数据结构进阶

杨乐

blog.csdn.net/yangle61

@897982078

yangle61@163.com

2017 年 1 月 18 日

数据结构进阶

1* 序列型

@ 这类数据结构主要处理“序列”问题。

(1) **线段树**：设计结点状态，熟练掌握标记操作

(2) **主席树**：函数式线段树，函数式编程

(3) **平衡树**：新增、删除操作，区间移动，区间翻转

2* 树型

@ 这类数据结构主要处理“树”上的问题。

(4) **树分治**：点分治、边分治、链分治（树链剖分）

(5) **动态树**：LCT，树上的割边和连边

线段树?

- 学习线段树的几个阶段：
  1. 单点修改，区间询问——信息合并
  2. 区间修改，单点询问——标记与下传
  3. 区间修改，区间询问——标记的正确使用
- 线段树进阶的几个阶段：
  1. 线段树的实现技巧
  2. 状态的设计与合并
  3. 多标记的处理与下传

线段树？线段树的实现方法

线段树的实现方法——三段式

```
if (r<=mid) query(x+x, l, r);
else if (l>mid) query(x+x+1, l, r);
else query(x+x, l, mid), query(x+x+1, mid+1, r);
// 优点：直观易懂
// 缺点：容易写错，繁琐
```

线段树？线段树的实现方法

线段树的实现方法——另一种方法

```
// opl, opr 代表的是操作的区间
void update(int x, int l, int r) {
    if(opl<=l && r<=opr) { operate(x); return; }
    int mid = (l+r) >> 1;
    if(opl<=mid) update(x+x, l, mid);
    if(opr>mid) update(x+x+1, mid+1, r);
}
opl = L, opr = R, update(1, 1, n);
// 缺点：稍微难于理解
// 优点：简单，代码量少，不用重复编程
```

线段树？线段树的实现方法

线段树的实现方法——三段式

```
// 在三段式中往往需要给不同的操作写单独的程序：
void add();
void mult();
int sum();
int max();
// 这样给编程和调试会带来许多不便
```

线段树？线段树的实现方法

线段树的实现方法——另一种方式

```
// 通过定义全局变量opx，来代表不同的操作，从而合并过程；
// 我们可以把操作的范围、类型、参数都放在全局变量中；
void update(int x, int l, int r) {
    if(opl<=l && r<=opr) { // opx为操作类型
        if(opx==1) add(x);
        else if (opx==2) mult(x);
        else if (opx==3) sum(x);
        else max(x);
        return;
    }
    ... }
```

线段树？线段树的实现方法

线段树的实现方法——另一种方式

```
void pup(int x) //合并x的两个子区间中的信息(put-up)
void modify(int x, int y) //对x的区间加上y的修改
void pdw(int x) //将x区间的信息下传(put-down)
```

解决线段树问题的一般范式

calc: 核心操作

```
void calc(x,l,r)
    若区间[l,r]在区间[opl,opr]中:
        对区间[l,r]进行操作（查询/修改）
        返回

    下传标记
    递归左儿子（若满足条件）
    递归右儿子（若满足条件）
    合并信息（如果是修改操作）
```

线段树？例题

线段树例题

- 区间信息的设计——我要记录些什么？
  - SPOJ GSS1/GSS3 ——区间最大子段和
  - Bzoj3813 奇数国——区间内某个值是否出现过
  - P2894 酒店 Hotel ——区间连续一段空的长度
  - Bzoj1858 序列操作——多种操作
- 区间信息的合并——我要怎么把信息合起来？
  - 自创题目 - Melancholy ——背包合并
  - BZOJ 2957: 楼房重建——区间的最长上升子序列
- Codevs3044 矩形面积求并——扫描线
- 多种标记、二分、等等

线段树？例题

SPOJ GSS1/GSS3

- GSS：最大子段和
- 给定一个序列，支持修改操作，查询某一段的 GSS
- 直观的，我们会想到维护一段区间的 GSS
- 那我们就要考虑如何从两段子区间合并出这段的 GSS
- 一段长的区间的 GSS 有三种情况：
  1. 完全在左子区间
  2. 完全在右子区间
  3. 横跨左右区间

线段树？例题

SPOJ GSS1/GSS3

- 一段长的区间的 GSS 有三种情况：



- 前两种情况可使用子区间的 GSS，但如何处理第三种情况？
- 注意到我们可以把区间拆成两部分，这两部分是不相关的。
- 所以我们需要维护一个区间的“最大左子段和”和“最大右子段和”。

线段树？例题

SPOJ GSS1/GSS3

- $gss = \mathbf{MAX}\{l.gss, r.gss, l.rgss + r.lgss\}$
- 接下来的问题，如何维护“最大左子段和”与“最大右子段和”。



线段树？例题

SPOJ GSS1/GSS3

- 由 GSS 一直推下来，对于每个节点，我们一共需要设计 4 个状态
  1. GSS：最大子段和
  2. LGSS：最大左子段和
  3. RGSS：最大右子段和
  4. SUM：整段和

线段树？例题

P2894 酒店 Hotel

- 一步一步推：区间最长的连续空位；从左端开始的连续空位；从右端开始的连续空位；最长的连续空位的开始位置。

线段树？例题

Bzoj1858 序列操作

- 对于询问 3：区间 1 的个数。
- 注意，对于区间操作，我们在进行区间操作的时候需要考虑如何维护区间的信息。
- 考虑操作 2 对询问 3 的影响：如果对一段进行了取反操作，我们该怎么维护呢？
- 新区间 1 的个数 = 旧区间 0 的个数
- 所以我们只用同时统计区间 0 的个数，交换就行了

线段树？例题

Bzoj1858 序列操作

- 对于询问 4：区间最多有多少个连续的 1，从左/右端开始有多少个连续的 1。
- 考虑操作 2 对询问 4 的影响：如果对一段进行了取反操作，我们该怎么维护呢？
- 新区间 1 的信息 = 旧区间 0 的信息
- 所以我们只用同时统计区间 0 的信息（多少个连续的 *0*，从左/右端开始有多少个连续的 *0*），交换就行了

线段树？例题

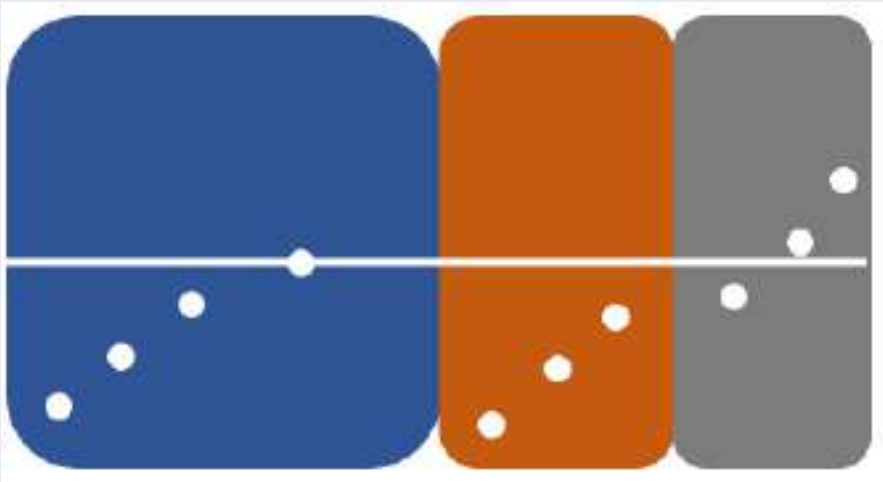
BZOJ 2957: 楼房重建

- 维护“区间的最长上升子序列”
- 左边的上升子序列一定保留；
- 把右边拆成两部分；
- 存在下面两种情况：

线段树？例题

BZOJ 2957: 楼房重建

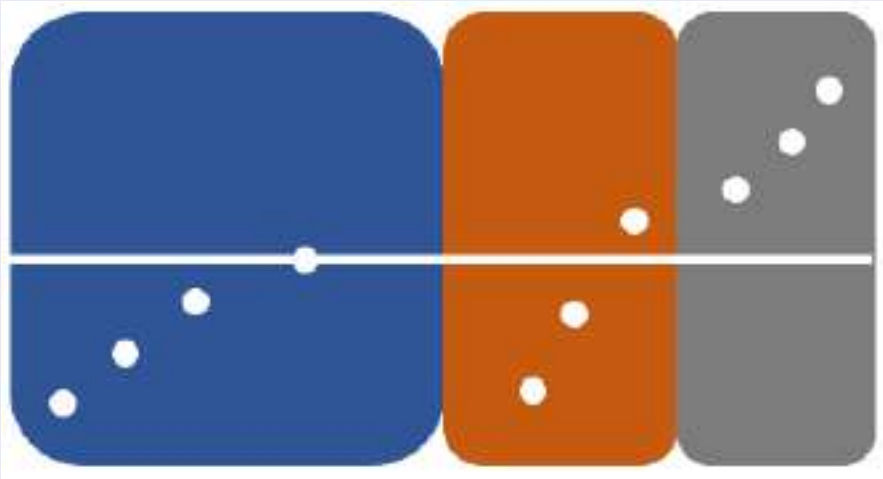
- 情况 1：最大值大于左部分的最大值
- 很明显，左部分是不能成为 LIS 的一部分的；
- 这样我们可以直接考虑右部分了；



