高级搜索树

Splay

动机——局部性

伸展——到近的地方

逐层伸展

双层伸展

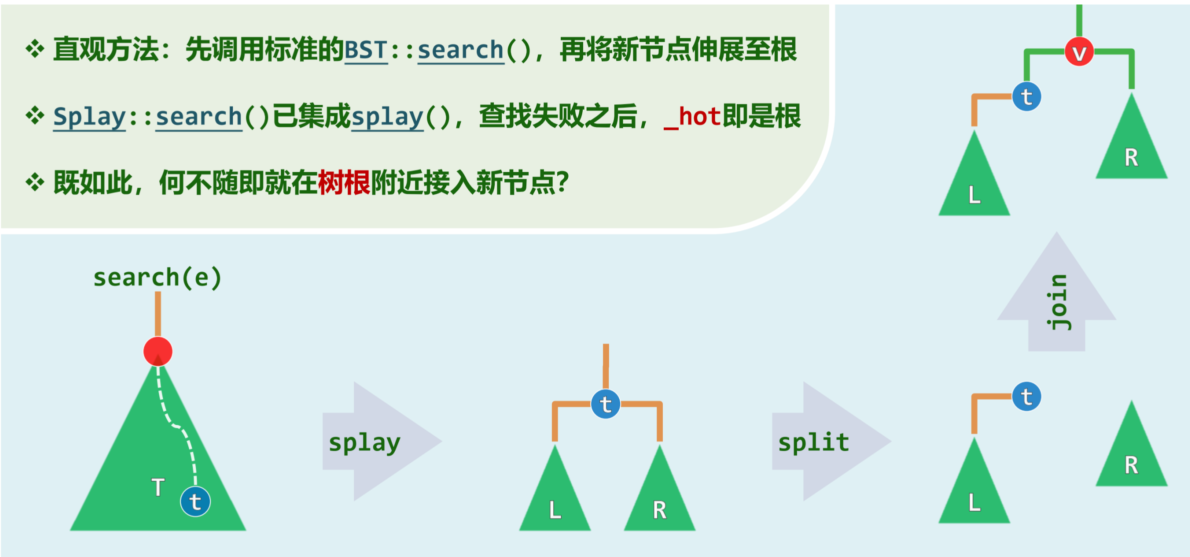
一字型先伸展祖父再伸展父亲，之字形先伸展父亲再伸展祖父，就和逐层伸展就没区别

搜索

将结点旋转至根

插入

先试图去找e，一定查找失败（不允许插入重复结点），将查找失败的点旋转至根，然后再将e直接作为根插入



删除

首先找到t，伸展至根，删除t，得到两棵子树，然后找到e的后继，再伸展至根部， 作为新的根

复杂度分析——势能分析

结论：均摊$log(n)$

S的势能

- （任何时刻的）任何一棵伸展树S，都可以**假想地**被认为具有势能：

$$\Phi(S) \;=\; \log\left(\prod_{v\in S} size(v)\right) \;=\; \sum_{v\in S}\log\left(size(v)\right) \;=\; \sum_{v\in S} rank(v) \;=\; \sum_{v\in S}\log V$$

- 直觉：越**平衡**/**倾侧**的树，势能越**小**/**大**

  - 单链：$\Phi(S) \;=\; \log n! \;=\; \mathcal{O}(n\log n)$

  - 满树：

$$\begin{aligned}\Phi(S) \;&=\; \log\prod_{d=0}^{h}\left(2^{h-d+1}-1\right)^{2^d} \;\le\; \log\prod_{d=0}^{h}\left(2^{h-d+1}\right)^{2^d}\\ &=\; \log\prod_{d=0}^{h} 2^{(h-d+1)\cdot 2^d} \;=\; \sum_{d=0}^{h}(h-d+1)\cdot 2^d \;=\; (h+1)\cdot\sum_{d=0}^{h}2^d-\sum_{d=0}^{h}d\cdot 2^d\\ &=\; (h+1)\cdot\left(2^{h+1}-1\right) \;-\; \left[(h-1)\cdot 2^{h+1} \;+\; 2\right] \;=\; 2^{h+2}-h-3 \;=\; \mathcal{O}(n)\end{aligned}$$

