分治算法

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。

可使用分治法求解的一些经典问题

1. 二分搜索
2. 大整数乘法
3. Strassen矩阵乘法
4. 棋盘覆盖
5. 合并排序
6. 快速排序
7. 线性时间选择
8. 最接近点对问题
9. 循环赛日程表
10. 汉诺塔

最大子数组问题

分治法：

[low,high]

I,j最大子数组的开始索引和结束索引

[low,mid] [mid + 1,high]

1. I,j同时位于低区间
2. I,j同时位于高区间
3. i位于低区间 j位于高区间

因为i和j是由mid分割的，所以分别取得在low mid里面的i值

树（数据结构的一种）

什么是树？

树是n个结点的有限集。n = 0时称为空树。在任意一颗非空树中：

1. 有且仅有一个特定的称为根（root）的结点；
2. 当n>1时，其余结点可分为m(m>0)个互不相交的有限T1、T2、Tm,其中每一个集合本身又是一棵树，并且称为根的子树（SubTree）,如图6-2-1所示。
3. 空树
4. 只有一个根节点的树

存储结构一般是 顺序存储和链式存储。

顺序存储的话其实就是使用数组了。

树的关系复杂 使用链式存储

链式存储呢，就是定义对象，对象间相互引用，互相访问。

如果用数组的话呢，这些节点的关系没法表示出来，比如说谁是父亲，是谁的父亲，谁是孩子，是谁的孩子。

链式存储有几种表示法

1. 双亲表示法 data parent 方便找到父亲，缺点是不太容易访问到兄弟结点，因为没保存兄弟结点。如果要访问兄弟结点的话，首先要找到父节点，然后遍历所有的结点，然后判断谁的父节点是它，然后这样的话才能得到它的兄弟。这样的话呢，就很麻烦，因为不能直接访问到。

如果我们想要找到当前结点的父亲的子节点也很难找到

1. 孩子表示法 data child1 child2 child3 优点：很方便的访问到它的所有孩子。

缺点：

很难得到双亲。

想访问兄弟的话，也很难访问到。

浪费空间

1. 孩子兄弟表示法 data firstchild rightcsib

优点：

1. 很方便的访问到所有的孩子
2. 很方便的访问到所有的兄弟

缺点：很难得到父亲。

二叉树

二叉树（Binary Tree）是n（n>=0）个结点的有限集合，该集合或者为空集（称为空二叉树），或者由一个根结点和两颗互不相交的、分别称为根结点的左子树和右子树的二叉树组成。

1. 空二叉树
2. 只有根结点
3. 大于两个结点

二叉树是在树的基础上做了一个限制，就是每个结点只能最多有两个结点，而且这两个结点是有左右顺序的。

满二叉树：

除了最后一层，其他层任意每个结点都有两个子结点。

完全二叉树：对一颗具有n个结点的二叉树按层序编号，如果编号为i(1<=i<=n)的结点与同样深度的满二叉树中编号为i的结点在二叉树中位置完全相同，则这颗二叉树称为完全二叉树。

在满二叉树的基础上，

满二叉树一定是完全二叉树，完全二叉树不一定是满二叉树。

完全二叉树，是在最后一层可以缺失的，从右往左的，并且是连续的。

二叉树的存储结构

一般的树来说是一对多的关系，使用顺序结构存储起来比较困难，但是二叉树是一种特殊的树，每个结点最多有两个子节点，并且子节点有左右之分，并且兄弟，父亲，孩子很可以很方便的通过编号得到，所以我们使用顺序存储结构使用二叉树的存储。

二叉树每个结点最多有两个孩子，所以为它设计一个数据域和两个指针域，我们称这样的链表为二叉链表。

缺点：

1. 占用内存比较高。
2. 对于一个完全树来说，链式存储的话，每个节点要占用三个格子，顺序存储的话只占用一个格子，相当于一个三倍的内存的差别，浪费内存。
3. 没有办法访问到父节点。
4. 方法：每个节点再添加一个字段，parent，解决问题。

