**算法设计之五大常用算法设计方法总结**

来源 http://blog.csdn.net/zolalad/article/details/11393915

**一、【分治法】**

 在计算机科学中，分治法是一种很重要的算法。字面上的解释是“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题，再把子问题分成更小的子问题……直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。**这个技巧是很多高效算法的基础**，如排序算法(快速排序，归并排序)，傅立叶变换(快速傅立叶变换)……等。任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，越容易直接求解，解题所需的计算时间也越少。例如，对于n个元素的排序问题，当n=1时，不需任何计算。n=2时，只要作一次比较即可排好序。n=3时只要作3次比较即可，…。而当n较大时，问题就不那么容易处理了。要想直接解决一个规模较大的问题，有时是相当困难的。

分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同问题，以便**各个击破，分而治之**。

分治策略是：对于一个规模为n的问题，若该问题可以容易地解决（比如说规模n较小）则直接解决，否则将其分解为k个规模较小的子问题，这些**子问题互相独立且与原问题形式相同**，递归地解这些子问题，然后**将各子问题的解合并得到原问题的解**。这种算法设计策略叫做**分治法**。

如果原问题可分割成k(1<k≤n)个子问题， 且这些子问题都可解并可利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的子问题往往是原问题的较小模式，**这就为使用"递归技术"提供了方便**。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易直接求出其解，这自然导致递归过程的产生。"**分治"与"递归"像一对孪生兄弟，经常同时应用在算法设计之中，并由此产生许多高效算法**。

**分治法所能解决的问题一般具有以下几个特征：**

1) 该问题的规模缩小到一定的程度就可以容易地解决

2) 该问题可以分解为若干个规模较小的相同问题，即该问题具有**最优子结构**性质。（前提）

3) 利用该问题分解出的子问题的解可以**合并**为该问题的解；（最关键的一点）

4) 该问题所分解出的各个子问题是**相互独立**的，即子问题之间**不包含公共的子子问题**。

上述的第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般是随着问题规模的增加而增加；**第二条特征是应用分治法的前提**它也是大多数问题可以满足的，此特征反映了**递归**思想的应用；第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑用贪心法或动态规划法。**第四条特征涉及到分治法的效率**，**如果各子问题是不独立的则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然可用分治法，但一般用动态规划法较好。**

**分治法的基本步骤**

分治法在**每一层递归上**都有三个步骤：

1)分解：将原问题分解为若干个规模较小，相互独立，与原问题形式相同的子问题；

2)解决：若子问题规模较小而容易被解决则直接解，否则递归地解各个子问题

3)合并：将各个子问题的解合并为原问题的解。

**它的一般的算法设计模式如下：**

Divide-and-Conquer(P)

if |P|≤n0 {

   then return(ADHOC(P))

}else{

将P继续分解为较小的子问题 P1 ,P2 ,...,Pk

for i←1 to k

    do yi ← Divide-and-Conquer(Pi) △ 递归解决Pi

T ← MERGE(y1,y2,...,yk) △ 合并子问题

}

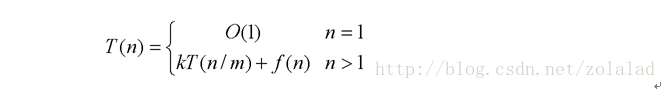
return(T)

其中|P|表示问题P的规模；n0为一阈值，表示当问题P的规模不超过n0时，问题已容易直接解出，不必再继续分解。ADHOC(P)是该分治法中的基本子算法，用于直接解小规模的问题P。因此，当P的规模不超过n0时直接用算法ADHOC(P)求解。算法MERGE(y1,y2,...,yk)是该分治法中的合并子算法，用于将P的子问题P1 ,P2 ,...,Pk的相应的解y1,y2,...,yk合并为P的解。

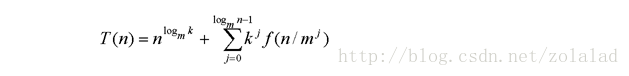
**分治法的应用：1.递归与HANOI塔问题； 2.二分法求方程近似解    3. 用C++实现合并排序   4.求最大值和最小值的分治算法**

**分治法的复杂性分析**

一个分治法将规模为n的问题分成k个规模为n／m的子问题去解。设分解阀值n0=1，且adhoc解规模为1的问题耗费1个单位时间。再设将原问题分解为k个子问题以及用merge将k个子问题的解合并为原问题的解需用f(n)个单位时间。用T(n)表示该分治法解规模为|P|=n的问题所需的计算时间，则有：



通过迭代法求得方程的解：



递归方程及其解只给出n等于m的方幂时T(n)的值，但是如果认为T(n)足够平滑，那么由n等于m的方幂时T(n)的值可以估计T(n)的增长速度。通常假定T(n)是单调上升的，从而当mi≤n<mi+1时，T(mi)≤T(n)<T(mi+1)。

