# 算法引论

算法：适合用计算机实现的解决问题的方法；数据结构：便于算法操作的组织数据的方法；

算法是指解决问题的过程与方法：输入、输出、确定性、有限性；

算法的问题对象（操作的数据）：数字、字符串、局针、声音.....

算法涉及到的运算：矩阵运算、加减乘除逻辑运算等.....

高级语言可以抽象表达算法；

算法从自然语言转化为代码需要做的事情：

* 选用数据模型、确定初始状态、结果状态，探索从初始状态到结果状态的关系，怎么从初始状态到结果状态；
* 自顶向下，逐步求精，设计算法顶层运算，在设计底层运算；顶层步骤是定义在数据模型级别上的步骤，是算法的主干，是抽象的，操作的是数据模型内的对象；底层运算是微观运算，是与数据模型定义的运算，具体实现的细节依赖数据模型的内部表示，为了让2层不影响，需要定义接口抽象，这个接口就是抽象数据类型（ADT）；
* 抽象数据类型实际就是定义了一个类型。

# 动态规划

与分治法类似，基本思想也是将问题分解为子问题，通过子问题的解得到问题的解；不同的地方是，动态规划问题分解的子问题往往不是相互独立的，若用分治法解决，则分解了很多重复的子问题，耗费了指数级的时间，实际上，去除冗余，子问题的数量是多项式级别的，保存已解决的子问题的结果，避免重复计算，需要一个表来记录所有已解决的子问题答案；这就是动态规划，具体的动态规划算法是多种多样的，但是都具有相同的填表形式。动态规划算法适合来解最优化问题，通常可以按照以下步骤设计动态规划算法：

* 找出最优解的性质，并刻画其结构特征；
* 递归的定义最优值；
* 以自底向上的方式计算出最优值；
* 根据计算最优值时得到的信息，构造最优解。

## 3.1 矩阵连乘问题

给定n个矩阵，相邻的矩阵都是可乘的，矩阵乘法满足结合律，所以可以是任意的计算次序；可以用加括号的方式唯一确定一个执行顺序；加括号的计算乘积的递归定义如下：

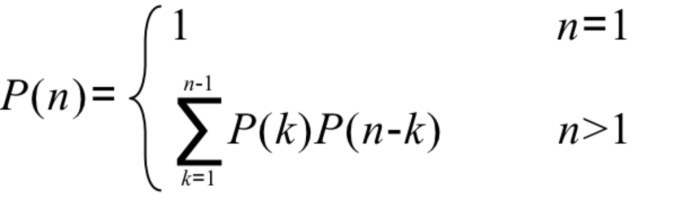
* + 单个矩阵是完全加括号的，是最高优先级；
  + 乘积A是完全加括号的，表示A的乘数矩阵B与C都是完全加括号的即。

矩阵乘积顺序关系到计算量，计算2个矩阵的标准算法如下：



计算量是3次循环的数乘；顺序对计算量影响很大，考虑3个矩阵的相乘，，维数分别是，如果顺序计算，则乘积需要次数乘，如果按照的顺序计算的，需要的数乘次数为；于是有了矩阵乘积的最优计算次序问题，就是要求最少的数乘次数。

穷举搜索法可以解决这个问题，把每种可能都计算出来，选出最优的；但计算量太大，对n个矩阵的乘积，设不同的计算次序种类为，由于可以在第k个与k+1个之间将矩阵连乘分为2个子连乘，，在对2个子矩阵连乘加括号计算次序种类，最终可以得到原连乘矩阵的加括号方式；递推式如下：



解方程，得到这是一个组合数，是指数级别的；下面按照动态规划的算法计算：

* 分析最优解的结构：刻画最优解的结构特征将矩阵连乘简记为，计算的最优次序，设这个计算次序在k与k+1位置处断开，则完全的加括号的形式为，依此计算，则先计算A[1:k]与A[k+1,n]在将计算结果相乘，这个问题的一个关键的特征是，计算A[1:n]的子矩阵的次序也是最优的，矩阵连乘计算次序的最优解包含其子问题的·最优解，这种性质叫做最优子结构问题，问题的最优子结构性质是问题可以被动态规划算法求解的显著特征；
* 建立递归关系：递归的定义最优值，设计算的最少数乘次数为，当i=j时，=Ai,为单一矩阵无需计算，m[i][j]=0；当i<j时，可利用最优子结构计算，设在k处断开，则由于不知道k的位置，k有j-i种可能，k是使这个计算量达到最小的位置从而m[i][j]的递归定义如下：  
  ，这个式子计算出了最小数乘次数与连接断开位置k；
* 计算最优值：根据上面的递归式计算将消耗指数级的时间，事实上，对于1<=I<=J<=n，不同的有序对，对应不同的子问题，不同子问题的个数最多有  
  个，许多子问题被重复计算了，用动态规划算法解决这个问题，依据递归式自底向上的方式计算，计算时，保存已解决的子问题答案，每个子文推只计算一次，后面需要是查询一次，得到多项式的时间的算法，这里看书啊，写不动了；
* 构造最优解：