线段树？例题

BZOJ 2957: 楼房重建

- 情况 2：最大值小于左部分的最大值
- 很明显，右部分全部都是 LIS 的一部分；
- 现在可以只考虑左部分。



线段树？回顾

@1 实现方法

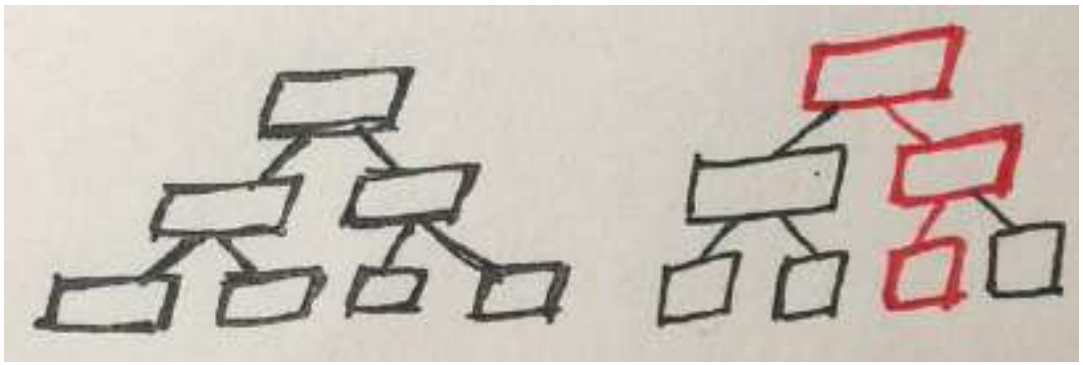
@2 区间设计——记录什么？

@3 区间合并——怎么维护？

@4 典型问题

主席树？主席树的定义

- **函数式编程**：不修改，只新增（保留所有的历史版本）。
- 考虑一次单点修改对整棵树的信息的影响；只有一条链上的信息真正改变了。
- **函数式线段树**：对于线段树所有的**单点修改**操作，不真正的修改，而是通过新增节点的方式来构建。



主席树？实现方式

- 由于除了链上的信息是不变的，所以把直接指向它们就可以了。
- 所以整棵线段树是动态的，要用动态的节点来实现。
- 只需要记录根节点，就能访问“第 x 次修改后的线段树”了。



主席树？主席树的实现方法

主席树——指针，虚拟指针池

```
// 这里使用指针的方式来写
// 同时介绍虚拟指针池的用法（不需要使用new）
// 虚拟指针池和用普通的数组实现想法是一样的，使用计数器
struct node {
    int s;
    node *lc, *rc;
};
node nil[N * lg], *rot[N];
```

主席树？主席树的实现方法

主席树——插入操作

```
// 插入操作
node* insert(node *u, int l, int r, int p) {
    node *ret = ++cnt; // 新增节点
    *ret = *u, *ret->s ++; // 克隆原节点
    if(l == r) return ret;

    int mid = (l + r) / 2;
   if(p <= mid) ret->lc = insert(ret -> lc, l, mid, p);
    else ret->rc = insert(ret -> rc, mid + 1, r, p);
    return ret; // 注意这里要返回新建的节点的位置
}
```

主席树？主席树的实现方法

主席树——初始化

```
cnt = nil->s = 0, rot[0] = nil->lc = nil->rc = nil;
for(int i = 1; i <= n; i++) // 在前一个的基础上插入，并记录根节点
    rot[i] = insert(rot[i-1], 1, dc[0], a[i]);
```

主席树？主席树的实现方法

主席树——查询

```
// 查询操作和普通查询并无差别
int query(node *u, node *v, int l, int r, int k) {
    if(l == r) return dc[l];
    int mid = (l + r) / 2;
    int c = u -> lc -> s - v -> lc -> s;
   if(k <= c) return query(u -> lc, v -> lc, l, mid, k);
    return query(u -> rc, v -> rc, mid + 1, r, k - c);
}
// 在主程序调用的时候，明确是哪个时候的
query(rot[r], rot[l - 1], 1, dc[0], k)
```

主席树？例题

主席树例题

- 区间第 K 大
  - POJ2104 K-th Number ——模板
  - P3168 [CQOI2015] 任务查询系统——前缀和转化
  - BZOJ2809: [Apio2012]dispatching ——*DFS* 序
  - SPOJ COT 树上第 K 大——树上操作
  - BZOJ3524: [Poi2014]Couriers
- 历史信息的查询
  - BZOJ 2653: middle ——二分，*GSS*，主席树维护
  - 自创题 - Trojan ——树上信息保存

主席树？区间第 K 大

主席树例题——区间第 K 大

- 给出一个数列，求 $[l, r]$ 中的第 K 大数。
- 考虑二分：二分第 K 大的数 x，只需求出 $[l, r]$ 中比 x 小的数有多少个 $\rightarrow$ 求出 $[1, r]$、$[1, l-1]$ 区间中比 x 小的数有多少个。
- 考虑维护 $[1, i]$ 的权值线段树，就能回答 $[1, i]$ 区间中比 x 小的数有多少个的问题了。
- 但无法做到维护 $[1, 1], [1, 2], \ldots, [1, n]$ 这 n 个线段树。

主席树？区间第 K 大

主席树例题——区间第 K 大

- 但无法做到维护 $[1,1],[1,2],\ldots,[1,n]$ 这 n 个线段树。
- 考虑到 $[1,i]$ 的线段树只比 $[1,i-1]$ 新增一个数；则可以使用主席树来维护。
- 最后也可以不用二分，只需在线段树上判断就行了。
- 思考：存在单点修改的区间第 K 大问题。

主席树？区间第 K 大

主席树例题——SPOJ COT：树上第 K 大

- 想一想序列的区间第 K 大是怎么做的：前缀和相减，得到某个数在这个区间出现了多少次。
- 那么如何统计某个数在树上出现了多少次？
- 同样是前缀和：记录某个点到根的路径上，这个数出现了多少次；那么路径 X, Y 之间，出现的次数等于 $X + Y - LCA \times 2$（Not Exactly!）
- 整体做法类似于序列的情况。

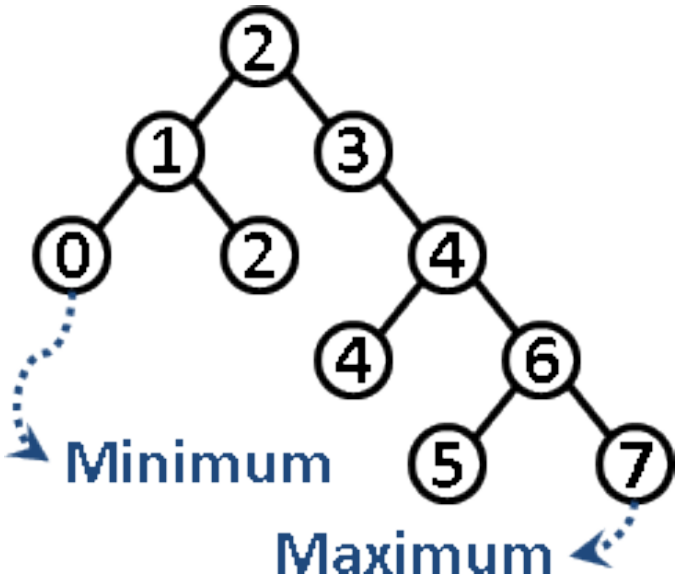
主席树？回顾

1. 函数式编程思想，可持久化数据结构
2. 实现方法
3. 典型问题——区间第 K 大

平衡树？二叉搜索树的定义

- **二叉搜索树**：一棵二叉树，每个点拥有键值，每个点的左子树中所有节点的键值均要小，右子树均要大（即树的中序遍历是有序的）。
- 比起线段树，BST 的特点侧重于维护数据的大小关系（根据数据大小来决定操作/ 而不是位置）。
- 所以 BST 支持插入/删除/查询排名等操作。
- *STL* 中的 *set* 和 *map* 能支持基础的 *BST* 操作。