**二、【动态规划法】**

**最优化原理**

1951年美国数学家R．Bellman等人，根据一类**多阶段问题**的特点，把“**多阶段决策**”问题变换为**一系列**互相联系的“**单阶段问题**”，然后**逐个加以解决**。而且**一些静态模型，只要人为地引进“时间”因素，分成时段，就可以转化成多阶段的动态模型，用动态规划方法去处理**。与此同时，他提出了解决这类问题的“**最优化原理**”(Principle of optimality)：“一个过程的最优决策具有这样的性质：即无论其初始状态和初始决策如何，其今后诸策略对以第一个决策所形成的状态作为初始状态的过程而言，必须构成最优策略”。简言之，一个最优策略的子策略，对于它的初态和终态而言也必是最优的。**这个“最优化原理”如果用数学化一点的语言来描述**的话，就是：假设为了解决某一优化问题，需要依次作出n个决策D1，D2，…，Dn，如若这个决策序列是最优的，对于任何一个整数k，1 < k < n，不论前面k个决策是怎样的，以后的最优决策只取决于由前面决策所确定的当前状态，即以后的决策Dk+1，Dk+2，…，Dn也是最优的。

**最优化原理是动态规划的基础**，任何一个问题，如果失去了这个最优化原理的支持，就不可能用动态规划方法计算。能采用动态规划求解的**问题都需要满足一定的条件：**

(1) 问题中的状态必须满足最优化原理；

(2) 问题中的状态必须满足无后效性。

所谓的无后效性是指：“下一时刻的状态只与当前状态有关，而和当前状态之前的状态无关，当前的状态是对以往决策的总结”。

**问题求解模式**

动态规划所处理的问题是一个多阶段决策问题，一般由初始状态开始，通过对中间阶段决策的选择，达到结束状态。这些决策形成了一个决策序列，同时确定了完成整个过程的一条活动路线(通常是求最优的活动路线)。如图所示。动态规划的设计都有着一定的模式，一般要经历以下几个步骤。

初始状态→│决策１│→│决策２│→…→│决策ｎ│→结束状态

图1 动态规划决策过程示意图

**(1)划分阶段：**按照问题的时间或空间特征，把问题分为若干个阶段。在划分阶段时，注意划分后的阶段一定要是有序的或者是可排序的，否则问题就无法求解。

**(2)确定状态和状态变量：**将问题发展到各个阶段时所处于的各种客观情况用不同的状态表示出来。当然，状态的选择要满足无后效性。

**(3)确定决策并写出状态转移方程：**因为决策和状态转移有着天然的联系，状态转移就是根据上一阶段的状态和决策来导出本阶段的状态。所以如果确定了决策，状态转移方程也就可写出。但事实上常常是反过来做，根据相邻两段各状态之间的关系来确定决策。

**(4)寻找边界条件：**给出的状态转移方程是一个递推式，需要一个递推的终止条件或边界条件。

**动态规划法的应用：1.动态规划求0/1背包问题      2.最长公共子串问题的实现** 3. 用动态规划实现导弹拦截      4.最大化投资回报问题的实现

**算法实现**

动态规划的主要难点在于理论上的设计，也就是上面4个步骤的确定，一旦设计完成，实现部分就会非常简单。**使用动态规划求解问题，最重要的就是确定动态规划三要素：问题的阶段,每个阶段的状态以及从前一个阶段转化到后一个阶段之间的递推关系**。递推关系必须是从次小的问题开始到较大的问题之间的转化，从这个角度来说，动态规划往往可以用递归程序来实现，不过因为递推可以充分利用前面保存的子问题的解来减少重复计算，所以对于大规模问题来说，有递归不可比拟的优势，这也是动态规划算法的核心之处。确定了动态规划的这三要素，整个求解过程就可以用一**个最优决策表**来描述，最优决策表是一个二维表，其中行表示决策的阶段，列表示问题状态，表格需要填写的数据一般对应此问题的在某个阶段某个状态下的最优值（如最短路径，最长公共子序列，最大价值等），填表的过程就是根据递推关系，从1行1列开始，以行或者列优先的顺序，依次填写表格，最后根据整个表格的数据通过简单的取舍或者运算求得问题的最优解。下面分别以求解最大化投资回报问题和最长公共子序列问题为例阐述用动态规划算法求解问题的一般思路。

**三、【贪心算法】**

所谓贪心算法是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题他能产生整体最优解或者是整体最优解的近似解。