## 3.2 动态规划算法的基本要素

动态规划算法的有效性依赖于2个性质：

* 最优子结构；
* 子问题重叠。

1. 最优子结构：设计动态规划算法的第一步就是刻画最优解的结构，当问题的最优解包含子问题的最优解时，称该问题具有最优子结构性质，分析最优子结构性质时，所用的方法具有普遍性：就是假设法，假设由问题的最优解导出的子问题的解不是最优的，然后在设法说明在这个假设下，可构造处比原问题最优解更好的解，从而导致矛盾；利用问题的最优子结构性质，以自底向上的方式递归的从子问题的解逐步构造出整个问题的最优解。
2. 重叠子问题：递归分解的子问题具有重叠性质，不需要重复计算，，只需要有个表来存储计算过的值；
3. 备忘录方法：动态规划方法的变形，动态规划算法是自底向上递归，备忘录方法是自顶向下，备忘录方法为每个解过的子问题建立了备忘录以备需要时查看，避免了相同子问题的重复求解。备忘录方法为每个子问题建立一个记录项，初始化时，该记录项存入一个特殊值，表示子问题没有被解决，求解过程中，每个子问题，首先查看记录项，若是初始值，则问题第一次遇到，计算该子问题并保存，若不是初始值，则已解决。

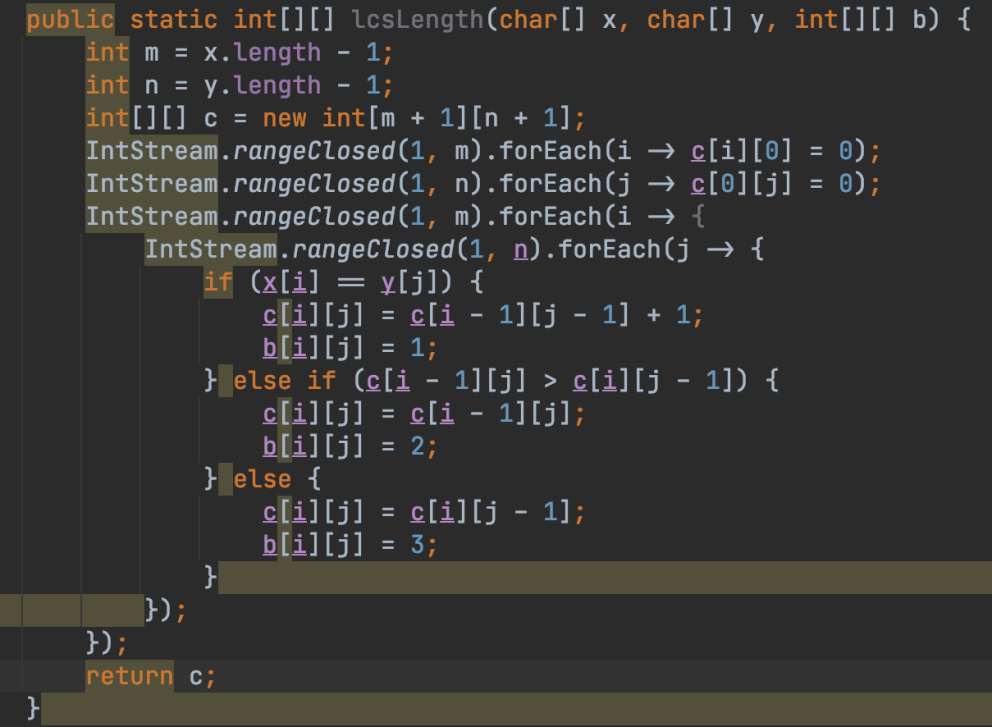
当一个问题的所有的子问题都至少需要解一次时，动态规划算法与备忘录方法实际是一样的，但是如果子问题空间内，有些子问题不需要计算时，则备忘录方法比较好，因为它只计算那些确实需要解决的子问题。

