# 二叉搜索树(BST)

二叉搜索树是一种二叉树的树形数据结构。

树上的每个节点都有一个权值，对于任意节点，若它的左儿子权值小于它，右儿子的权值大于它，这就是一棵二叉搜索树。

容易发现维护一些信息后可以轻易地查询第 $k$ 大，查询排名和前驱后继等信息，复杂度和深度有关系。

相关操作：

约定：$val[x]$ 为 $x$ 点的权值，$cnt[x]$ 为 $x$ 点权值出现次数，$siz[x]$ 为子树 $cnt$ 的和

- 找最大/最小值：一直跳左/右儿子即可。
- 查找元素 $v$ ： 在树上二分
- 查找前驱（第一个比元素 $v$ 小的数）/后继（第一个比元素 $v$ 小的数）： 树上二分
- 插入元素：树上二分，如果当前点的权值等于 $v$ 就 $cnt++$，如果大于就向当前点左儿子跳，跳到空节点就新建一个节点。
- 删除元素：先二分找到这个元素，$cnt--$ ，若 $cnt$ 变为0：
  - 是叶子节点直接删除
  - 有一个儿子，直接返回儿子
  - 有两个儿子，将左子树最大值或右儿子最小值移到当前点位置
- 找第 $k$ 大：树上二分

遍历顺序：

- 先序遍历：先根后左子树再右子树
- 中序遍历：先左后根再右
- 后续遍历：先左后右再根

以中序遍历为例，“先左”的意思是一直走左儿子直到叶子节点，此时所在的节点是遍历的第一个节点。

显然一条链也可以是二叉搜索树，此时操作复杂度退化为 $O(n)$ ，我们要避免这种情况发生。

# AVL树

平衡树就是在二叉树基础上维护树是平衡的。

我们称一棵树是完全平衡的当且仅当任何一个节点左右儿子深度差都不超过 1。

此时整棵树的深度是 $logn$ 级别的，操作复杂度有保证。

如果一棵树已经平衡了，最多会使得某个节点左右子树的深度差增加 1 ，并且是修改位置到根的路径上的所有点都有可能不平衡。

树不平衡时，可以通过旋转操作来使之平衡。

考虑一个旋转操作，如下图：

```
  A            B
 / \          / \
1   B   ->   A   3
   / \       /\  /\
  2   3     1  2 C D
     / \
    C   D
```

上述的左旋操作（将 $B$ 旋转到 $A$ 的位置并保证二叉搜索树的性质）：

- 先将 $A$，$B$ 间的边断开
- 将 $B$ 的左子树接在 A 的右子树上。
- 将 $A$ 接在 $B$ 的左子树上

“2”子树原来在A的右子树里，所以将它接到 $A$ 的右子树上仍然满足二叉搜索树的性质。

"A"与“B”交换的正确性同理。

当发现某个子树不平衡时，转一下就好啦！

*欸，上述旋转太复杂，记不住也不想推怎办？*

*考虑暴力一点，不平衡时重构一下树*

*只要不平衡就重构未免也太离谱了*

考虑每隔插入 $\sqrt{n}$ 个点就重构一下整棵树，复杂度为 $n\sqrt{n}logn$，时间复杂度允许时可以写这个。

这个东西就叫替罪羊树。

# Treap

市面上最流行的平衡树结构。

为啥叫 $treap$ 呢？因为它是堆（Heap）和二叉树（Tree）的结合体。

朴素的二叉搜索树，可以按权值从小到大插入，会一直插到右儿子上，最终退化成一条链。

$Treap$ 通过每个节点一个随机的 $priority$ 值，然后在满足BST的条件下，使树的 $priority$ 值满足小根堆的性质（由于是随机值，可以证明树高为 $O(logn)$~~但是咱不会证~~）

按照下图的旋转，单旋只交换根节点和儿子的 $priority$ 关系，所以可以用单旋来使 $BST$ 满足小根堆性质

```
  A            B
 / \          / \
1   B   ->   A   3
   / \      /\  /\
  2   3    1  2 C D
     / \
    C   D
```