二叉搜索树? 二叉搜索树的基本操作

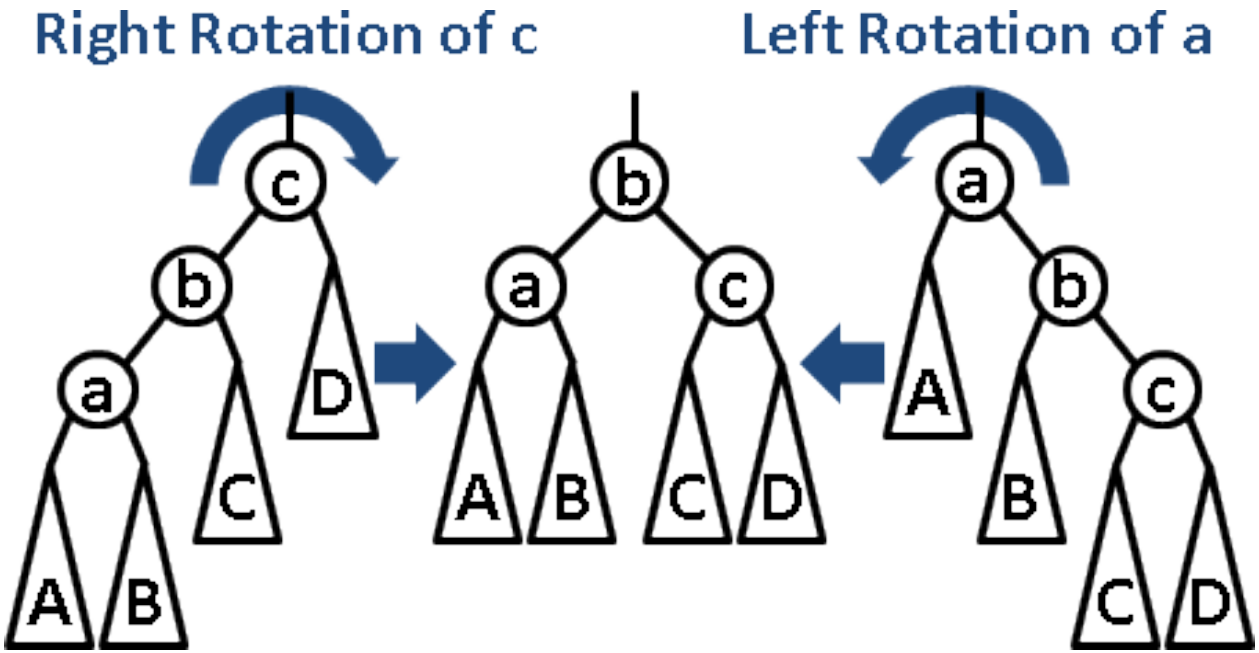


- 插入
- 删除（懒惰删除）
- 查询某元素的排名
- 查询排名第 K 的元素

平衡树？平衡？

- 二叉搜索树最大的问题就是“树有可能跑偏”，最极端的情况是变成一条链。
- 所以面对树即将可能“跑偏”的时候，要通过操作将其“扭”回来。

平衡树？旋转操作——左旋与右旋

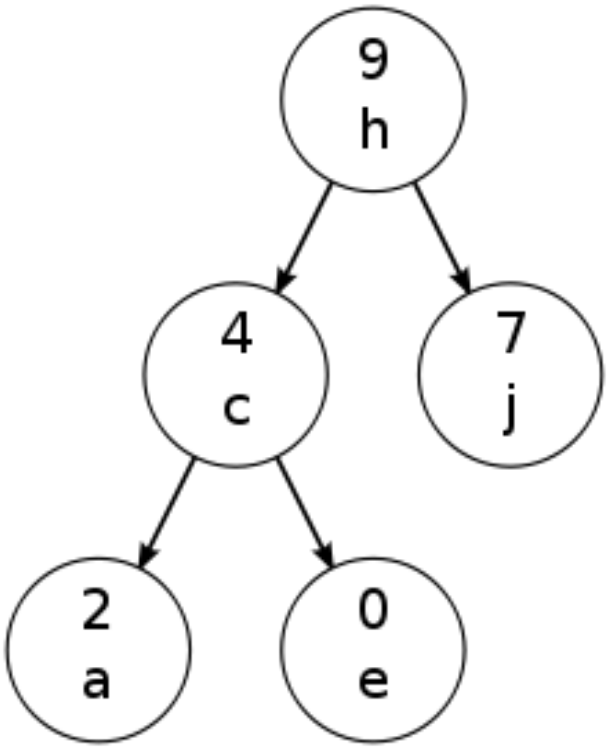


平衡树？平衡树的种类

平衡树的种类

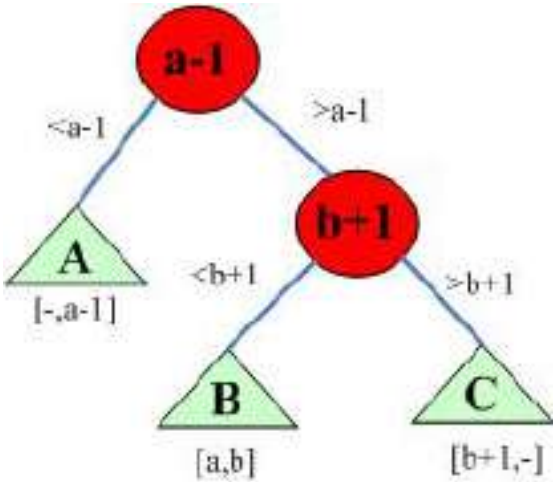
- 严格控制树的高度的平衡树
  - AVL
  - 红黑树（STL 中 set 使用的）
  - Size Balanced Tree
- 通过随机因子保证树的平衡
  - Treap
- 保证均摊时间，并不保证树的严格平衡
  - Splay（伸展树）

二叉搜索树？Treap 的简单介绍



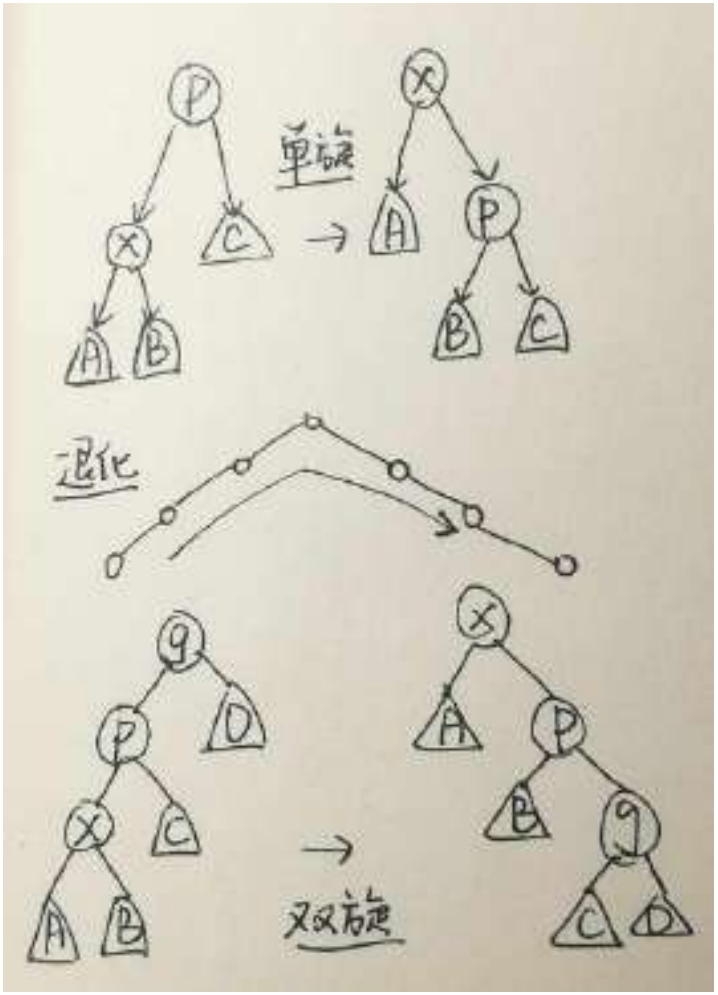
- 给每个点加上“随机值”
- 在键值保证“二叉搜索树”的前提下，随机值要保证“堆”的性质。
- 通过旋转操作达到上浮/下沉的效果。

二叉搜索树? Splay 的介绍



- Splay 的核心操作就是“把一个点通过旋转操作到达根的位置”
- 有了这个操作后，我们可以在树中提取出一个子区间；
- 然后对子区间就可以像线段树一样进行区间合并，区间标记等等操作。