## 3.3 最长公共子序列

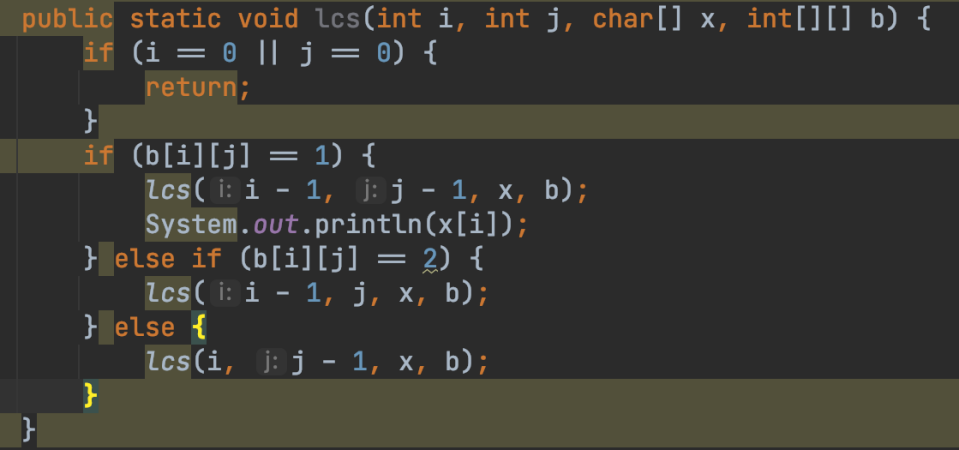
一个给定序列的子序列就是从序列中递增的取出一些元素，构成的序列；最长公共子序列就是2个序列的最长的公共子序列；可以使用动态规划算法解决。

1. 最长公共子序列的结构：最长公共子序列具有最优子结构性质，设序列，序列，的最长公共子序列为，则有3条性质是最优子结构性质：
2. 若，则，且是与的最长公共子序列；
3. 若且，则是与Y的最长公共子序列；
4. 若且，则是与X的最长公共子序列；
5. 子问题的递归结构：根据上面的3个子结构性质，可以得到递归计算公式，用c[i][j]记录Xi与Yj的最长公共子序列的长度

1. 计算最优值：直接根据上述的递归式写代码，时间复杂度是指数级别的，同时有很多子问题重复计算，通过考虑得到子问题的个数是m\*n个，通过动态规划算法自底向上计算最优值就可以得到最长的公共子序列，用c[i][j]记录Xi与Yj的最长公共子序列的长度 ，b[i][j]记录长度是通过那一个种类的子问题的得到的，通过b[i][j]的路径就可以得到子序列的内容，代码如下：



1. 构造最长公共子序列：根据数组b的计算路径可以获得最长公共子序列，代码如下：

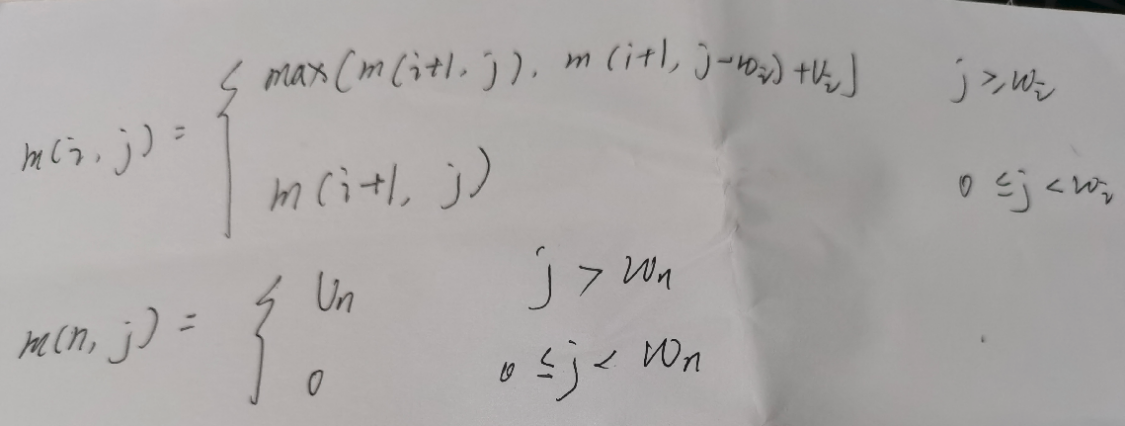


## 3.4 凸多边形最优三角剖分

## 3.9 0-1 背包问题

0-1背包问题的定义：给定n种物品与一个背包，物品i的重量是w，其价值是v，背包的容量为·C，如何选择装入背包中的物品，使得装入背包中的物品的总价值最大？物品只能装入背包或者不装入背包，不能装入多次，所以称为0-1背包问题，此问题的形式化表达是：

