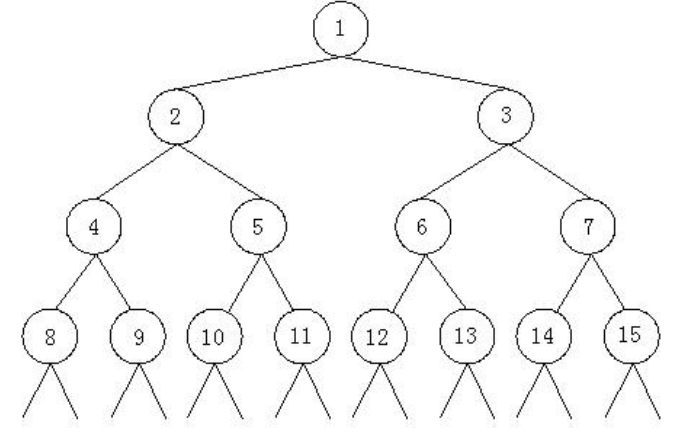
**多维背包问题（回溯法）解释文档**

（此文章语言表达比较通俗，请老师谅解！）

多维背包，主要与0-1背包的区别是：0-1背包的约束条件只有一个，筛选放入的物品，比如这个约束条件是体积，背包最大体积容量为10，则放入的物品其体积之和不能超过10；要是多维背包的话，则是通过多个约束条件筛选出放入的物品，多个约束条件，如最大体积容量、最大重量限制等条件，所以每个约束条件分别都有一个约束值来限制背包的承载量

我使用回溯法来解决多维背包问题，只要懂得回溯法及利用回溯法解决0-1背包问题，多维也就不在话下，0-1背包是只有一个条件限制，所以通过通过解决0-1背包解决多维背包问题，通过一个条件筛选物品的操作是0-1背包问题，那我们就重复这一操作，也基本就解决了多维背包问题，那接下来我讲一下利用回溯法解决0-1背包问题：

 假如：背包最大体积容量为10，物品一：体积2，价值5；物品二：体积6，价值4；物品三，体积3，价值7；物品四，体积4，价值4。

请看上面这个图，物品放入背包，放或者不放，两种选择，所以用二叉树很好的诠释着这个问题，我们把每层看做一个物品，以为判断物品一放或不放两个分支，物品二也如此，以此类推，物品二物品1为物品一，2和3为物品二，以此类推，然后就形成了一个上面的图。

假设规定向左开支就是放入，享有开支就是不放入，首先物品一从1开始，试放入，转到2，则占空间为2，2<10，则放入还剩8的空间；物品二从2开始，试放入，转到4，占的空间为6，6<8，则放入，还剩2的空间；物品三从4开始，试放入，3>2，则不放入，转到9，以此类推，直到到达叶子结点开始回溯，回到物品二从4开始，试着不放入转到5然后在继续判断，直到把所有的节点判断结束后，整个流程就结束了，借着0-1背包这个解决方案，多维背包就开始多次重复以上工作。

以下是我的代码分析：

n：物品数

m：约束条件数

i：物品的标记（下标）

v1：物品价值

data2:接收我们输入的物品条件

data3：物品条件及条件约束值

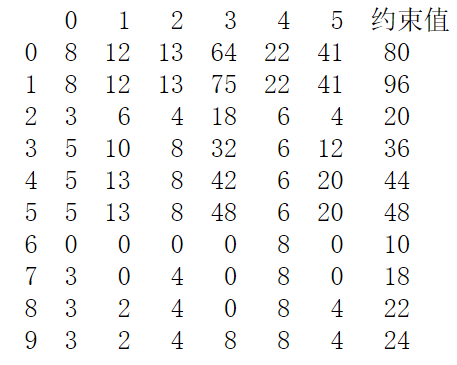
v2：存放途中的结果

v\_max：若是途中的结果大于v\_max，则v\_max=v2

刚开始，我们用n、m、v1、data2接收我们输入的各种数据

并把物品条件和约束值放在data3里

data3:



接着建立一个函数f(i)

判断是否到达叶子结点：

若到达，则把筛选得的物品的价值相加——v2，再用v2与v\_max相比，若v2>v\_max，则v2赋值给v\_max。

若没到达，则开始筛选，第i个物品并判断是否能放入：(data3['约束值']-data3[i])>=0

放入则：data3['约束值']=data3['约束值']-data3[i]

#（上面这行）约束值将物品属性条件值减掉

v2=v2+v1[i] #将放入物品的价值加入v2

f(i+1) #调用函数进行下一个物品的执行

当到达叶子结点时

则开始回溯：v2=v2-v1[i]

data3['约束值']=data3['约束值']+data3[i]

在程序的最后调用f（0），开始执行f（i）函数