B-Tree

动机——缓存与大数据

结构——平衡多路搜索树

最大树高和最小树高

第k层结点个数n_k取值范围：$2\times\lceil m/2\rceil^{k-1}\;\le n_k\le\;2\times m^{k-1},\forall k\ge 0$

外部结点满足$N+1=n_h$，其中N为关键码数量

联立得$2\times\lceil m/2\rceil^{h-1}\;\le N+1=n_h\le\;m^h$

由此可以得到h的取值范围

$\Omega(log_m N)=\lceil log_m(N+1)\rceil\;\le h\;\le 1+\lfloor log_{\lceil m/2\rceil}\frac{N+1}{2}\rfloor=\mathrm{O}(log_m N)$

让B树达到最高树高的方法：顺序插入所有关键码，使所有除了左侧/右侧藤以外的子树都处于“稀疏状态”

操作

查找

每一个超级节点内部顺序查找，找到不大于的第一个时向下走。

插入

关键是上溢操作。当前节点插入后若溢出（即有m个关键码），则将第m/2个关键码提到父节点，其余部分分裂成两个子节点，若父节点继续溢出，则重复上述操作。若根节点溢出，则单开一层，B树高度增加。

删除

先跟直接后继交换，换到叶子结点。删除后若发生下溢，优先考虑借。若兄弟节点有富裕，则旋转交换，即父节点下来补，另一兄弟最大（最小）节点上去。

分裂和融合满足这样的关系：$n-h=s-m$，其中n为树结点个数，h为树高，s为分裂次数，m为融合次数。

平均而言，大致每经过$\lceil m/2 \rceil - 1$次插入，才会发生一次分裂（习题8-6b），不过这也只是一个下界

Red-Black Tree

动机——尽可能少地调整树的结构

并发性、锁等等

结构——(2,4)树

最大树高和最小树高

- 最大最小黑高度：转化为求（2，4）树的高度范围
- 最小高度：转化为求常规二叉搜索树最小高度
- 最大高度：h为偶，$h_{max}=2(\lfloor log_2(N+2) \rfloor - 1)$，h为奇，$h_{max}=2\lfloor log_2 \frac{N+2}{3}) \rfloor + 1$