二叉搜索树? Splay 操作详解



旋转操作

- 我们可以通过旋转操作将一个点提上一层；
- 注意到还是会出现退化情况；
- 所以在连续两次相同方向的旋转中，先旋转上层节点。

Splay? Splay 的代码实现

Splay ——旋转

```
struct rec
{
    int a, tag, siz;
    rec *c[2], *f; // 定义两个儿子

    int d(){ return f->c[1] == this; } // 定义方向
    void sc(rec* x, int d){ (c[d]=x)->f = this; }
        // 定义连接操作
}
```

Splay? Splay 的代码实现

Splay ——旋转

```
void zig(rec *x) // 将节点x旋转上一层
{
    int d = x->d();
    rec* p = x->f;
    p->sc(x->c[!d], d);
    if(p==root) x->f = nil, root = x;
    else p->f->sc(x, p->d());
    x->sc(p, !d);
}
```

Splay? Splay 的代码实现

Splay ——主过程

```
// 将x旋转至根节点
void splay(rec *x)
{
    for(rec *y; x!=root; )
    {
        y = x->f;
        if(y!=root) (x->d()^y->d()) ? zig(x):zig(y);
        zig(x);
    }
}
```

Splay? Splay 的代码实现

Splay ——插入操作

```
// 插入q节点
void insert(rec *q)
{
    rec *p = root;
    for(;;){
        int d = q->a > p->a; // 插入方向
        if(p->c[d] != nil) p = p->c[d];
        else {p->sc(q, d), splay(q); break;}
    } // 最后记得伸展操作
}
```

Splay? Splay 的代码实现

Splay ——查询第 K 个元素

```
void select(int k){
    rec *p = root;
    for(;;){
        int t = p->c[0]->siz;
        if(t >= k) p = p->c[0];
        else if(t + 1 < k) k -= t+1, p = p->c[1];
        else break;
    }
    splay(p); // 最后伸展
}
```

Splay? Splay 的代码实现

Splay ——size 的维护

```
// 我们会发现，在旋转过程中，size（子树大小）是会变的
// 需要维护子树的大小
void rec::pup(){ siz = c[0]->siz + c[1]->siz + 1; }
// 在zig（旋转过程中），也需要维护
p->pup();
// 在splay过程的最后，也需要维护
x->pup();
```

Splay? Splay 的代码实现

Splay ——旋转 + size 的维护

```
void zig(rec *x) // 将节点x旋转上一层
{
    int d = x->d();
    rec* p = x->f;
    p->sc(x->c[!d], d), p->pup(); // 合并信息!!
    if(p==root) x->f = nil, root = x;
    else p->f->sc(x, p->d());
    x->sc(p, !d);
}
```

Splay? Splay 的代码实现

Splay ——主过程 + size 的维护

```
// 将x旋转至根节点
void splay(rec *x) {
    for(rec *y; x!=root; ) {
        y = x->f;
        if(y!=root) (x->d()^y->d()) ? zig(x):zig(y);
        zig(x);
    }
    x->pup(); // 最后合并信息!!
}
```

Splay? Splay 的代码实现

Splay ——翻转标记

```
// 在一些题目中会涉及到“区间翻转”操作，所以要给区间打上翻转标记
void rec::mrev(){rev ^= 1;}
// 在下传标记的时候记住调换左右子树的位置
void rec::pdw(){
    if(rev) swap(c[0], c[1]), c[0]->mrev(),
        c[1]->mrev(), rev=0;
}
```

Splay? Splay 的代码实现

Splay ——Splay + 标记下传

```
// 在伸展操作中，我们必须要把涉及到的节点的标记先下传
void splay(rec *x) {
    for(rec *y; x!=root; ) {
        if((y=x->f) != root) y->f->pdw(); // !!
        y->pdw(), x->pdw(); // 顺序下传!!
        if(y!=root) (x->d()^y->d()) ? zig(x):zig(y);
        zig(x);
    }
    x->pup();
}
```

Splay? Splay 的代码实现

Splay ——查询第 K 个元素

```
// 同理，在查询操作中，也先需要下传标记
void select(int k){
    rec *p = root;
    for(;;){
        p->pdw(); // 下传标记!!
        int t = p->c[0]->siz;
        if(t >= k) p = p->c[0];
        else if(t + 1 < k) k -= t+1, p = p->c[1];
        else break;}
    splay(p);} // 最后伸展
```

平衡树？例题

平衡树的例题

- 由于处理大小关系的平衡树（查询比某一个数的前驱/后继，查询最值）可以用 set 完美替代，所以这方面的题目很少；
- 故平衡树的题目都是处理序列问题；
- 特点是插入/删除，区间翻转。
  - P3369 普通平衡树——模板
  - Codevs 1286 郁闷的出纳员 (NOI2004)——插入删除，打标记
  - Codevs 1743 反转卡片——*Splay*，翻转操作
  - Codevs 3723 排序机械臂——*Splay*，翻转操作
  - Codevs 1758 维护数列 (NOI2005)——*Splay* 操作集大成者

平衡树？回顾

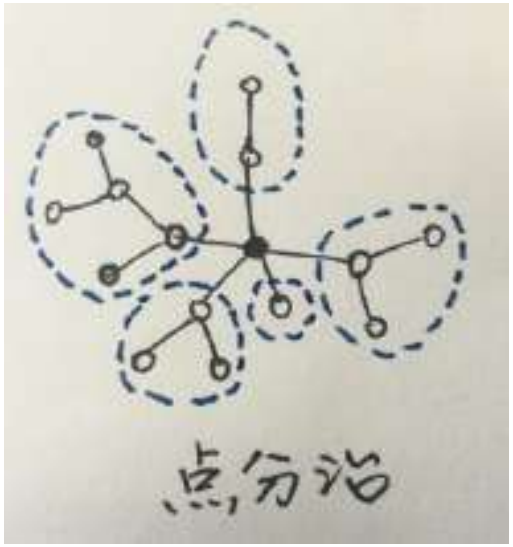
@1 二叉搜索树与平衡树

@2 平衡树的种类

@3 平衡树的实现方法：维护信息，标记下传

@4 平衡树的典型例题

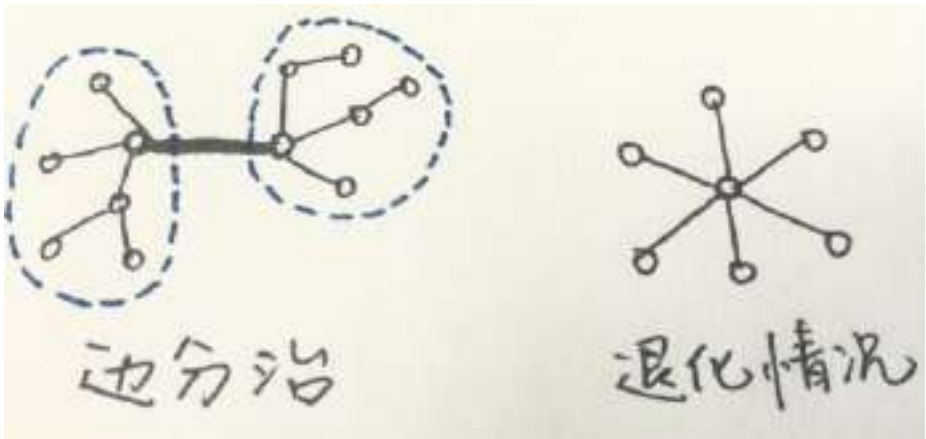
点分治？点分治的定义



- 序列的分治方法：快速排序，归并排序 ——将一个大问题分为若干个规模更小的子问题。
- **树分治**：树上的分治方法——将一棵树上的问题分为若干个子树的问题。
- **点分治**：找树上的一个顶点，将它删去，剩余的结点则会形成若干个连通块。

点分治？点分治与边分治

- **边分治**：找树上的一条边，将它删去，剩余的结点则会形成**两个**连通块。
- 注意！一般情况下推荐使用点分治，边分治在遇到“菊花型”情况时会有严重的退化。



点分治？例题

点分治例题——点对信息

- 一般的点分治的题目均是要求统计点对信息。
  - POJ1741tree ——路径和小于 K
  - BZOJ2152 聪聪可可——路径和 3 的倍数
  - Tsinsen A1486 树——路径 XOR 最大
  - HDU4812 D Tree ——路径乘积模 10^6+3 等于 K
  - Codechef BTREE ——树上某个顶点 X 在距离 D 以内