部分操作会改变树的形态。

- 插入操作：按二叉搜索树的方式插入，再满足小根堆的性质，不断向上旋转即可。
- 删除操作：将要删节点的 $priority$ 设为无限大，从左右儿子中选取 $priority$ 较小的转到根节点。此时根节点的深度增加一。重复上述操作直到要删节点为叶子，直接删除。

设置的 $priority$ 通过随机对树旋转保证复杂度。~~不会证明QAQ，据说是树高期望log的~~

其余的操作与 $BST$ 操作一样。

时间复杂度 $O(nlogn)$，空间复杂度 $O(n)$

因为有随机的 $priority$，复杂度其实看脸（），百分之九十九的情况是对的。

被卡了说明随机数不够优秀（bushi）

# 无旋Treap

随机旋转？这玩意根本记不住啊！写起来也很麻烦QAQ

无旋treap应运而生，又叫范浩强treap（膜拜）

无旋treap的核心操作是树的分裂与合并。

先来看以树的大小划分的分裂（分出前 $val$ 个数）：

比如下树我们要划分前三个数出来到树 $l$ 中，其他数到 $r$ 中。

```
  A
 / \
1   B
   / \
  2   3
     / \
    C   D
```

走到 $A$ 时发现左子树大小加当前节点一共有两个节点，就把节点 $A$ 和它的左子树划分到树 $l$ 中，并向右子树 $B$ 递归。

发现此时左子树大小加当前节点一共有两个节点，加上先前的两个超过了四，于是将节点 $B$ 和它的右子树划分到 树 $r$ 中，并向左子树 "2" 递归。

发现“2”应划分到 树 $l$ 中，直接接到上一次划分到 树 $l$ 中的右儿子就行。因为当把一个点划分到 树 $l$ 中时，会向这个点的右子树拿来递归，这个点在 树$l$ 中的右子树是空的。且节点“2”及其对应的子树是在树 $l$ 上一次加入的节点的右子树，这样做也满足二叉搜索树的性质。

对应到具体代码上：

```
inline void split(int nown,int &l,int &r,int val){
    if(!nown){
        l=r=0;//分到底了
        return ;
    }
    if(size[ls[nown]]+1<=val){
        l=nown;
        split(rs[nown],rs[nown],r,val-size[ls[nown]]-1);
    }//当前节点的size小于等于分裂值，分到左边
    else{
        r=nown;
        split(ls[nown],l,ls[nown],val);
    }//同上，分到右边
    update(nown);
    return ;
}
```

**注意这么一个回传值的写法**

再看合并（合并时的值域不能有交）：

其中树 $l$ 中的所有节点比树 $r$ 中的所有数大。

```
inline void merge(int &k,int l,int r){
    if(l==0||r==0){
        k=l+r;//到底，选择其中不为0的节点回传
        return ;
    }
    if(num[l]<num[r]){
        k=l;
        merge(rs[l],rs[l],r);
    }//num即满足节点小根堆性质的rand()值
    else{
        k=r;
        merge(ls[r],l,ls[r]);
    }
    update(k);
    return ;
}
```

