[**BST拓展与伸展树(splay)一日通**](http://www.artofproblemsolving.com/Forum/blog.php?u=30934&b=13018&)

[Permanent Link](http://www.artofproblemsolving.com/Forum/blog.php?b=13018&)by [**zkw**](http://www.artofproblemsolving.com/Forum/memberlist.php?mode=viewprofile&u=30934&), May 17, 2009, 6:44 am

　　BST 是 OI 中一个常用的数据结构，主要支持的操作是动态维护一个有序表，从而支持字典，前驱，后继，中序遍历，优先队列等等多种操作。如果在域中维护了子树的 size，还可以支持查找第 k 大的数，询问名次等等附加功能。伸展树（Sleator and Tarjan, 1985）是 BST 的一个变种，可以自调整以维护平衡，并支持根据需要随时把任意节点旋转到根（称为 splay 操作），从而很好的支持了分裂合并等操作，从某种意义上（详见下文）也简化了思维和编程的复杂度。由于 splay 操作的迅速而优美，伸展树不仅可以维护有序表，还可以放弃关键字的有序性，像块状链表一样维护一个一般的序列（这是一般的 BST 甚至其他平衡二叉树难于做到的），支持块状链表的大多数方法，并拥有更低的理论复杂度和实际代码量。另外，splay 是唯一支持稳定排序的平衡二叉树，可以妥善处理关键字相同的元素（按照插入顺序有序），在输出时可以选择返回其中最近插入、最远插入的，或是直接返回整个区间。  
  
　　1. 伸展树的基本操作与应用（参见国家集训队杨思雨前辈的论文或 *The Magical Splay* by sqybi）  
　　这里有几个需要注意的地方：  
　　(1) 自底向上的实现中可以很轻易的维护 size 等附加信息，操作实现也更像传统的 BST，只需在每次操作完成后额外执行一次 splay 操作，将对应的节点旋转到根即可。  
　　(2) 只用 zig 和 zag 的确可以实现伸展树的所有操作，但这时 splay 会丧失自调整平衡的特性。（基本退化成 BST）但是可以证明 zig+zag=zigzag，所以只需要 zig, zag, zigzig, zagzag 就可以实现一个实用的伸展树了。考虑对称性就是两种操作：zig 和 zigzig，称之为单旋和双旋。  
  
　　2. 自顶向下伸展树  
　　不再拘泥于传统 BST 的操作方式（逐层向下查找），**任何时刻正在访问或操作的节点都是整棵树的根**。显然查找过程和 splay 过程已经合而为一了。  
　　为了实现新的查找过程，我们维护两棵临时树，分别代表未来树根的左右子树，初始为空。这里左边的临时树又可以看成是一些没有右儿子的子树串成的链表，右边亦然。每次前进时比较两层关键字，前进方向一致（都是向左或都是向右）执行双旋，否则执行单旋。  
　　单旋操作（以向左走为例）：左儿子变成新的根，原来的根没有了左儿子，正好挂到右临时树下面。  
　　双旋操作（以向左走为例）：左孙子变成新的根，左儿子挂到右临时树下面，把它的右儿子送给根当左儿子，拿根来当它的右儿子。  
　　当查找完成的时候，把左儿子挂到左临时树下面，右儿子挂到右临时树下面，拿临时树做新的儿子。  
　　简约高效的插入、删除：先查找键值旋到根，然后……在根旁边插入元素……删除根……不用我说了吧，很简单（还不懂就看 code）  
　　优点：简单（不存父亲指针），速度快（常数小一半以上），优美（整个结构中本质只有一种操作：splay，别的操作只是衍生产品），非递归（栈溢？不怕不怕啦）。  
  
　　3. Code Trick  
　　(1) 注意到 size 域本质上是需要自底向上的，自顶向下的实现中难以维护。如果像自底向上一样，维护父亲域，然后额外维护，是可行的，但是太麻烦了。难道必须放弃了吗？不。既然临时树的本质是一个链表，查找过程中我们可以反过来存，即……左临时树里每个节点的右儿子域其实指向的是它的父亲。查找完成后，我们可以很容易的自底向上算size，顺便把这个链表反过来。:) 还有一个好处是每次临时树里挂子树时都变成在链表头插入节点了，比链表尾插入要好写多多（表尾不存了，空表都不用特判了）  
　　(2) 指向不存在的节点的指针可以设为 NULL ，但是统计 NULL 的 size 要出错，难道要特判？不，新建一个普通节点当 null，size=0，都指向它就可以了。  
　　(3) 既然左右是对称的，那两个儿子声明成 child[2]，旋转时传入一个方向 bool，就可以少写一半过程。  
　　(4) null 的儿子闲着也是闲着，既然两棵临时树还没地方存，就当临时树玩好了。这样一来好像链表反向也好写了……（为什么？我也不知道，自己看 code，如果你发现不这么玩能更好写就告诉我）  
　　(5) 稳定排序？在查找时不判断是否相等。也就是只要要查的只要不大于当前节点都向左转。比如序列“(1)4 (2)4 (3)4 (4)5 (5)5”中查找 5，这样可能会查到(3)，也可能查到(4)。原则是只要查到的数小于要查的数，就再查一次后继。查后继是均摊 [O\left( 1 \right)](http://www.artofproblemsolving.com/Forum/code.php?hash=ae233bb28eccf8c3300870412e966e63bb9feff6&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)的！而且这样做每步查找都节省一个判断，在保持稳定性的同时，速度也会更快。  
　　(6) 空树不好初始化？把树初始化成有一个节点[+ \infty](http://www.artofproblemsolving.com/Forum/code.php?hash=bdef0543744e33d999c617d7221590908f16e2c4&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)。好处是，每一个节点现在都有后继了，这样删除的时候也方便了。  
  