点分治？点对统计

点分治例题——统计点对信息

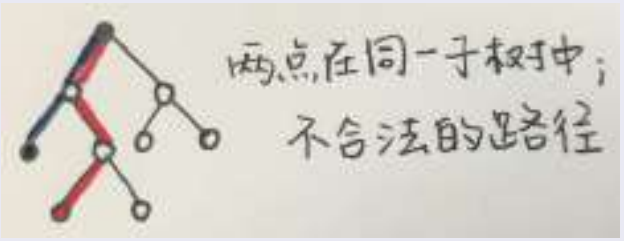
- POJ1741tree ——路径和小于 K
- 在确定分割点后，所有的路径分为两类：
  1. 经过分割点的：路径跨过两个子树
  2. 不经过分割点的：路径完整的在一个子树里（递归入子树中解决）



点分治？点对统计

点分治例题——统计点对信息

- POJ1741tree ——路径和小于 K
- 对于第一个问题，只需统计每个点到根的距离；对于一个点 x（设其距离为 D），以它为端点的答案是 $A - A'$，其中 A 为所有点距离不大于 $K - D$，A' 为 x 所在的子树中距离不大于 $K - D$。（去除不合法的路径）



点分治？例题

点分治例题——Tsinsen A1486 树——路径 *XOR* 最大

- 对于这类的最大值问题，我们不能使用上面这种去重的方法，必须严格将子树间分开。
- 有一种方法是，设有子树 $A, B, C \ldots$，则统计 B 与 A 之间的路径，C 与 $A + B$ 之间的路径...
- 对于路径和小于 K，记录一个表，记录的是前 i 个子树每个点到根的距离，当有新的子树加入的时候，只要和这个大表比较就可以了。
- 于是这道题目简单的记录一个 Trie 就可以了。对于一棵新加入的子树，先在原来的 Trie 中查询，再整体插入到 Trie 中。

点分治？例题

点分治例题——统计点对信息

- BZOJ2152 聪聪可可——路径和 *3* 的倍数
- 记录子树中，有几个点到根的路径模 3 余 0，余 1，余 2。
- HDU4812 D Tree ——路径乘积模 10^6+3 等于 K
- 记录子树中，有几个点到根的路径模 10^6+3 余 0，余 1，余 2...。

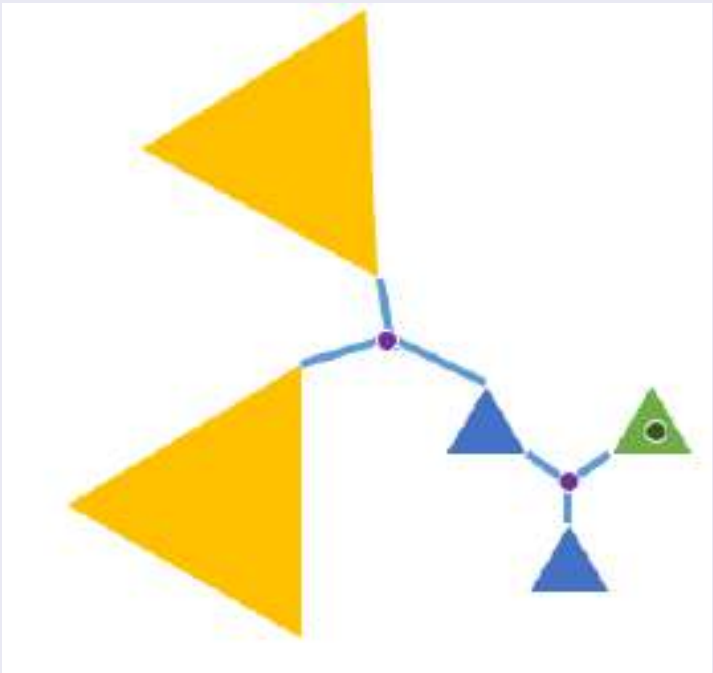
点分治？例题

点分治例题——Codechef BTREE ——树上某个顶点 X 在距离 D 以内

1. 考虑点分治：在第一层分治中，假若节点 X 距离分治点 U 相距 d，则只需统计在其余的子树中，距离小于 $D-d$ 有多少个点。
2. 接下来递归进入节点 X 所在的子树当中。
3. 由于点分治的层数最多有 $LogN$ 层，每次的询问的时间复杂度为 $O(LogN)$ 的。

点分治？例题

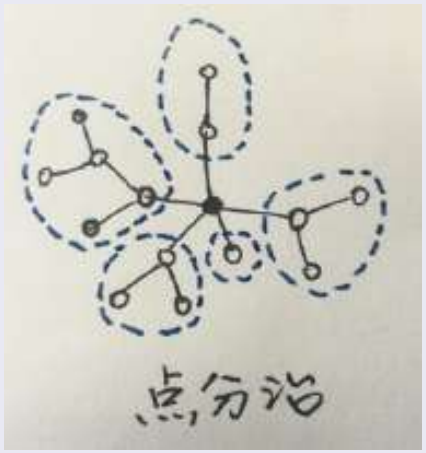
点分治例题——Codechef BTREE ——树上某个顶点 X 在距离 D 以内



点分治？代码实现

点分治——如何找根？

- 现在的问题是，如何找到一个合适的根？
- 序列分治：对半分；
- 树分治：尽量让子树的大小都小于整棵树的一半。
- 事实证明这是可以做到的！



点分治——代码实现

点分治找根——DFS1（预处理结点大小）

```
void sizd(int x, int f=0)
{
    size[x] = 1;
    for(int p=end[x], v; p; p=next[p])
        if(v=to[p], !vis[v] && v!=f)
            sizd(v, x), size[x] += size[v];
}
```

点分治——代码实现

点分治找根——DFS2（查找合适的根）

```
void pickd(int x, int f=0)
{
    int ok(size[x]*2>=all);
    for(int p=end[x], v; p; p=next[p])
        if(v=to[p], !vis[v] && v!=f)
            pickd(v, x), ok &= size[v]*2<=all;
    if(ok) centre = x;
}
```

点分治——代码实现

点分治主程序

```
int construct(int x){
    sizd(x), all=size[x], pickd(x);
    vis[x = centre] = 1; // 标记不能走

    for(int p=end[x], v; p; p=next[p])
        if(v=to[p], !vis[v])  dfs(v, x); // 统计信息
    ... // 统计信息
    for(int p=end[x], v; p; p=next[p])
        if(v=to[p], !vis[v]) construct(v);

    return x;}
```

