k=1;k<=10;k++)

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

cv-=v[i]\*x[i];//当前总价=当前总价-第i个物品价值

}

}

}

int main()

{

int i,j;

int cw[100];

for(i=1;i<=10;i++)

cw[i]=0;

bestv=0.0;

printf("物品个数是:\n");

scanf("%d",&n);

printf("物品的价值是:\n");

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

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

printf("约束限制是:\n");

for(i=1;i<=10;i++)

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

FILE \*fr;

fr = fopen("1.txt", "r");//打开文件名

while(!feof(fr))

{

printf("约束条件自动输入成功.\n");

for(j=1; j<=10; j++)

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

fscanf(fr, "%d", &w[j][i]);

}

Backtrack(1,0.0,cw);//当递归函数达到条件i=1时外层调用结束执行下一步

printf("最优价值为:\n");

printf("%lf\n",bestv);

printf("选中了的物品:(0未选中，1选中)\n");

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

printf("%d",bestx[i]);//输出最优价值

printf("\n");

return 0;

}