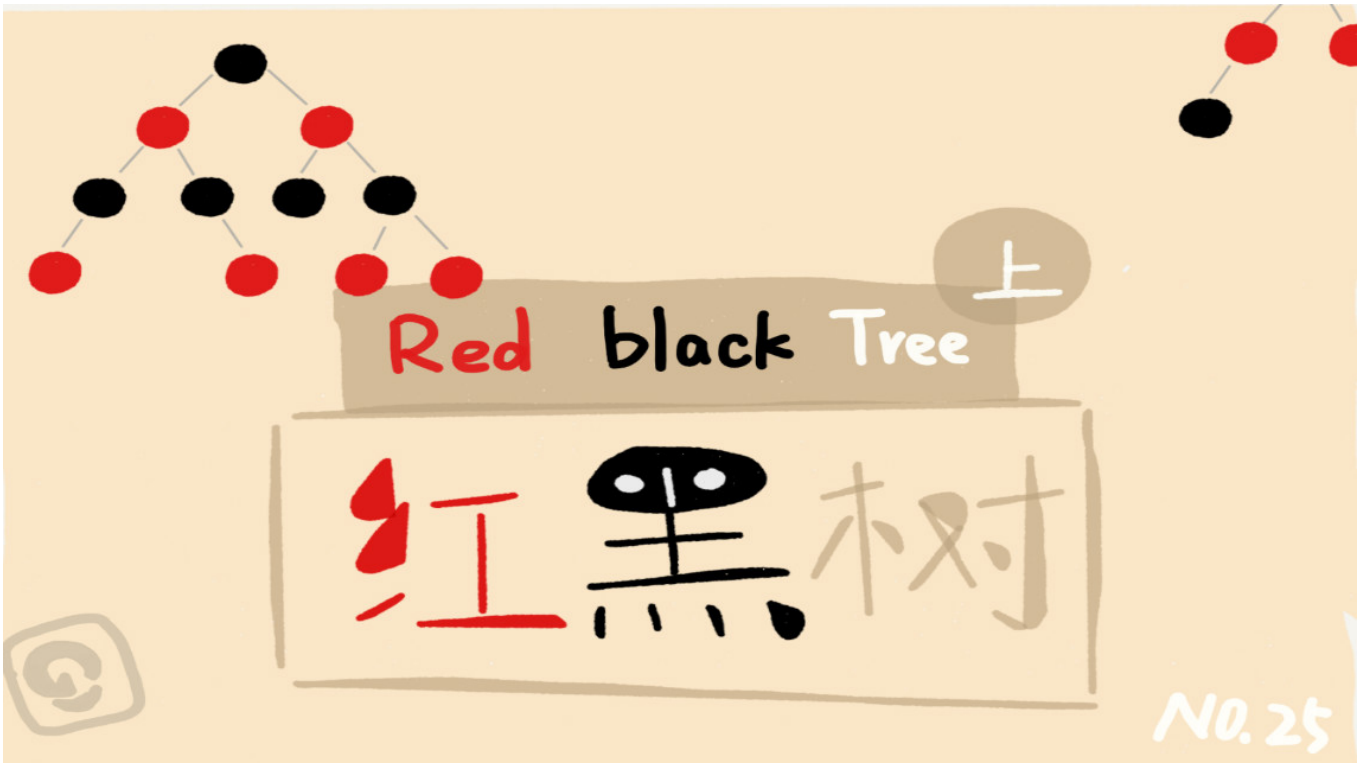
25 | 红黑树（上）：为什么工程中都用红黑树这种二叉树?