点分治？回顾

@1 树分治

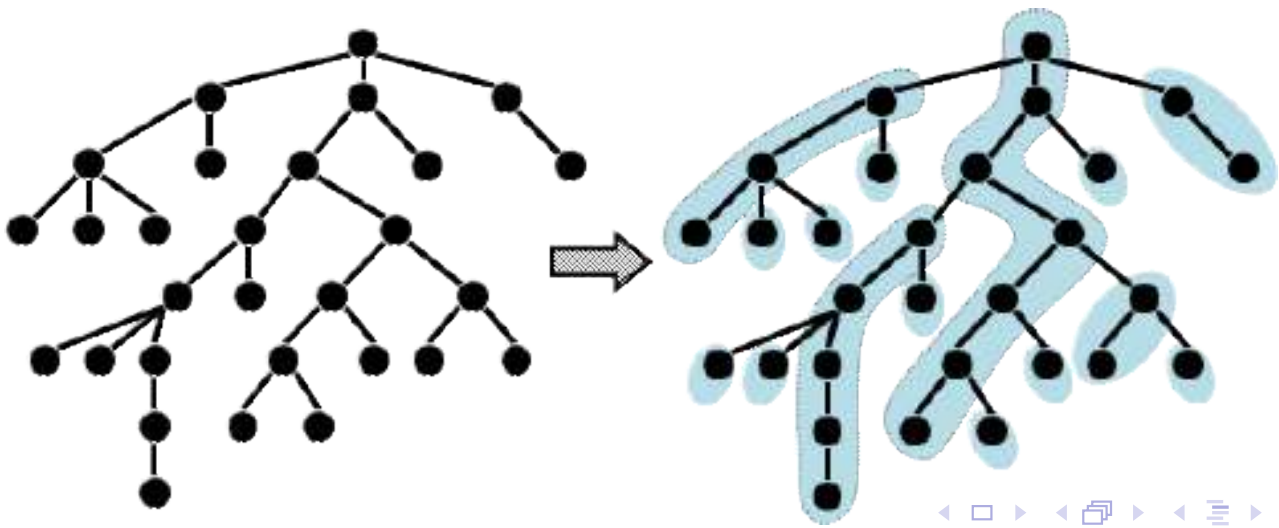
@2 点分治与边分治

@3 典型例题——树上点对统计

@4 点分治的实现——找重心

树链剖分？树链剖分的定义

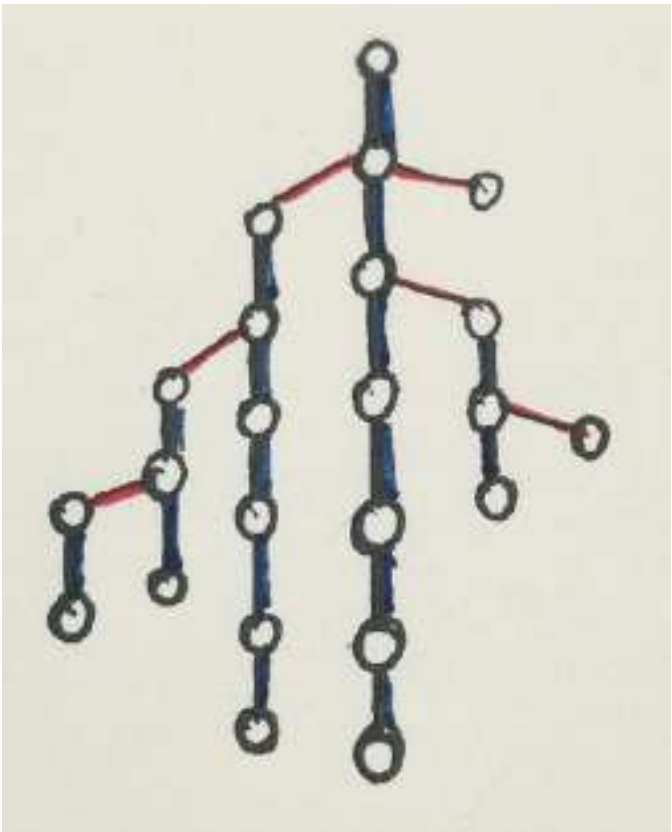
- **树分治：**树上的分治方法——将一棵树上的问题分为若干个子树的问题。
- **树链剖分：**将树划分成若干条链；这样树上的一条路径可以由若干条子路径的所合成；
- **为什么？：**将树上的路径操作 $\rightarrow$ 序列操作。序列操作就可以使用线段树等数据结构了。



树链剖分？树链剖分的方式和性质

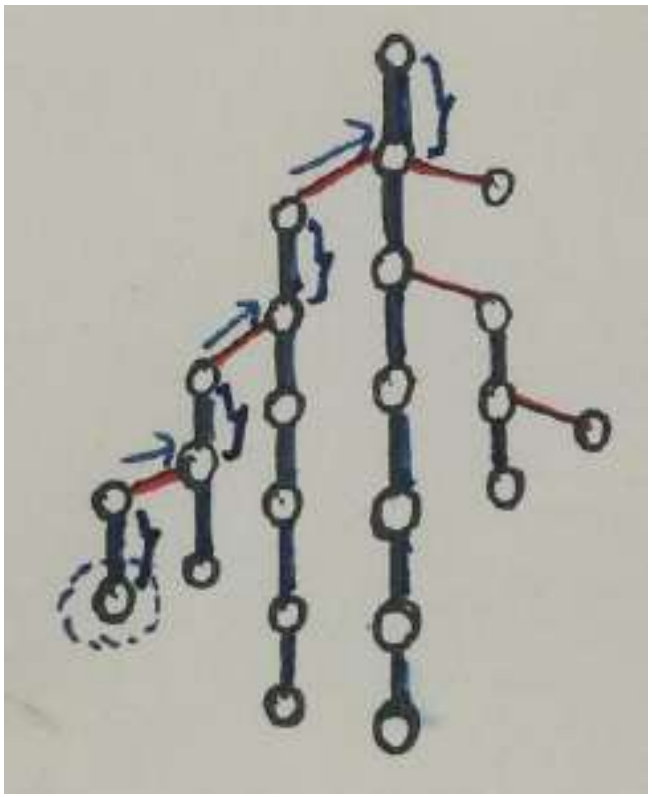
- **如何剖分：**直观的，当一条链上有更多的点，这样的划分方式似乎会更好。
- **剖分方式：**当一个点有多个儿子的时候，链连向子树最大的儿子。
- **重链与轻链：**我们称在链上的边为重链，不在边上的为轻链。
- **剖分的性质：**从根到任何一个节点，至多会经过 $LogN$ 条重链，$LogN$ 条轻链。这样划分就能保证时间复杂度。

树链剖分？树链剖分的结构



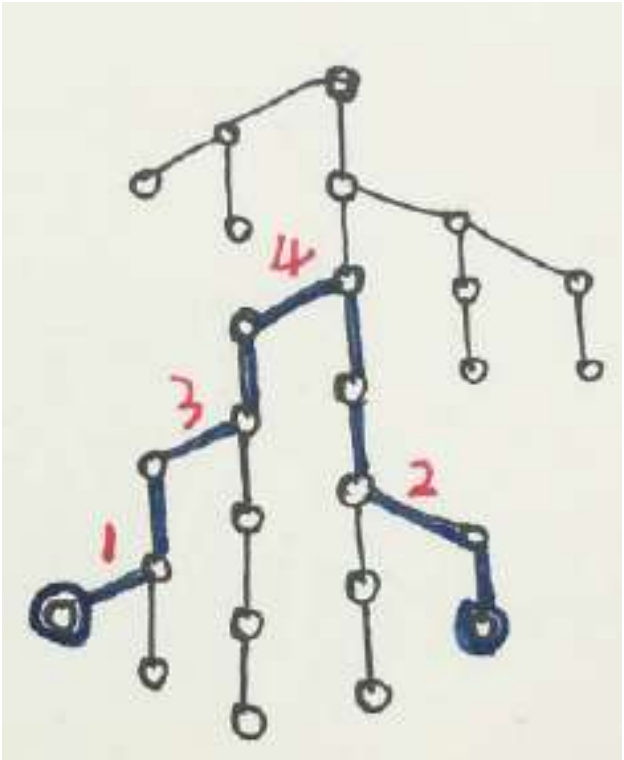
- 可以这样理解树链剖分，把重链拉直，把轻链斜着放。
- 这样每一条重链相当于一个数列，可以在上面维护线段树等数据结构。
- 而在树上的每一条路径都能拆分为**数列上的一段**拼凑起来的。

树链剖分？树链剖分的结构



- 如何找到树上的一条路径所对应的重链的区间？
- 简化版问题：从某一个节点到根的路径？
- 路径上肯定是重链 -轻链 -重链 -轻链... 的交错方式；
- 模拟点在链上的行走过程：首先跳到重链的顶端，然后跳过一段轻链，一直重复至跳到根节点。

树链剖分？树链剖分的结构



- 如何找到树上的一条路径所对应的重链的区间？
- 还是模拟两个点在链上的行走过程：两点均是重链 -轻链 -重链 -轻链... 的交错方式；
- 怎么保证它们能刚好跳到同一条链上就停下来？（不会跳过头）
- 解决方法：让重链深的先跳。

树链剖分——代码实现

HLD - 构建树（找出重儿子）

```
void df1(int x, int f=0)
{
    siz[x]=1, dep[x]=dep[f]+1, ff[x]=f;
    for(int p=end[x]; p; p=next[p]) if(to[p]!=f)
    {
        df1(to[p], x), siz[x]+=siz[to[p]];
        if(siz[to[p]]>siz[son[x]]) son[x]=to[p];
    }
}
```

树链剖分——代码实现

HLD - 构建树（标记 DFS 序，预处理重链顶端）

```
// 把重链上的点连续标号，这样用一棵大线段树就能处理了
void df2(int x, int f=0)
{
    dfn[x]=++dfn[0];
    if(son[x]) top[son[x]]=top[x], df2(son[x], x);
    for(int p=end[x]; p; p=next[p])
        if(to[p]!=f && to[p]!=son[x])
            top[to[p]]=to[p], df2(to[p], x);
    sec[x]=dfn[0];
}
```

树链剖分——代码实现

HLD - 路径操作（给出两个点，修改/查询之间的路径）

```
int ask(int r, int p, int q) {
    int ret(0);
    for(; top[p]!=top[q]; ) {  // 挑深的来跳
        if(dep[top[p]] < dep[top[q]]) swap(p,q);
        ret += query(r, 1, n, dfn[top[p]], dfn[p]);
        p = ff[top[p]]; // 跳轻链
    }
    if(dep[p] > dep[q]) swap(p, q); // 最后的链上的处理
    ret += query(r, 1, n, dfn[p], dfn[q]);
    return ret;
}
```