在某一个位置加入或删除节点即按照其节点位置前后分裂，加入或删除这个节点后合并。

其他操作都和普通treap大同小异。

时间复杂度 $O(nlogn)$，空间复杂度 $O(n)$

因为有随机的 $priority$，复杂度其实看脸（），百分之九十九的情况是对的。

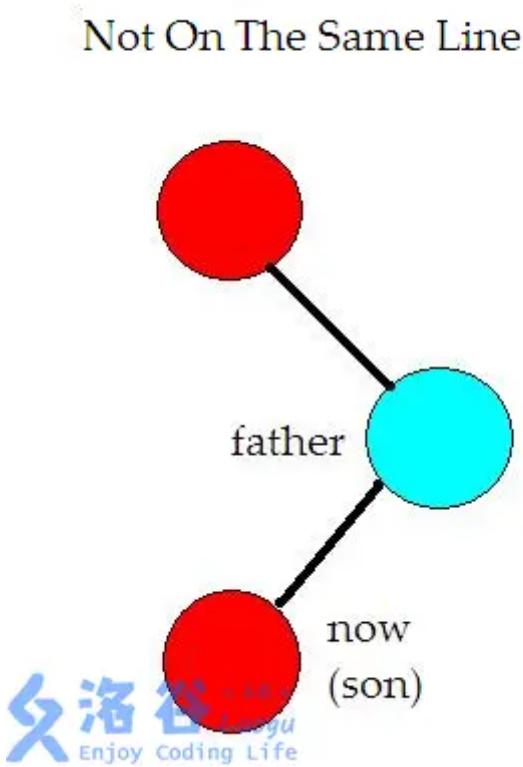
# Splay

Splay相比普通BST多了两个东西：

- 每进行一个节点 $x$ 的操作后都要将其转到根节点
- 这个转到根节点的过程有特殊的策略

$now$ 是我们现在要往上旋转的节点。

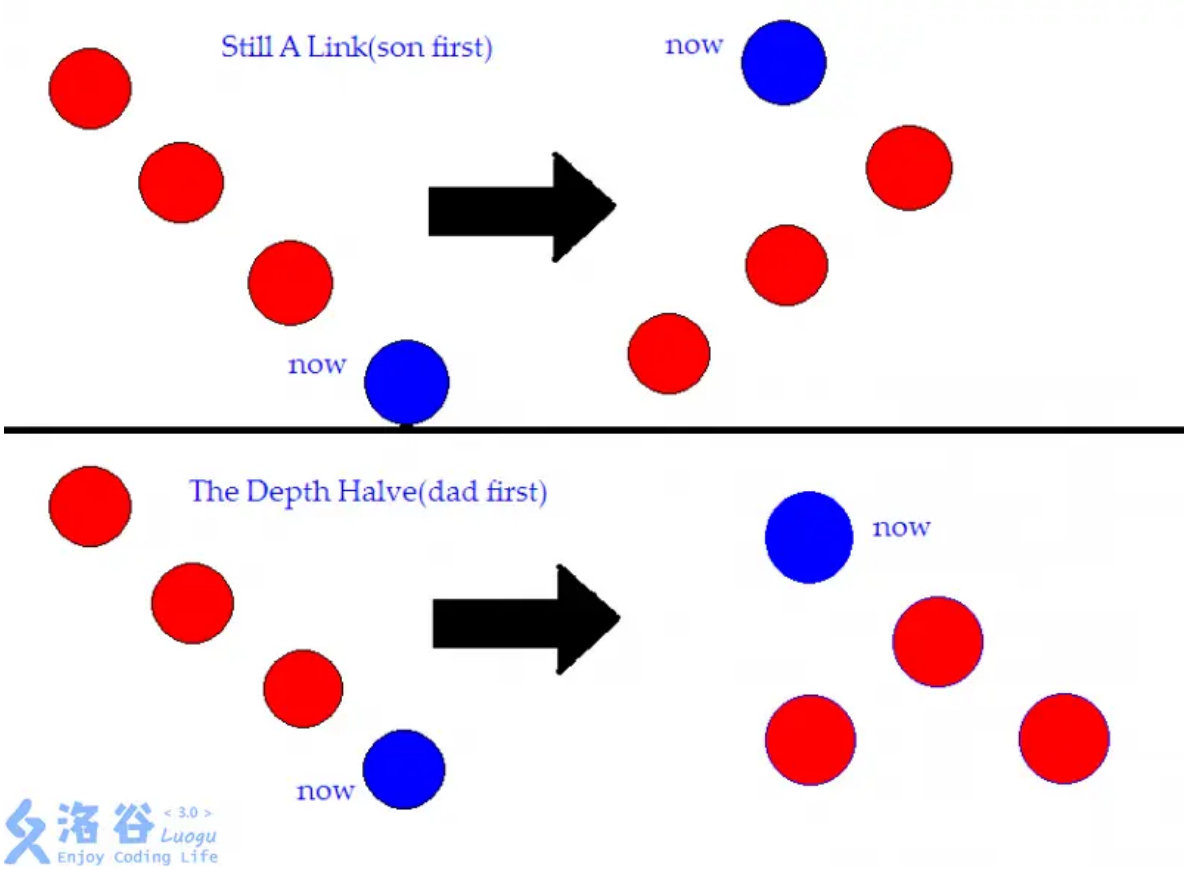
情况一：$now$ , $now$ 的父亲 和 父亲的父亲 不在同一直线上，这时候直接转一下就行。