　　4. 复杂度  
　　完整的复杂度分析参见任何一本均摊分析的教科书（伸展树的分析和并查集的分析是两个均摊分析的经典之作），这里只给出一些结论。  
　　(1) 插入，删除，查找  
　　数据结构 时间复杂度  
　　BST 平均 [O\left(\log n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=7bb9500ad834ba2fb2e3529278a9c579da771bb1&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)  
　　Treap 期望 [O\left(\log n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=7bb9500ad834ba2fb2e3529278a9c579da771bb1&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)  
　　Splay 均摊 [O\left(\log n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=7bb9500ad834ba2fb2e3529278a9c579da771bb1&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)  
　　AVL 严格 [O\left(\log n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=7bb9500ad834ba2fb2e3529278a9c579da771bb1&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)  
　　(2) 在伸展树中，连续访问一段长度为 [m](http://www.artofproblemsolving.com/Forum/code.php?hash=6b0d31c0d563223024da45691584643ac78c96e8&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)的区间的均摊代价是 [O\left(m + \log n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=c94d336ac622155a56717046a0786602f620149c&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)而不仅仅是 [O\left(m\log n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=e9d34f16e68b83d6da6baf0d5fc4055beb36b328&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)。这个结论蕴含前驱，后继的查找都是均摊 [O(1)](http://www.artofproblemsolving.com/Forum/code.php?hash=6c69dc5bbeb314ef78589c0047ce04c8e5006432&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)。  
　　(3) 如果存在非随机访问（有规律的，比如按顺序插入），伸展树的形态可能相当不平衡（极端情况下，一条链）。但是一旦沿着这条链花费 [O\left( n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=284ba20453fc623a2de1fb045230d5b2a40b81e4&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)时间走过一次，树的结构又会变得平衡许多。而这次 [O\left( n\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=284ba20453fc623a2de1fb045230d5b2a40b81e4&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)的代价，可以通过前 [n](http://www.artofproblemsolving.com/Forum/code.php?hash=d1854cae891ec7b29161ccaf79a24b00c274bdaa&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)次不必旋转节省的开销来弥补。有趣的是，如果存在非随机访问，伸展树要比上面提到的所有平衡树都要快。例如，如果用 splay 维护一个双端队列（只在两端增删节点），上面的所有操作都是均摊 [O\left(1\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=6c66d235da64d190f346bfbba05038bf1c22bbb6&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)的。  
　　(4) 由于均摊复杂度的局限性，伸展树在生活中的应用受到限制。想像下面的场景：  
　　-我已经等了整整五十九分钟了！你们的服务怎么这么慢！  
　　-我们的服务均摊复杂度是每人一分钟，也就是六十个人一个小时。一个小时之前，在你前面排队的五十九个人在一分钟之内全部得到了服务。  
　　-……  
　　但是这并不影响 OI 中的程序总耗时。:wink:  
  
　　5. 应用  
　　维护无关键字序列：  
　　NOI2003 文本编辑器 [O\left(I + O + t\log I\right)](http://www.artofproblemsolving.com/Forum/code.php?hash=c1660adf4c7d8d9091a19497d85824958cc3b19a&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)[I](http://www.artofproblemsolving.com/Forum/code.php?hash=ca73ab65568cd125c2d27a22bbd9e863c10b675d&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)为输入时间，[O](http://www.artofproblemsolving.com/Forum/code.php?hash=08a914cde05039694ef0194d9ee79ff9a79dde33&sid=9b4eb9a082a51fd46c11f492b2d0d2b4) 为输出时间  
　　NOI2005 维护数列 [O\left(P + M\logQ)](http://www.artofproblemsolving.com/Forum/code.php?hash=b8377d0e39d6200252068f56f29e532fb656c28e&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)[P](http://www.artofproblemsolving.com/Forum/code.php?hash=511993d3c99719e38a6779073019dacd7178ddb9&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)为初始和插入总数，[Q](http://www.artofproblemsolving.com/Forum/code.php?hash=c3156e00d3c2588c639e0d3cf6821258b05761c7&sid=9b4eb9a082a51fd46c11f492b2d0d2b4) 为最大长度  
　　维护稳定有序序列：  
　　NOI2006 生日快乐 [O\left(N \log N)](http://www.artofproblemsolving.com/Forum/code.php?hash=92815e1a3083331f12c187ebd0e882ed4d7ee781&sid=9b4eb9a082a51fd46c11f492b2d0d2b4)  
　　还有更难的……动态树。  
  