树链剖分？例题

树链剖分——树上路径操作

- 一般涉及到树上路径操作的都可以用树链剖分来处理：
  - BZOJ 1036: [ZJOI2008] 树的统计 Count ——模板
  - HDU 3966 Aragorn's Story ——模板
  - NOI2015 软件包管理器——简单
  - BZOJ 2243: [SDOI2011] 染色——线段树操作
  - BZOJ 3531: [Sdoi2014] 旅行——动态线段树
  - POJ 2763 Housewife Wind ——边权转点权
  - POJ 3237 Tree ——边权转点权
  - HDU 5274 Dylans loves tree ——异或操作
  - HDU 5840 This world need more Zhu ——根据数据范围分块处理，难

树链剖分？回顾

@1 树链剖分

@2 重链，轻链，路径的表示

@3 树链剖分的代码实现——（构建树，找出路径）

@4 典型例题——路径操作

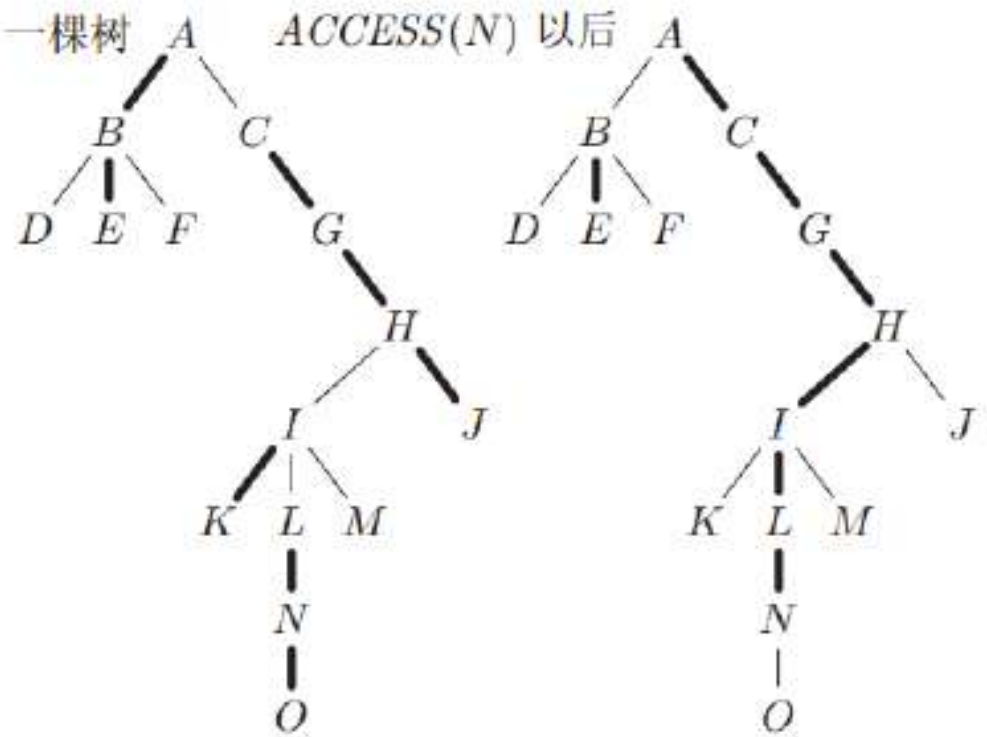
动态树？动态树的定义？

- 动态树处理这样的一类问题：
  - 删边操作：删去“森林”中的一条边
  - 连边操作：将两棵树用一条边连起来，成为一棵更大的树
  - 询问操作：询问两个点是否在一棵树内
- **LCT(Link - Cut - Tree)**是实现这样问题的一个数据结构。
- 它的实现方式和树链剖分有点类似，但是链的划分是动态的。

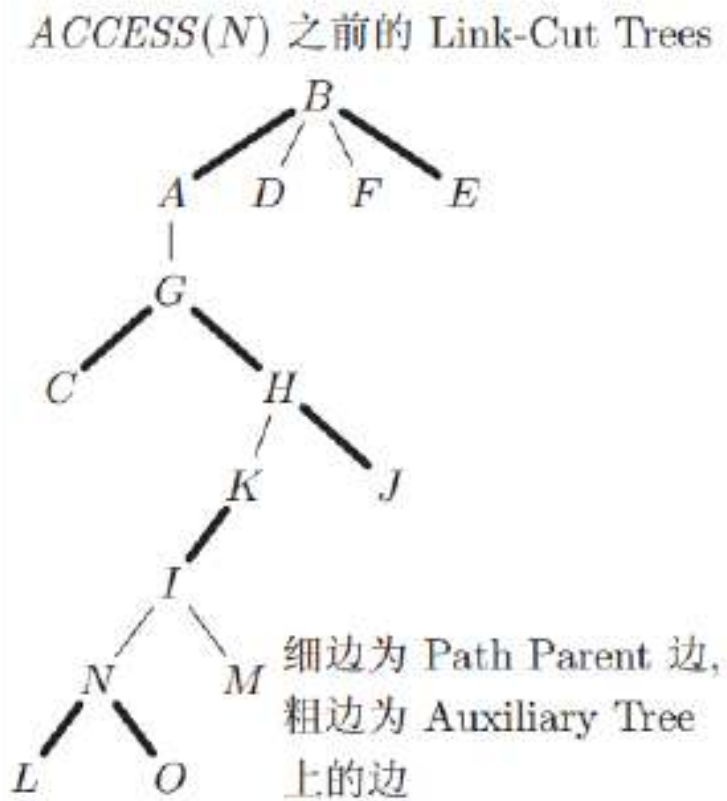
LCT? LCT 的实现方法

- 将所有的边分为实边和虚边。其中每个点最多向一个儿子连一条实边（可以不连）。
- 实边相连的点，必定是树上从上到下的一条链。用 Splay 存储这些点的信息。
- 注意到，如果使用虚边相连，则还是要连一条边到父亲节点（即使不是父亲真正的两个儿子之一）。

LCT? LCT 的结构示意图



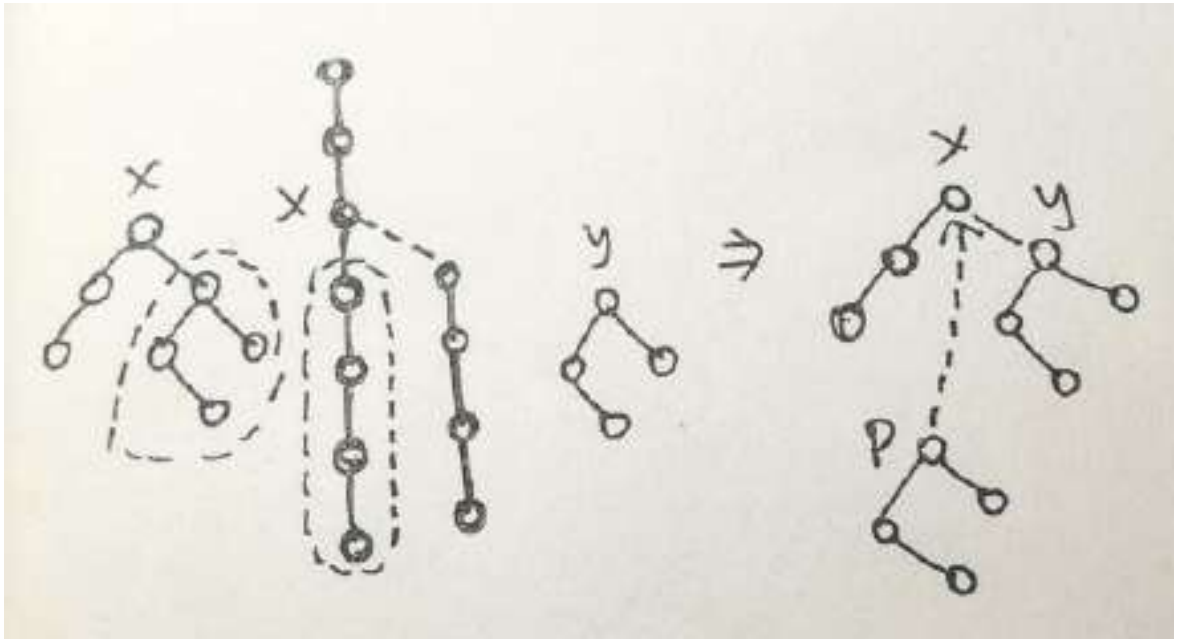
粗边为 Preferred Edge, 细边为普通边



LCT? LCT 的 Access 操作

- **Access 操作**，就是把一个点到根的路径全都变为实边，同时将与自己儿子的实边变为虚边。
- 类似于树链剖分，实现时也是模拟顶点一直向上走的过程。

LCT? LCT 的 Acess 操作示意图



LCT 的 Access 操作——代码实现

LCT - Access 操作

```
void access(rc *x) {
    rc *v = x, *y = nil; // y保存当前底下已经合并完的链
    for(int u; x!=nil; )
    {
        splay(x), x->pdw();
        x->sc(y, 1), x->pup();
        y = x, x = x->f;
    }
    splay(v);
}
```