给定，要求找出n元0-1向量，使得且达到最大；数学化的表达形式是：

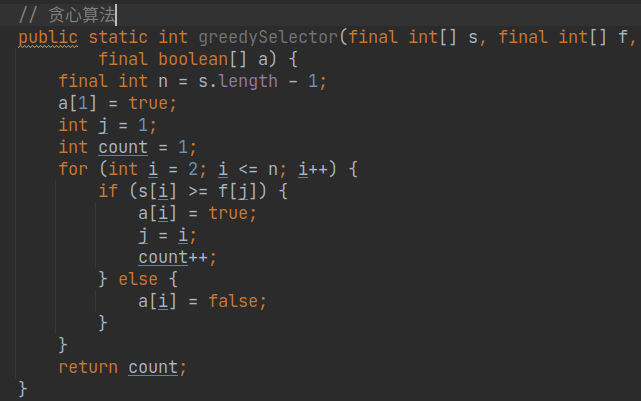
1. 最优子结构性质：0-1背包问题具有最优子结构，设是所给0-1背包问题的一个最优解，则 是下面相应子问题的一个最优解，
2. 递归关系：设最优值是，背包容量是j，i表示可选择的物品为i,i+1,i+2,....n,则可以建立如下的递归关系式：
3. 算法描述：

# 贪心算法

当一个问题具有最优子结构问题时，可以用贪心算法解决，贪心算法总是做出对于当前看来最好的选择，贪心算法并不从整体最优考虑，所以的到的结果可能是接近最优解的解。

## 4.1 活动安排问题

活动安排问题是贪心算法的一个很好的例子，设有n个活动的集合E={1,2,...,n}，每个活动都要使用一个资源，比如教室，场地等，同一个时间只能有一个活动使用这个资源，每个活动i都有使用资源的左开右闭区间段,如果任意2个活动的时间段不重合，则2个活动就是相容的，在活动的集合中选出最大的相容活动子集合；给出的条件是个活动的起始、结束数组分别是s与f，且活动的顺序按照f的时间升序排列，按照贪心算法的解决方式则是：



通过数学归纳法，可以证明贪心算法对这个问题得到的解就是最优解。

## 4.2 贪心算法的基本要素

贪心算法解决的问题有2个基本要素：

* 贪心选择性质：整体最优解可以通过局部最优的选择既贪心选择来达到，与动态规划算法的主要的区别，贪心算法仅在当前状态下做出最好的选择，可以以来以前做的选择，不依赖将来的选择或者子问题的解，贪心算法是自顶向下的，没做出一个最优选择，问题就简化为子问题；证明贪心选择性质，要证明局部最优可以得到整体最优，首先考察问题的最优解，并证明可以修改这个最优解。
* 最优子结构性质：当一个问题的最优解包含子问题的最优解时，此问题具有最优子结构性质。

贪心算法与动态规划算法的差异是什么呢？

* 0-1背包问题：给定，需要找到n元0-1向量，使得且最大；
* 背包问题：与0-1背包问题类似，但是背包问题可以选择物品的一部分装入背包，不是必须全部或者完全不装入给定，需要找到n元0-1向量，使得且最大；

上面2个问题都是具有最优子结构的问题，但是背包问题可以使用贪心算法解决，先计算每种物品的单位重量的价值，依靠单位价值最大的原则依次装入，最终得到最大价值。算法面熟如下：



## 4.3 最优装载

已写过，但是丢失了

## 4.4 哈夫曼编码

哈夫曼编码广泛应用于数据文件的压缩，比如根据字符出现的频率给字符编码，对比定长码来说总的码长会减少很多，达到压缩的效果；任一字符的代码都不是其他字符代码的前缀就是前缀码，使用哈弗曼树来表示前缀码特别方便，叶节点就是所表示的字符，路径代表0/1编码，哈夫曼算法是构造最优前缀码（码长最小）的贪心算法方案，通过自底向上的方式构造最优前缀码的二叉树表示。代码描述如下：