二叉树的遍历

二叉树的遍历是指从根节点出发，按照某种次序一次访问二叉树中的所有结点，使得每个结点被访问一次且仅被访问一次。

1. 前序遍历

先输出当前结点的数据，再依次遍历输出左结点和右结点。

1. 中序遍历

先遍历输出左结点，再输出当前结点的数据，再遍历输出右结点。

1. 后序遍历

先遍历输出左结点，再遍历输出右结点，最后输出当前结点的数据。

1. 层序遍历
2. 从树的第一层开始，从上到下逐层遍历，在同一层中，从左到右对结点 逐个访问输出

因为我们不知道它存什么，所以用泛型。

二叉排序树，又称为二叉查找树。它或者是一颗空树，或者是具有下列性质的二叉树。

若它的左子树不为空，则左子树上所有的结点的值均小于根节点的值。

若它的右子树不为空，则右子树上所有结点的值均大于它的根节点的值。

它的左右子树也分别为二叉排序树。

1. 排序方便
2. 方便查找
3. 方便插入和删除

二叉排序树的存储

因为二叉排序树的存储，跟自身值的大小有关系，并不是像之前学习的完全二叉树使用顺序结构可以存储的，所以我们使用链式结构存储二叉排序树。

二叉排序树的删除操作

1. 叶子节点
2. 仅有左子树和右子树的节点
3. 左右子树都有

堆

堆是具有下列性质的完全二叉树：

每个结点的值都大于或等于左右孩子的结点的值，称为大顶堆；

或者每个结点的值都小于或等于其左右孩子结点的值，称为小顶堆！

堆排序

堆排序算法就是利用堆（小顶堆或者大顶堆）进行排序的方法。

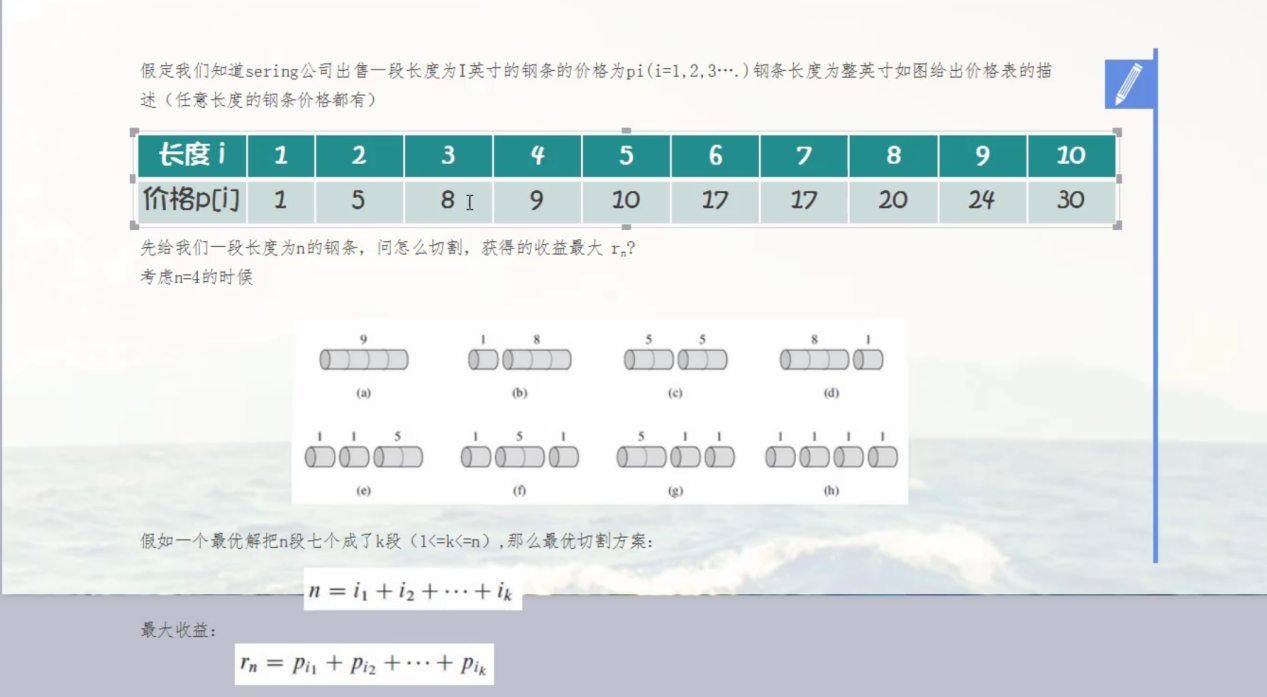
将待排序的序列构造成一个大顶堆，此时整个序列的最大值就是根节点。将他移走（跟堆的最后一个元素交换，此时末尾元素就是最大值），然后将剩余的n-1个序列重新构造成一个堆，这样就会得到n个元素中的次小值。如此反复执行，便能得到一个有序序列了。