操作

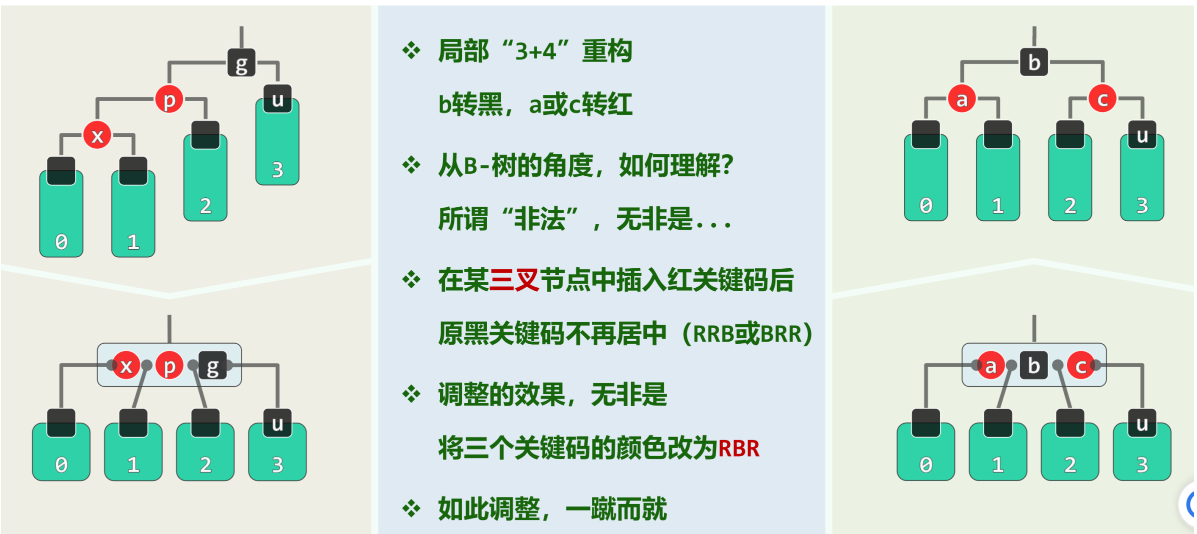
插入——双红修正

image-20230210100444881

如果uncle为黑，则旋转祖父和父亲，并改变其颜色

如果uncle为红，则不做旋转，改变祖父、父亲和叔叔结点的颜色，并上溢

因此应该至多$O(1)$次旋转和$O(logn)$次重染色



```
RR-2: u->color == R
```

删除——双黑修正

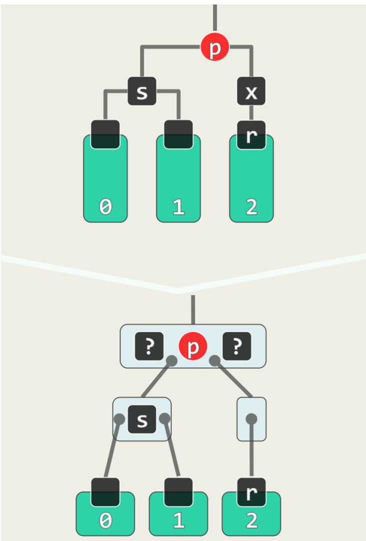
image-20230210100059305

首先先看兄弟是不是黑，如果兄弟是红，则其余结点都为黑，此时只需进行一次旋转和变色，则可以转换回兄弟为黑的情况。

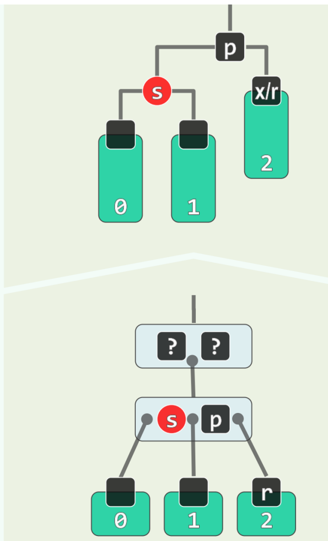
如果兄弟是黑，则看有没有红孩子，有红孩子则转换成b树下溢向兄弟“借”结点的情况，相当于对t,s,p做3+4重构并染色。

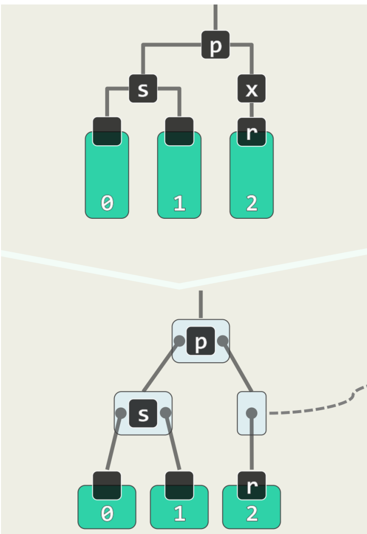
如果兄弟的孩子都是黑，则转换成下溢时向父亲“要"结点的情况，具体操作为将兄弟变红，父亲（如果是红）则变黑，不同之处在于如果父亲为红则上溢不会传递，父亲为黑则上溢将继续向上传递（相当于删除了父亲上面一个假象的黑结点）

因此也是至多$O(logn)$次染色和$O(1)$次旋转



- r保持黑；s转红；p转黑
- 在对应的B-树中，等效于
  下溢节点与兄弟合并
- 红黑树性质在**全局**得以恢复
- 失去关键码p后，上层节点
  会否**继而**下溢？不会！
- 合并之前，在p之左或右侧
  还应有一个黑关键码





- s转红；r与p保持黑
- 红黑树性质在**局部**得以恢复
- 在对应的B-树中，等效于
  下溢节点与兄弟合并
- 合并前，p和s均属于**单关键码**节点

孩子的下溢修复后，父节点继而下溢

- 好在可继续分情况处理
  高度递增，至多 $\mathcal{O}(\log n)$ 层（步）