情况二：$now$ , $now$ 的父亲 和 父亲的父亲 在同一直线上

若此时依次旋转，如下图上面的情况，仍然有一条链。

因此要先旋转 $now$ 的父亲 再转 $now$，再重复上述判断过程。



相比treap复杂度是期望 $O(nlogn)$，splay复杂度是均摊 $O(nlogn)$。不会证QAQ

复杂度证明

# 可持久化平衡树

基于非旋平衡树实现。

考虑可持久化线段树是怎么实现的。

每次修改只会修改一条链，只要把修改这条链剖出来处理就行。这种方法叫 $pathcopying$

而非旋平衡树分裂/合并也是修改的是一条链，类似做同样的事就行。

---

STL中的平衡树实现：

```
STL中使用一个平衡树使用 set<int>tree; 建出一棵元素为int类型的平衡树
相关函数：
tree.begin():得到一个指向第一个元素的迭代器（指针）
tree.end():得到最后一个元素的后面一个空节点的指针
迭代器可以直接通过++来移动
tree.empty()：是否为空
tree.size()：元素数目
tree.clear():清空
tree.insert(x)：插入x
tree.erase(x)：删除x
tree.count(x)：查询x的数目
tree.lower_bound(x)：返回指向首个不小于x的元素的迭代器
tree.upper_bound(x)：返回指向首个大于x的元素的迭代器
set中的元素不能重复，插入两次相同元素只会有一个节点在set中。
set中元素不能修改，默认的比较运算符是小于号
底层实现是红黑树
```

# 题目选讲

## P3391 【模板】文艺平衡树

您需要写一种数据结构（可参考题目标题），来维护一个有序数列。

其中需要提供以下操作：翻转一个区间，例如原有序序列是 $5\ 4\ 3\ 2\ 1$，翻转区间是 $[2,4]$ 的话，结果是 $5\ 2\ 3\ 4\ 1$。

最终要求 $m$ 次变换后的序列。

对于 $100\%$ 的数据，$1 \le n, m \le 100000$，$1 \le l \le r \le n$。

Sol:

fhq非旋树实现的方法：反转一个区间就是将这个区间的树分裂出来并打上tag

---

P5055 【模板】可持久化文艺平衡树

您需要写一种数据结构，来维护一个序列，其中需要提供以下操作（**对于各个以往的历史版本**）：

1. 在第 $p$ 个数后插入数 $x$。
2. 删除第 $p$ 个数。
3. 翻转区间 $[l,r]$，例如原序列是 $\{5,4,3,2,1\}$，翻转区间 $[2,4]$ 后，结果是 $\{5,2,3,4,1\}$。
4. 查询区间 $[l,r]$ 中所有数的和。

**和原本平衡树不同的一点是，每一次的任何操作都是基于某一个历史版本，同时生成一个新的版本（操作 $4$ 即保持原版本无变化），新版本即编号为此次操作的序号。**

**本题强制在线。**

第一行包含一个整数 $n$，表示操作的总数。

接下来 $n$ 行，每行前两个整数 $v_i,\mathrm{opt}_i$，$v_i$ 表示基于的过去版本号（$0\le v_i<i$），$\mathrm{opt}_i$ 表示操作的序号（$1\le \mathrm{opt}_i\le 4$）。