**贪心算法的基本思路如下：**

1.建立数学模型来描述问题。

2.把求解的问题分成若干个子问题。

3.对每一子问题求解，得到子问题的局部最优解。

4.把子问题的解局部最优解合成原来解问题的一个解。

**实现该算法的过程：**

从问题的某一初始解出发；

while 能朝给定总目标前进一步 do

求出可行解的一个解元素；

由所有解元素组合成问题的一个可行解；

**下面是一个可以使用贪心算法解的题目，贪心解的确不错，可惜不是最优解：**

**例题分析：**

[背包问题]有一个背包，背包容量是M=150。有7个物品，物品可以分割成任意大小。

要求尽可能让装入背包中的物品总价值最大，但不能超过总容量。

物品：  A   B   C   D   E    F   G

重量： 35  30  60  50  40  10  25

价值： 10  40  30  50  35  40  30

**分析：**

目标函数： ∑pi最大

约束条件是装入的物品总重量不超过背包容量：∑wi<=M( M=150)

（1）根据贪心的策略，每次挑选价值最大的物品装入背包，得到的结果是否最优？

（2）每次挑选所占重量最小的物品装入是否能得到最优解？

（3）每次选取单位重量价值最大的物品，成为解本题的策略。

值得注意的是，贪心算法并不是完全不可以使用，**贪心策略一旦经过证明成立后**，它就是一种高效的算法。贪心算法还是很常见的算法之一，这是由于它简单易行，构造贪心策略不是很困难。可惜的是，它需要证明后才能真正运用到题目的算法中。

**一般来说，贪心算法的证明围绕着：整个问题的最优解一定由在贪心策略中存在的子问题的最优解得来的。**

对于例题中的3种贪心策略，都是无法成立（无法被证明）的，解释如下：

    （1）贪心策略：选取价值最大者。反例：

    W=30

    物品：A B C

    重量：28 12 12

    价值：30 20 20

    根据策略，首先选取物品A，接下来就无法再选取了，可是，选取B、C则更好。

    （2）贪心策略：选取重量最小。它的反例与第一种策略的反例差不多。

    （3）贪心策略：选取单位重量价值最大的物品。反例：

    W=30

    物品：A B C

    重量：28 20 10

    价值：28 20 10

根据策略，三种物品单位重量价值一样，程序无法依据现有策略作出判断，如果选择A，则答案错误。

**贪心算法应用：**1.最小生成树之Prim算法  2. 最小生成树之kruskal算法

3. 贪心算法在背包中的应用 4.汽车加油问题之贪心算法

**四、【回溯法】**

回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术为回溯法，而满足回溯条件的某个状态的点称为“回溯点”。

**1、回溯法的一般描述**

可用回溯法求解的问题P，通常要能表达为：对于已知的由n元组（x1，x2，…，xn）组成的一个状态空间E={（x1，x2，…，xn）∣xi∈Si ，i=1，2，…，n}，给定关于n元组中的一个分量的一个约束集D，要求E中满足D的全部约束条件的所有n元组。其中Si是分量xi的定义域，且 |Si| 有限，i=1，2，…，n。我们称E中满足D的全部约束条件的任一n元组为问题P的一个解。

解问题P的最朴素的方法就是枚举法，即对E中的所有n元组逐一地检测其是否满足D的全部约束，若满足，则为问题P的一个解。但显然，其计算量是相当大的。

我们发现，对于许多问题，所给定的约束集D具有完备性，即i元组（x1，x2，…，xi）满足D中仅涉及到x1，x2，…，xi的所有约束意味着j（j<i）元组（x1，x2，…，xj）一定也满足D中仅涉及到x1，x2，…，xj的所有约束，i=1，2，…，n。换句话说，只要存在0≤j≤n-1，使得（x1，x2，…，xj）违反D中仅涉及到x1，x2，…，xj的约束之一，则以（x1，x2，…，xj）为前缀的任何n元组（x1，x2，…，xj，xj+1，…，xn）一定也违反D中仅涉及到x1，x2，…，xi的一个约束，n≥i>j。因此，对于约束集D具有完备性的问题P，一旦检测断定某个j元组（x1，x2，…，xj）违反D中仅涉及x1，x2，…，xj的一个约束，就可以肯定，以（x1，x2，…，xj）为前缀的任何n元组（x1，x2，…，xj，xj+1，…，xn）都不会是问题P的解，因而就不必去搜索它们、检测它们。回溯法正是针对这类问题，利用这类问题的上述性质而提出来的比枚举法效率更高的算法。

