综合运用学习过的算法设计与分析方法：设计MKP问题的一种算法，分析复杂性，实现算法（语言不限），调试测试，讲解代码。

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).

算法代码github链接：

https://github.com/dajinglingpake/suanfayufenxi/commit/3257c4366998c41181283f2799d0b07d8857c677

参考代码

<https://www.cnblogs.com/huanxiyun/articles/5391571.html>

报告模板

<https://wenku.baidu.com/view/bcd62e517e21af45b307a8cf.html>

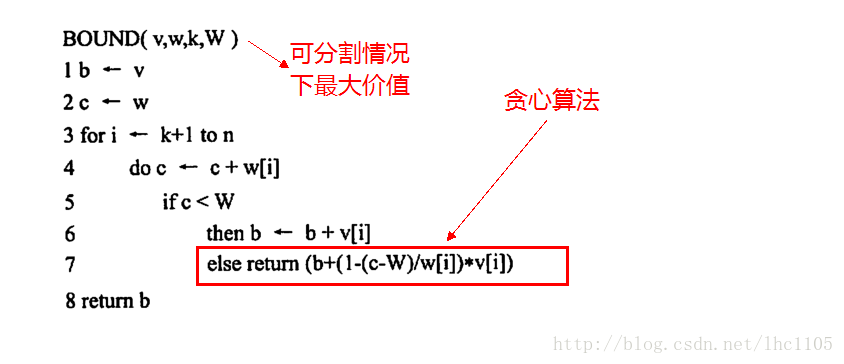
# **什么是界限函数？**

        例如以下图：

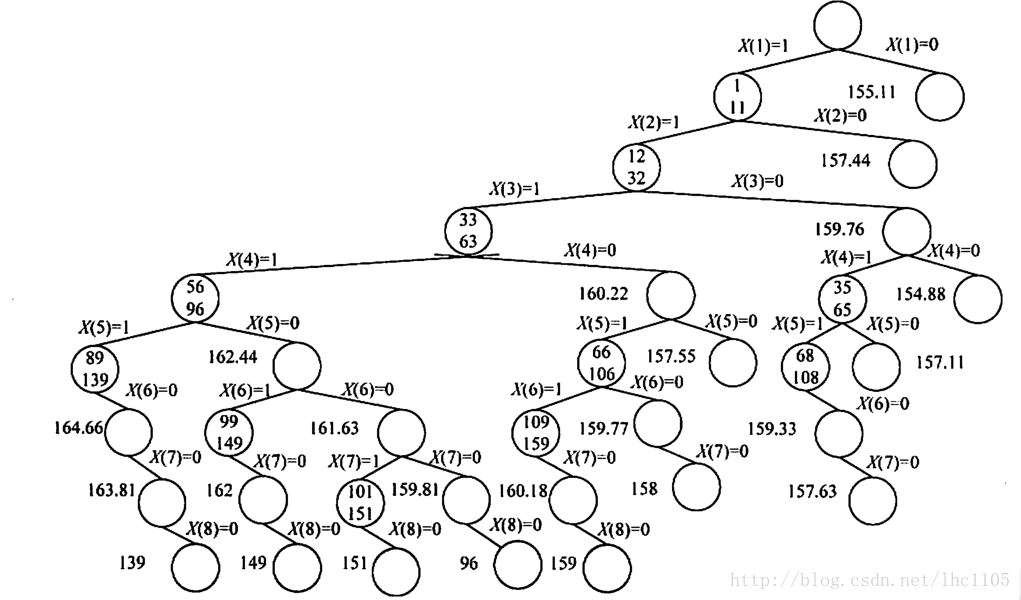
            当我们身在一棵搜索空间树中，站在一个K点举棋不定的时候。我们能够用它估算假设我们继续向下走，我们走完本段路会获得的总价值。

假设我们如今有一个最大解，当我用界限函数算出一个价值后和我当前的最大解比較。假设能获得更大利益。我们选择继续向下走，假设不能。果断放弃。

         从下图中的伪代码能够看出，我们计算后半段最大价值的时候。使用的还是一个贪心算法。尽管切割的情况是不被同意的，可是我们能够用这个结果来进行估算。



回溯法得到的搜索空间树:



# **什么时候使用界限函数？**

       数学一点儿的说法是：当X[i]=0时。

   通俗一点说：当进入右结点的时候。

# **怎样回溯的问题？**

        向上回溯到第一个不是0的结点（而且这个结点不是顶点）。

# **求解思路**

               如上图搜索树。在建立搜索树之前，我将全部的物品依照V/W（价值重量比）从大到小排序。然后从第一个開始。依次向背包（背包大小110）中放入，放到第6个的时候，这时候发现6太大了，不能装入了，这时候用界限函数推断下，假设继续下去。会获得的最大价值，得出这个价值后，和上几次查找得到的最大价值对照，可是由于我们在这之前还没有获得过别的解，所以界限函数再和最大价值的初值-1比較的时候，总是会选择继续。这样我们就得到了一个解139.然后我们回溯到第一个X[i]不等于0的地方，此处为X[5],然后将X[5]置为0。这时候X[5]置0了，我们就先用界限函数推断下X[6]到X[8]的情况，得出了个164.44，这个比我们上次得到的第一个解也是最大的解139大。说明向后继续，肯会出现一个比139还大的解，所以我们选择向后继续。。

但我们回溯到X[1]的时候。我们将X[1]置0。这时候用界限函数估算下物品2到物品8可能获得的最大价值，发现是155.11，比我们实际得到的最大解159还小，然后果断放弃，再向上回溯，发现这已经到了尽头了，然后停止。

                  结合曾经的N皇后问题，N皇后问题是我一行一行的放皇后。假设当下一行放到最后一个位置的时候还是会产生攻击。这时候我们就调整上一行皇后的位置，然后再回到本行从第一个開始放。对照0-1背包，这个是完毕一次求解过程，然后就回溯继续求解。

            所以，回溯法是先一直做，做不下去了，然后才向回走。

# **小结：**

      0-1背包问题的用回溯法解决最開始提出的三个问题挺关键的，试想，假设一个问题足够大的话。用界限函数可以砍掉非常多不合条件的子节点。极大的提高了效率。

01背包回溯法

#include<stdio.h>

int n,c,bestp;//物品的个数，背包的容量，最大价值

int p[10000],w[10000],x[10000],bestx[10000];//物品的价值，物品的重量，x[i]暂存物品的选中情况,物品的选中情况

void Backtrack(int i,int cp,int cw)

{ //cw当前包内物品重量，cp当前包内物品价值

int j;

if(i>n)//回溯结束

{

if(cp>bestp)//如果当前价值＞最优价值

{

bestp=cp;//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(j=0;j<=1;j++)

{

x[i]=j;//x[i]=1或x[i]=0

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

{

cw+=w[i]\*x[i];//当前总重=当前总重＋第i个物品重量

cp+=p[i]\*x[i];//当前总价=当前总价＋第i个物品价值

Backtrack(i+1,cp,cw);

cw-=w[i]\*x[i];//当前总重=当前总重-第i个物品重量

cp-=p[i]\*x[i];//当前总价=当前总价-第i个物品价值

}

}

}

int main()

{

int i;

bestp=0;

printf("请输入背包最大容量:\n");

scanf("%d",&c);

printf("请输入物品个数:\n");

scanf("%d",&n);

printf("请依次输入物品的重量:\n");

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

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

printf("请依次输入物品的价值:\n");

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

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

Backtrack(1,0,0);//当递归函数达到条件i=1时外层调用结束执行下一步

printf("最大价值为:\n");

printf("%d\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");

return 0;

}