**笛卡尔树（Cartesian Tree）**

[上一篇日志](http://www.hhanger.com/blog/?p=118)提到了，[COCI](http://www.hsin.hr/coci/)某题vls向我介绍了[笛卡尔树](http://en.wikipedia.org/wiki/Cartesian_tree" \o "Cartesian Tree" \t "_blank)这个东东，今天上班无聊的时候就在网上搜了一下，发现介绍的东西有不少，dd牛也N早前就写过这个，于是发现自己太土了。由于介绍的文章是在太多，不知道的同学可以随便搜搜，我在这里写写纯粹是为了加深自己的影响，毕竟这东西估计用到的次数不会太多。

笛卡尔树是一棵二叉树，树的每个节点有两个值，一个为key，一个为value。光看key的话，笛卡尔树是一棵二叉搜索树，每个节点的左子树的key都比它小，右子树都比它大；光看value的话，笛卡尔树有点类似堆，根节点的value是最小（或者最大）的，每个节点的value都比它的子树要大。

网上的介绍往往把笛卡尔树和[Treap](http://en.wikipedia.org/wiki/Treap" \o "Treap" \t "_blank)拿来类比，因为它们的结构非常类似。（因为我其实没学过Treap，似乎只微微看过几眼，所以下面的言论有误，不过大致的意思应该没错）事实上，两者的结果不是类似，而是一模一样，但是两者的用途和用法不一样。笛卡尔树是把已有的一些（key, value）二元组拿来构造树，然后利用构树过程和构好的树来解决问题。而Treap的目的只是对一些key进行二叉搜索，但是为了保证树的平衡性，为每个key随机地额外增加了一个value（或者叫权重）属性，这样从概率上来讲可以让这棵树更加平衡。理解的两者的关系和区别后，什么时候该用什么结构就一目了然了。

笛卡尔树比较优美，也是关键的地方就是它的构树过程。整个过程的第一步是把所有点按照key排序，然后从一个节点开始，按key递增顺序依次插入节点。想象一下，假设已经有一棵笛卡尔树，那么现在我们要插入一个新的节点，而这个节点比这棵树所有节点的key都大，那么应该如何插入呢？假设这个节点已经被插入，那么它的位置肯定是在从根节点开始一直向右走到第的位置。所以，每次插入新节点的时候，一定插入到最右侧那条路中的某个位置，而原来位置的节点变成了这个新节点的左子树，新插入的点变成最右侧那条路的最后一个节点。那么如何确定插入的位置呢？那就要根据这个节点的value值了，因为满足堆的性质，所以一条路从上到下，其value值肯定是递减的。就是因为这个递减的性质，我们可以把最右侧的那条路用一个栈表示，栈底是根，栈顶是最新节点，从底到顶，value值和key值都递增。每次新插入一个节点的时候，就从顶往底一个个看，找到第一个value大于新节点value的节点，作为新节点的父亲即可。因为每个节点最多进栈一次，出栈一次，所以整个构树过程是O(N)的。仔细回想一下构树的过程，其实不难想，但是实在是挺巧妙的，佩服yy出来的人！

最后提两道相关的题目：第一个是[SGU 155](http://acm.sgu.ru/problem.php?contest=0&problem=155)，名字就叫Cartesian Tree，是裸的笛卡尔树

第二题是[ZOJ 2243](http://acm.zju.edu.cn/onlinejudge/showProblem.do?problemId=1243)，似乎也是裸的，悲剧-\_-有兴趣的可以去试试。。。