回溯法首先将问题P的n元组的状态空间E表示成一棵高为n的带权有序树T，把在E中求问题P的所有解转化为在T中搜索问题P的所有解。树T类似于检索树，它可以这样构造：设Si中的元素可排成xi(1) ，xi(2) ，…，xi(mi-1) ，|Si| =mi，i=1，2，…，n。从根开始，让T的第I层的每一个结点都有mi个儿子。这mi个儿子到它们的双亲的边，按从左到右的次序，分别带权xi+1(1) ，xi+1(2) ，…，xi+1(mi) ，i=0，1，2，…，n-1。照这种构造方式，E中的一个n元组（x1，x2，…，xn）对应于T中的一个叶子结点，T的根到这个叶子结点的路径上依次的n条边的权分别为x1，x2，…，xn，反之亦然。另外，对于任意的0≤i≤n-1，E中n元组（x1，x2，…，xn）的一个前缀I元组（x1，x2，…，xi）对应于T中的一个非叶子结点，T的根到这个非叶子结点的路径上依次的I条边的权分别为x1，x2，…，xi，反之亦然。特别，E中的任意一个n元组的空前缀（），对应于T的根。

因而，在E中寻找问题P的一个解等价于在T中搜索一个叶子结点，要求从T的根到该叶子结点的路径上依次的n条边相应带的n个权x1，x2，…，xn满足约束集D的全部约束。在T中搜索所要求的叶子结点，很自然的一种方式是从根出发，按深度优先的策略逐步深入，即依次搜索满足约束条件的前缀1元组（x1i）、前缀2元组（x1，x2）、…，前缀I元组（x1，x2，…，xi），…，直到i=n为止。

在回溯法中，上述引入的树被称为问题P的状态空间树；树T上任意一个结点被称为问题P的状态结点；树T上的任意一个叶子结点被称为问题P的一个解状态结点；树T上满足约束集D的全部约束的任意一个叶子结点被称为问题P的一个回答状态结点，它对应于问题P的一个解.

**用回溯法解题的一般步骤：**

（1）针对所给问题，定义问题的解空间；

（2）确定易于搜索的解空间结构；

（3）以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

回溯法应用：1.回溯法之数的划分               2.回溯法求解 运动员最佳配对问题

3.回溯法解决汽车加油次数最少问题 4. 用回溯法找出n个自然数中取r个数的全排列

**五、【分支限界法】**

**基本思想 ：**分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树。

在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿子结点被加入活结点表中。 此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止。

**常见的两种分支限界法：**

（1）队列式(FIFO)分支限界法

按照队列先进先出（FIFO）原则选取下一个节点为扩展节点。

（2）优先队列式分支限界法

按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点。

**分支限界法与回溯法的不同:**

（1）求解目标：回溯法的求解目标是找出解空间树中满足约束条件的所有解，而分支限界法的求解目标则是找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解。

（2）搜索方式的不同：回溯法以深度优先的方式搜索解空间树，而分支限界法则以广度优先或以最小耗费优先的方式搜索解空间树。

**解空间树的动态搜索**

（1）回溯求解0/1背包问题，虽剪枝减少了搜索空间，但整个搜索按深度优先机械进行，是盲目搜索（不可预测本结点以下的结点进行的如何）。

（2）回溯求解TSP也是盲目的（虽有目标函数，也只有找到一个可行解后才有意义）

（3）分支限界法首先确定一个合理的限界函数，并根据限界函数确定目标函数的界[down, up]；然后按照广度优先策略遍历问题的解空间树，在某一分支上，依次搜索该结点的所有孩子结点，分别估算这些孩子结点的目标函数的可能取值（对最小化问题，估算结点的down，对最大化问题，估算结点的up）。如果某孩子结点的目标函数值超出目标函数的界，则将其丢弃（从此结点生成的解不会比目前已得的更好），否则入待处理表。

**分支限界法的设计思路**

设求解最大化问题，解向量为X=(x1,…,xn)，xi的取值范围为Si，|Si|=ri。在使用分支限界搜索问题的解空间树时，先根据限界函数估算目标函数的界[down, up]，然后从根结点出发，扩展根结点的r1个孩子结点，从而构成分量x1的r1种可能的取值方式。

对这r1个孩子结点分别估算可能的目标函数bound(x1)，其含义：以该结点为根的子树所有可能的取值不大于bound(x1)，即：

bound(x1)≥bound(x1,x2)≥…≥ bound(x1,…,xn)

若某孩子结点的目标函数值超出目标函数的下界，则将该孩子结点丢弃；否则，将该孩子结点保存在待处理结点表PT中。

再取PT表中目标函数极大值结点作为扩展的根结点，重复上述。

直到一个叶子结点时的可行解X=(x1,…,xn)，及目标函数值bound(x1,…,xn)。

**分支限界法应用：1.**分支限界法之装载问题       2. 分支限界法之布线问题

3. 分支限界法之0 1背包问题 4. 分支限界法之旅行售货员问题