若 $\mathrm{opt}_i=1$，则接下来两个整数 $p_i,x_i$，表示操作为在第 $p_i$ 个数后插入数 $x$。
若 $\mathrm{opt}_i=2$，则接下来一个整数 $p_i$，表示操作为删除第 $p_i$ 个数。
若 $\mathrm{opt}_i=3$，则接下来两个整数 $l_i,r_i$，表示操作为翻转区间 $[l_i,r_i]$。
若 $\mathrm{opt}_i=4$，则接下来两个整数 $l_i,r_i$，表示操作为查询区间 $[l_i,r_i]$ 的和。

**强制在线规则：**
**令当前操作之前的最后一次 $4$ 操作的答案为 $lastans$（如果之前没有 $4$ 操作，则 $lastans=0$）。**
**则此次操作的 $p_i,x_i$ 或 $l_i,r_i$ 均按位异或上 $lastans$ 即可得到真实的 $p_i,x_i$ 或 $l_i,r_i$。**

对于 $100\%$ 的数据，$1\le n\le 2\times 10^5$，$|x_i|<10^6$。

Sol:

第一种做法就是可持久化fhq，标记下放时对被下放的节点也新建节点就行。

第二种做法是分别维护正序列和反序列的两棵fhq树，如1 5 3 4 2，同时维护2 4 3 5 1的序列。

区间反转时就将两棵树上对应区间split出来交换一下。

---

## P2042 [NOI2005] 维护数列

给出一个序列（有正有负），支持：
插入或删除一个数
区间赋值，区间翻转
区间求和，求整个序列的最大子段

Sol:

fhq板子加上个求最大子段和。

---

## P4309 [TJOI2013] 最长上升子序列

给定一个序列，初始为空。现在我们将 1 到 $N$ 的数字**从小到大依次**插入到序列中，每次将一个数字插入到一个特定的位置。每插入一个数字，我们都想知道此时最长上升子序列长度是多少？

$100\%$ 的数据 $n \leq 10^5$。

强制在线?

Sol:

由于每插入一个数，这个数是序列中最大的，答案要么是没插入前序列的答案，要么是以插入数结尾的LIS的长度。

若插入位置为 $k$，就是找 $[1,k]$ 中的最长LIS和原答案比较并将其插入数据结构就行。

支持插入可以先离线后用树状数组，也可以在线平衡树做。

---

## [POI2008]砖块Klo

现在有n个数字，可以进行以下两种操作:
选择一个数，使其-1
选择一个数，使其+1
最少通过多少次操作，可以使数列中出现长度为K的连续子段，满足该段中数字完全相同

n,k<=10w，数字大小不超过 100w

Sol:

如果确定了某一段变相同，如何计算最小代价
等价于一个数轴上k个点，到某个点距离之和最小，显然是中位数
所以只需要滑动窗口维护一下中位数即可
值域线段树和平衡树等可以动态维护中位数的结构均可

---

## P3224 [HNOI2012] 永无乡

永无乡包含 $n$ 座岛，编号从 1 到 $n$，每座岛都有自己的独一无二的重要度，按照重要度可以将这 $n$ 座岛排名，名次用 1 到 $n$ 来表示。某些岛之间由巨大的桥连接，通过桥可以从一个岛到达另一个岛。如果从岛 $a$ 出发经过若干座（含 0 座）桥可以到达岛 $b$，则称岛 $a$ 和岛 $b$ 是连通的。

现在有两种操作:

`B x y` 表示在岛 $x$ 与岛 $y$ 之间修建一座新桥。

`Q x k` 表示询问当前与岛 $x$ 连通的所有岛中第 $k$ 重要的是哪座岛，即所有与岛 $x$ 连通的岛中重要度排名第 $k$ 小的岛是哪座，请你输出那个岛的编号。

- 对于 $100\%$ 的数据，保证 $1 \leq m \leq n \leq 10^5$, $1 \leq q \leq 3 \times 10^5$，$p$ 为一个 $1 \sim n$ 的排列，$op \in \{\texttt{Q},\texttt{B}\}$，$1 \leq u,v,x,y \leq n$。

Sol:

同一并查集内用平衡树维护，并查集合并时启发式合并。

---

练习题：

P3215 [HNOI2011] 括号修复 / [JSOI2011]括号序列

写数据结构一定要固定写法，形成自己的模板！