因为哈夫曼算法具有贪心选择性质与最优子结构性质，所以使用贪心算法是正确的。

## 4.5 单源最短路径

给定带权有向图，给定V中的一个点，叫做源，求到其他点的最短长度（权重之和）这个叫做单源最短路径问题。

Dijkstra是解决这个问题的贪心算法，基本思想：设置已确定最短定点S并不断的做贪心选择来扩充，初始S中仅有源，设u是G的某个定点，从源到u，中间只经过S中的点的路径中选出最短的一条，这个即最短路径，每次从V-S集合中的点中选出一个，直到全部完成。具体算法如下：



* 贪心选择性质：每次获取的u点都是最短路径，没有更短的路径了，如果存在，那肯定不是u了；
* 最优子结构性质：

## 4.6 最小生成树

# 第五章 回溯法

回溯法是通用解题法，可以系统的搜索问题的所有的解，回溯法就是深度优先搜索找问题解的一个算法。一般是解决树结构的问题，算法按深度搜索树的节点，先判断节点是否可能包含解，如果一定不包含，向上回溯，否则深度搜索，找到一个解就返回，否则遍历所有的子树。

## 5.1 回溯法的算法框架

回溯法解问题首先要定义问题的解空间，意思就是确定问题的所有的解。从解空间中找到最优解。然后将解空间使用树或者图组织起来，以便使用回溯算法。

回溯法解空间树时，采用2种策略来避免无效搜索：

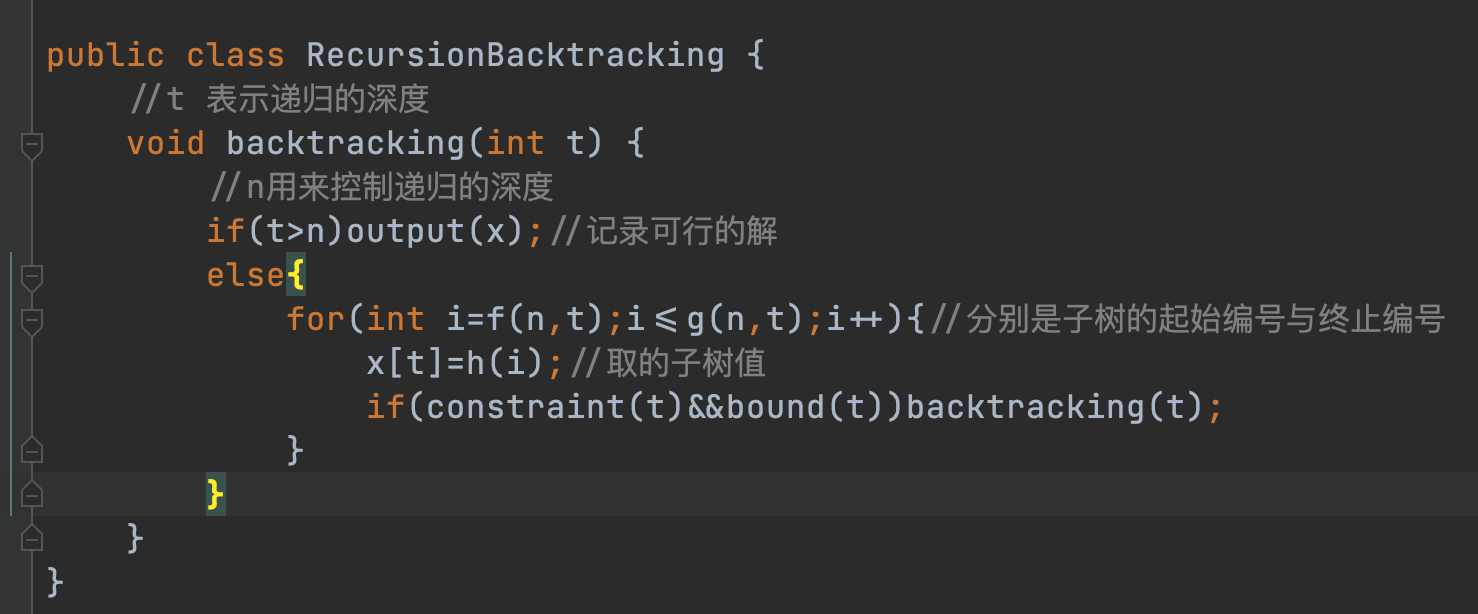
* 约束函数在扩展节点处剪去不满足约束的子树；
* 用限界函数剪去得不到最优解的子树；