动态规划

什么是动态规划，我们要如何描述它？

动态规划算法通常基于一个递推公式及一个或多个初始状态。当前子问题的解将由上一次子问题的解推出。

动态规划和分治法相似，都是通过组合子问题的解来求解原问题。分治法将问题划分成互不相交的子问题，递归求解子问题，再将他们的解组合起来，求出原问题的解。与之相反，动态规划应用于子问题重叠的情况，即不同的子问题具有公共的子问题。在这种情况下，分治算法会做出许多不必要的工作，它会反复的求解那些公共子子问题。而动态规划算法对每个子子问题只求解一次，将结果保存到表格中，从而每次求解一个子子问题都要重新计算。



钢条求解方案

我们从钢条的左边切下长度为i的一段，只对右边剩下长度为n-i的一段继续进行切割。这样，不做任何切割的方案就是：当第一段长度为n的时候，收益为Pn，剩余长度为i,收益为Pi:

Rn = max(Pi + Rn-i); (1<= i <=n)

使用价格表，价格表有一个条件：价格表的长度不能小于n，就是说n有多少个，价格表长度就得有多少个，因为n要使用下面的价格表，我们长度为n,那么我们长度为n的价格表必须有，那小于n的这个价格表也必须有。

递归方法的缺点：影响性能，效率比较低。

递归法之所以效率低，是因为它反复求解相同的子问题。因此，动态规划算法安排求解的顺序，对每个子问题只求解一次，并将结果保存下来。如果随后再次需要此子问题的解，只需查找保存的结果，不必重新计算。因此动态规划的方法是付出额外的内存空间来节省计算时间。

动态规划有两种等价的实现方法（我们使用上面的钢条切割问题为例，实现这两种方法）

第一种方法是 带备忘的自顶向下法

此方法依然是按照自然的递归形式编写过程，但过程中会保存每个子问题的解（通常保存在一个数组中）。当需要计算一个子问题的解时，过程首先检查是否已经保存过此解。如果是，则直接返回保存的值，从而节省了计算时；如果没有保存过此解，按照正常方式计算这个子问题。我们称这个递归过程是带备忘的。

第二种方法是 自底向上法

首先恰当的定义子问题的规模，是的任何问题的求解都只依赖于更小的子问题的解。因而我们将子问题按照规模排序，按从小到大的顺序求解。当求解某个问题的时候，它所依赖的更小的子问题都已经求解完毕，结果已经保存。

动态规划 0-1背包问题

问题描述：

假设现有容量mkg的背包，另外有i个物品，重量分别为w[1]、w[2]...w[i](kg)，价值分别为p[1],p[2]...p[i]（元），将哪些物品放入背包可以使背包的总价值最大？最大价值是多少？

1. 穷举法（把所有情况列出来，比较得到 总价值最大的情况）
2. 动态规划算法

我们要求得i个物体放入容量为m(kg)的背包的最大价值（计为c[i,m]）。在选择物品的时候，对于每种物品i只有两种选择，即装入背包或不装入背包。某种物品不能装入多次（可以认为每种物品只有一个），因此该问题被称为0-1背包问题

对于c[i,m]有下面几种情况：

1. c[i,0] = c[0,m] = 0
2. c[i,m] = c[i-1,m] w[i] > m （最后一个物品的重量大于容量，直接舍弃不用）

W[i] <= m的时候有两种情况，一种是放入i，一种是不放入i

不放入i c[i,m] = c[i-1,m]

放入i c[i,m] = c[i-1,m - w[i]] + p[i]

c[i,m] = max(不放入i,放入i)