LCT 的 Access 操作——代码实现

LCT - 判断根

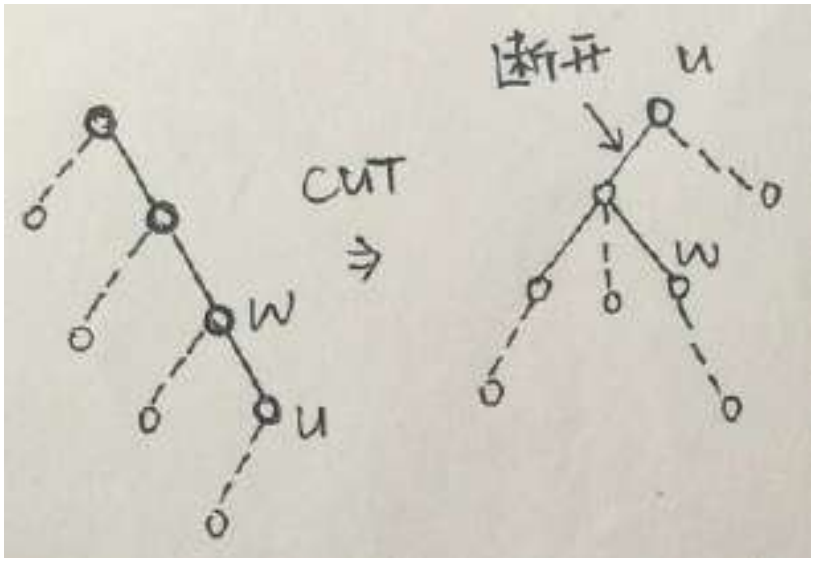
```
// 在LCT中，有两种根：
// 1 - 自己所在链的根，但还是有虚边，连向父亲
// 2 - 整棵树的根
// 所以判断根需要做出相应的改变
// 判断情况1
int rt(){return f->c[1]!=this && f->c[0]!=this;}
// 判断情况2
x->f == nil
// 旋转部分，伸展操作也会有相应的改变，具体看代码实现
```

LCT? LCT 的 Cut 操作

@ **Cut 操作：**把一个点 x 和父亲的路径割开。

@1 $Access(x)$

@2 断开 x 与左子树的连接



LCT? LCT 的 Getroot 操作

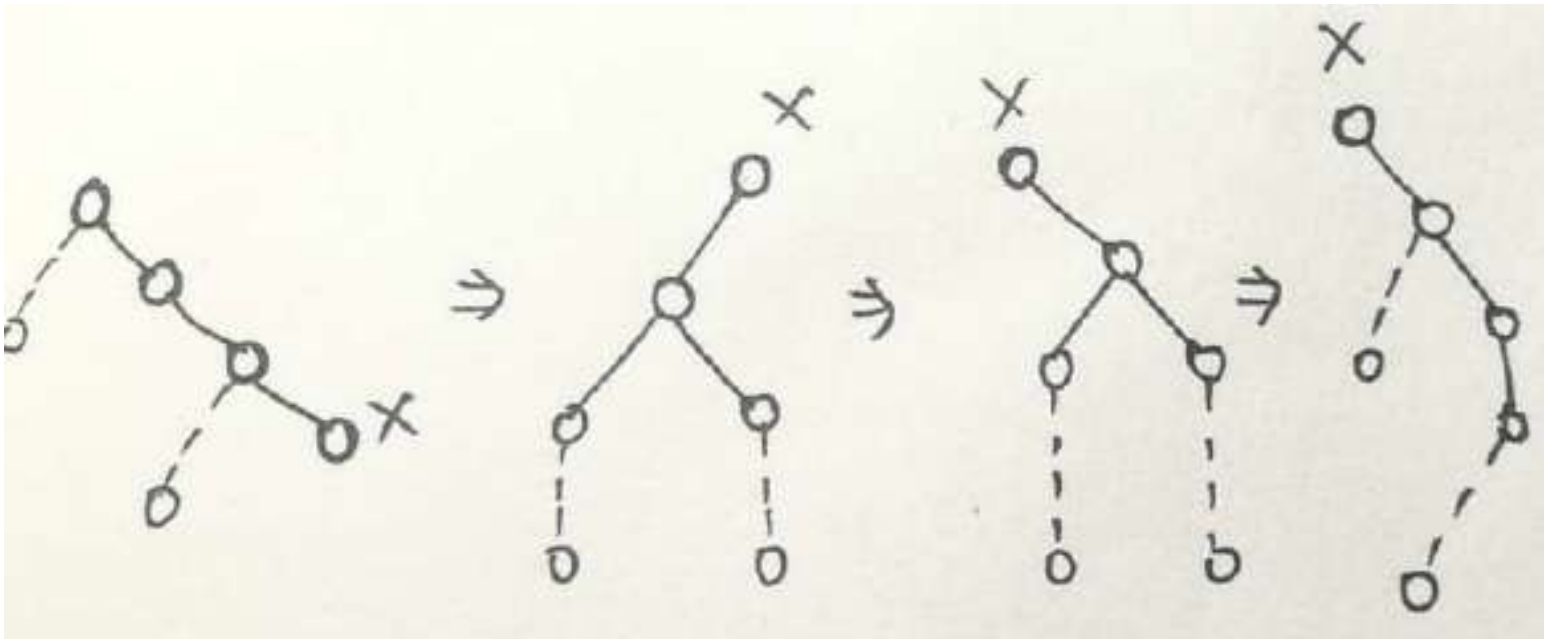
0. **Getroot 操作：** 查询 x 节点所在树的根。
1. $Access(x)$
2. 在 x 的 Splay 上一直向左走

LCT? LCT 的 Evert 操作

@ **Evert 操作：** 把点 x 换成这棵树的根。

@1 $Access(x)$

@2 将 x 所在的链翻转

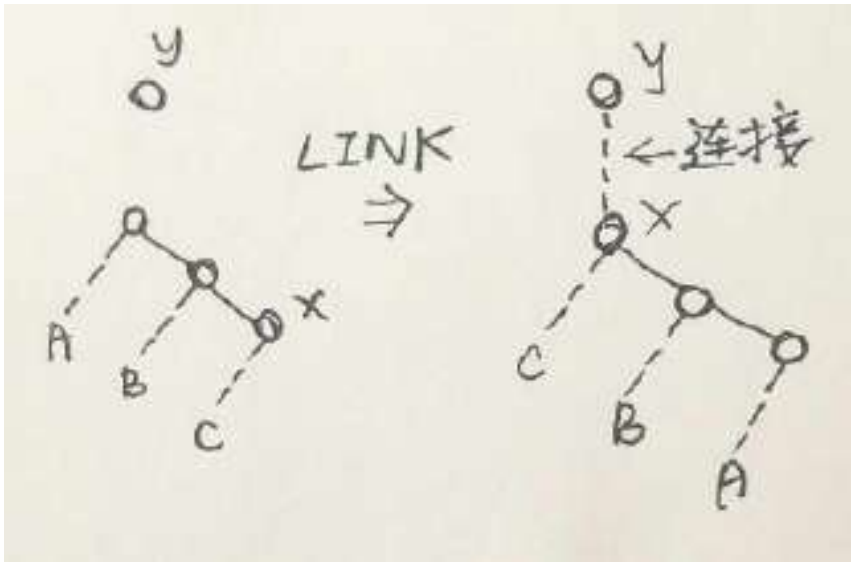


LCT? LCT 的 Link 操作

@ **Link 操作：**把两个点 x, y 连接到一起。

@1 $Evert(x)$

@2 x 的父亲指向 y（x, y 之间连一条虚边）



动态树？例题

- HZW 学长的博客
- **动态树问题**：增删边，查询链的信息（通过换根和 *Access* 操作可以得到树上任意一条链的信息）。
  - Bzoj1180 [CROATIAN2009] OTOCI
  - Hdu4010 Query on The Trees ——边权转化为点权
  - NOI2014 魔法森林——动态增删
  - Bzoj2002 [Hnoi2010] Bounce 弹飞绵羊——动态增删
- **特意设计成 LCT 形式的题目**：题目给出的操作十分诡异，但和 LCT 的操作是符合的。
  - BZOJ 3779 重组病毒

动态树？回顾

@1 动态树问题

@2 LCT：实边、虚边、树的结构

@3 LCT 的操作：Access, Cut, Link, Evert, Getroot

@4 典型例题