黑客派

红黑树为什么比二叉查找树更高效

作者：zyjImmortal

原文链接：https://hacpai.com/article/1578230896592

来源网站：黑客派

<img src="https://static.hacpai.com/images/img-loading.svg" alt="" data-src="https://img.hacpai.com/bing/20180622.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
<h2 id="平衡二叉查找树">平衡二叉查找树</h2>
<h2 id="什么是平衡二叉查找树">什么是平衡二叉查找树</h2>
二叉树中任意一个节点的左右子树的高度相差不能大于 1，这是一种较为严格的定义，但是实际程中使用，不会要求这么严格，只要实际的高度不比 log2(n)大很多，能达到平衡的效果，我们就认它是一棵合格的平衡二叉树。
<h2 id="平衡二叉树要解决的问题">平衡二叉树要解决的问题</h2>
二叉查找树在理想状态下查找一个节点的时间复杂度是 log(n)也就是树的高度，但是二叉查找树频繁的动态更新过程中，只在二叉树的一个子树上进行数据的插入，也就可能会出现树的高度远大于 lg2n 的情况，从而导致各个操作的效率下降。极端情况下，二叉树会退化为链表，时间复杂度会退化 O(n)。
平衡二叉树是要解决普通二叉查找树在频繁的插入、删除等动态更新的情况下，出现时间复杂度化的问题。所以平衡的意思就是整棵树左右看起来比较对称，比较平衡，不要出现左子树很高、右子很矮的情况。这样就能让整棵树的高度相对来说低一些，相应的插入、删除、查找等操作的效率高一。
<h2 id="红黑树">红黑树</h2>
平衡二叉查找树有很多实现，比如，Splay Tree（伸展树）、Treap（树堆）等，但是在实际应中红黑树出镜率最高。
<h2 id="什么是红黑树">什么是红黑树</h2>
红黑树中的节点，一类被标记为黑色，一类被标记为红色。是一种不严格的平衡二叉查找树
<h2 id="红黑树都有哪些要求">红黑树都有哪些要求</h2>

根节点是黑色的；
每个叶子节点都是黑色的空节点（NIL），也就是说，叶子节点不存储数据；
任何相邻的节点都不能同时为红色，也就是说，红色节点是被黑色节点隔开的；
每个节点，从该节点到达其可达叶子节点的所有路径，都包含相同数目的黑色节点；

<h2 id="为什么说红黑树是不严格的平衡">为什么说红黑树是不严格的平衡</h2>
平衡二叉查找树的初衷，是为了解决二叉查找树因为动态更新导致的性能退化问题。所以，“平”的意思可以等价为性能不退化。“近似平衡”就等价为性能不会退化的太严重。
 二叉查找树查找、插入、删除操作的性能都跟树的高度成正比，一个及其平衡的二叉树的高度大约是 log2(n)，以证明红黑树是近似平衡，只要证明红黑树的高度比较稳定的接近 log2(n)就行。
 <img src="https://static.hacpai.com/images/img-loading.svg" alt="红黑树近似分析.png" data-src="https://bog.immortalp.com/upload/2020/1/%E7%BA%A2%E9%BB%91%E6%A0%91%E8%BF%91%E4%BC%BC%E5%88%86%E6%9E%90-7502e9e2f79f44348511508a1df480d4.png">
 如图，将黑树的红色节点全部去掉，有些节点就没有父节点了，会直接以祖父节点作为父节点，就会变成一个叉树。
 面红黑树的定义里有这么一条：从任意节点到可达的叶子节点的每个路径包含相同数目黑色节点。我们从四叉树中取出某些节点，放到叶节点位置，四叉树就变成了完全二叉树。所以，仅含黑色节点的四叉树的高度，比包含相同节点个数的完全二叉树的高度还要小。
 完全二叉树的度近似 log2(n)，这里的四叉“黑树”的高度要低于完全二叉树，所以去掉红色节点的“黑树”的高也不会超过 og2(n)。
 然后把红色节点加进去，红黑树中最长黑色节点的路径不会超过 log2(n)如果在这条路径上按照要求，每隔一个黑色节点就要有一个红色节点，那最长路径也不过是 2log2(n), strong>所以红黑树的高度之比高度平衡的平衡二叉树多出一倍去，在性能上下降不多</>
<h2 id="为什么近似平衡的红黑树会被广泛使用">为什么近似平衡的红黑树会被广泛使用</h2>
AVL 树是一种高度平衡的二叉树，所以查找的效率非常高，但是，有利就有弊，AVL 树为了维这种高度的平衡，就要付出更多的代价。每次插入、删除都要做调整，就比较复杂、耗时。所以，对有频繁的插入、删除操作的数据集合，使用 AVL 树的代价就有点高了。
红黑树只是做到了近似平衡，并不是严格的平衡，所以在维护平衡的成本上，要比 AVL 树要低所以，红黑树的插入、删除、查找各种操作性能都比较稳定。对于工程应用来说，要面对各种异常情，为了支撑这种工业级的应用，我们更倾向于这种性能稳定的平衡二叉查找树。
<h2 id="总结">总结</h2>
红黑树是一种平衡二叉查找树。它是为了解决普通二叉查找树在数据更新的过程中，复杂度退化

问题而产生的。红黑树的高度近似 log2n，所以它是近似平衡，插入、删除、查找操作的时间复杂度是 O(logn)。
因为红黑树是一种性能非常稳定的二叉查找树，所以，在工程中，但凡是用到动态插入、删除、找数据的场景，都可以用到它。不过，它实现起来比较复杂，如果自己写代码实现，难度会有些高，个时候，我们其实更倾向用跳表来替代它
<h2 id="拓展-动态数据结构">拓展：动态数据结构</h2>
动态数据结构就是动态的更新操作，里面存储的数据是时刻在变化的，不仅仅支持查询还支持插、删除数据，而且这些操作都是非常高效的，像红黑树、散列表、跳表都是动态数据结构，链表、队、栈实际上不算是，因为操作非常有限，查询效率不高。

 散列表：插入删除查找都是 O(1), 是最常用的，但其缺点是不能顺序遍历以及扩容缩容的能损耗。适用于那些不需要顺序遍历，数据更新不那么频繁的。
 跳表：插入删除查找都是 O(logn), 并且能顺序遍历。缺点是空间复杂度 O(n)。适用于不么在意内存空间的，其顺序遍历和区间查找非常方便。
 红黑树：插入删除查找都是 O(logn), 中序遍历即是顺序遍历，稳定。缺点是难以实现，去找不方便。其实跳表更佳，但红黑树已经用于很多地方了