　　6. code  
　　程序代码

**Code:**

#include <cstdio>   
using namespace std;   
const int maxint=~0U>>1;   
  
struct node   
{   
   int key,size;   
   node \*c[2];   
   node():key(0),size(0){c[0]=c[1]=this;}   
   node(int key\_,node\* c0\_,node\* c1\_):key(key\_){c[0]=c0\_;c[1]=c1\_;}   
   node\* rz(){return size=c[0]->size+c[1]->size+1,this;}   
} Tnull,\*null=&Tnull;   
  
struct splay   
{   
   node \*root;   
   splay()   
   {   
      root=(new node(\*null))->rz();   
      root->key=maxint;   
   }   
   void zig(bool d)   
   {   
      node \*t=root->c[d];   
      root->c[d]=null->c[d];   
      null->c[d]=root;   
      root=t;   
   }   
   void zigzig(bool d)   
   {   
      node \*t=root->c[d]->c[d];   
      root->c[d]->c[d]=null->c[d];   
      null->c[d]=root->c[d];   
      root->c[d]=null->c[d]->c[!d];   
      null->c[d]->c[!d]=root->rz();   
      root=t;   
   }   
   void finish(bool d)   
   {   
      node \*t=null->c[d],\*p=root->c[!d];   
      while(t!=null)   
      {   
         t=null->c[d]->c[d];   
         null->c[d]->c[d]=p;   
         p=null->c[d]->rz();   
         null->c[d]=t;   
      }   
      root->c[!d]=p;   
   }   
   void select(int k)   
   {   
      int t;   
      for(;;)   
      {   
         bool d=k>(t=root->c[0]->size);   
         if(k==t||root->c[d]==null)break;   
         if(d)k-=t+1;   
         bool dd=k>(t=root->c[d]->c[0]->size);   
         if(k==t||root->c[d]->c[dd]==null){zig(d);break;}   
         if(dd)k-=t+1;   
         d!=dd?zig(d),zig(dd):zigzig(d);   
      }   
      finish(0),finish(1);   
      root->rz();   
   }   
   void search(int x)   
   {   
      for(;;)   
      {   
         bool d=x>root->key;   
         if(root->c[d]==null)break;   
         bool dd=x>root->c[d]->key;   
         if(root->c[d]->c[dd]==null){zig(d);break;}   
         d!=dd?zig(d),zig(dd):zigzig(d);   
      }   
      finish(0),finish(1);   
      root->rz();   
      if(x>root->key)select(root->c[0]->size+1);   
   }   
   void ins(int x)   
   {   
      search(x);   
      node \*oldroot=root;   
      root=new node(x,oldroot->c[0],oldroot);   
      oldroot->c[0]=null;   
      oldroot->rz();   
      root->rz();  
   }   
   void del(int x)   
   {   
      search(x);   
      node \*oldroot=root;   
      root=root->c[1];   
      select(0);   
      root->c[0]=oldroot->c[0];   
      root->rz();   
      delete oldroot;   
   }   
   int sel(int k){return select(k-1),root->key;}   
   int ran(int x){return search(x),root->c[0]->size+1;}   
} sp;   
  
int main()   
{   
   for(;;)   
   {   
      char cmd;   
      int num;   
      scanf(" %c%d",&cmd,&num);   
      switch(cmd)   
      {   
         case'i':sp.ins(num);break;   
         case'd':sp.del(num);break;   
         case's':printf("%d\n",sp.sel(num));break;   
         case'r':printf("%d\n",sp.ran(num));break;   
      }   
   }   
}

　　原来的程序里 ins() 会算错 root 的 size，所幸 root 的 size 并不参与实际操作:oops:，现已改正。

　　非常好玩的程序，zig, zigzig, finish 三个过程中的语句都是循环顶针……跑得也挺快，至少比 treap 要快。 :P  
  
　　p.s. 退役之后在电脑边的时间锐减，也许一段时间之内不会有这么长的东西发了。