这2种函数统称为剪枝函数。

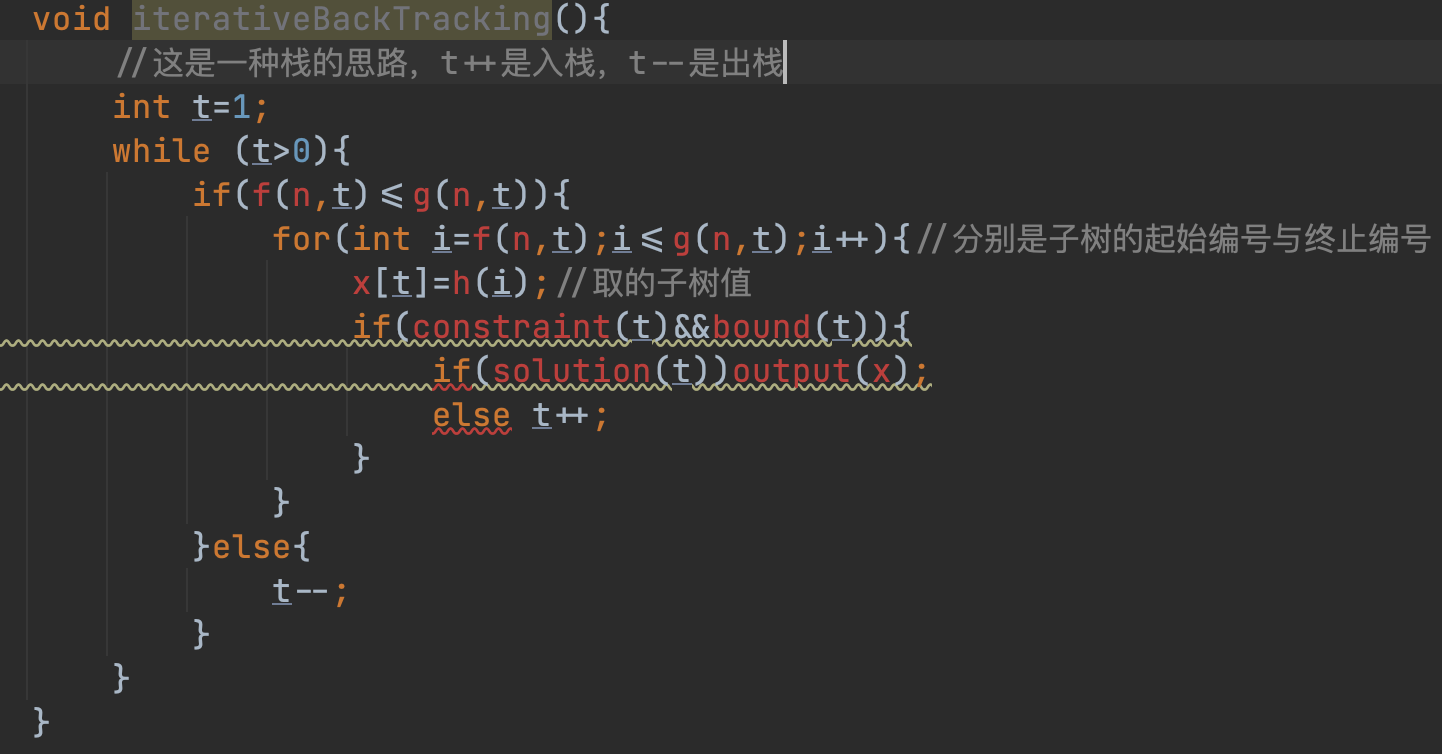
回溯法解题步骤：

* 定义问题的解空间；
* 确定易于搜索的解空间结构；
* 以深度优先的方式搜索解空间，在搜索的过程中使用剪枝函数避免无效搜索。

回溯法使用递归的方式比较好。



也可以是用迭代回溯：



当所给的问题是从n个元素的集合S中找到S中满足某种性质的子集时，相应的解空间叫做子集树，就是不关系顺序，通常是满二叉树的形式，符合满二叉树的特点。

当n个元素有的顺序是有特定意义，就是顺序不同就不同，解空间树叫做排列树，排列树有n！个叶节点。

## 5.2 装载问题

问题描述：有一批集装箱装上2艘载重量是c1,c2轮船，集装箱i的重量是wi满足条件，是否有方案全部装上船。