2018-11-16 王争



25 | 红黑树（上）：为什么工程中都用红黑树这种二叉树?
朗读人：修阳　10′00″ | 4.59M

上两节，我们依次讲了树、二叉树、二叉查找树。二叉查找树是最常用的一种二叉树，它支持快速插入、删除、查找操作，各个操作的时间复杂度跟树的高度成正比，理想情况下，时间复杂度是 O(logn)。

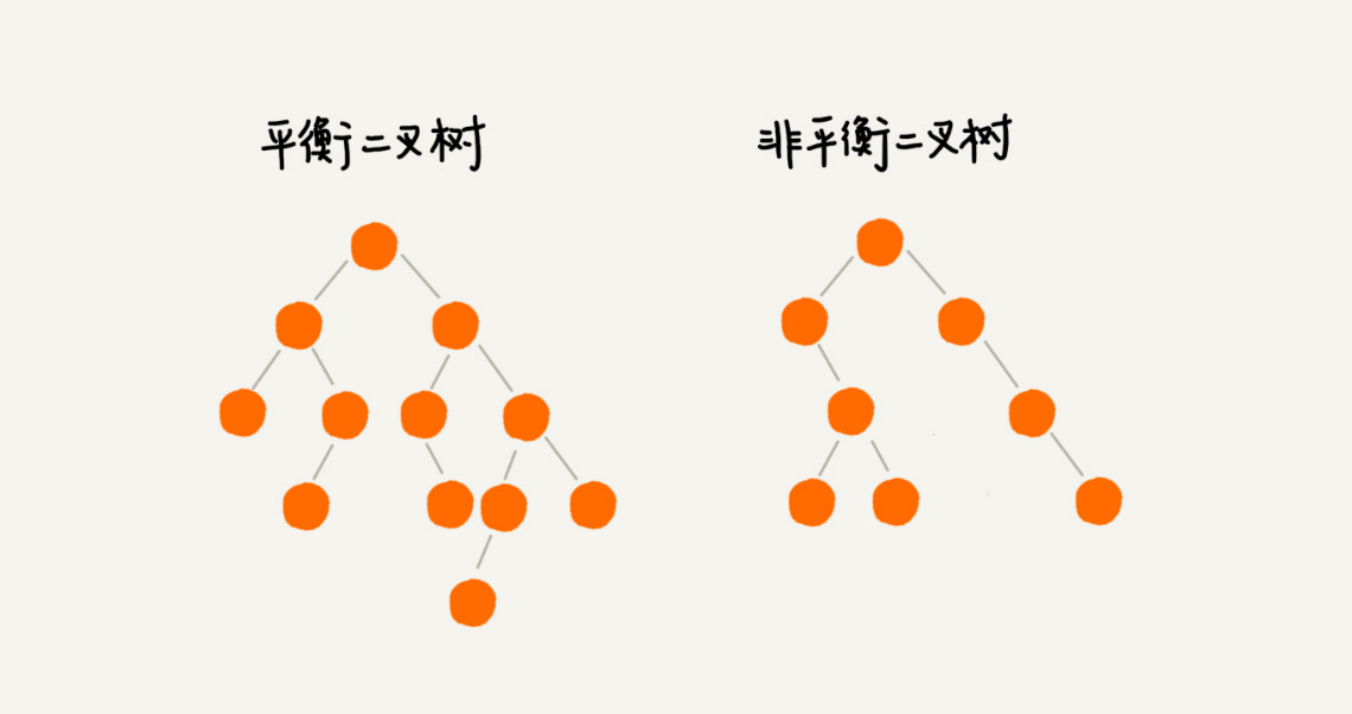
不过，二叉查找树在频繁的动态更新过程中，可能会出现树的高度远大于 $\log_2 n$ 的情况，从而导致各个操作的效率下降。极端情况下，二叉树会退化为链表，时间复杂度会退化到 O(n)。我上一节说了，要解决这个复杂度退化的问题，我们需要设计一种平衡二叉查找树，也就是今天要讲的这种数据结构。

很多书籍里，但凡讲到平衡二叉查找树，就会拿红黑树作为例子。不仅如此，如果你有一定的开发经验，你会发现，在工程中，很多用到平衡二叉查找树的地方都会用红黑树。你有没有想过，**为什么工程中都喜欢用红黑树，而不是其他平衡二叉查找树呢?**

带着这个问题，让我们一起来学习今天的内容吧！

什么是“平衡二叉查找树”？

平衡二叉树的严格定义是这样的：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。从这个定义来看，上一节我们讲的完全二叉树、满二叉树其实都是平衡二叉树，但是非完全二叉树也有可能是平衡二叉树。



平衡二叉查找树不仅满足上面平衡二叉树的定义，还满足二叉查找树的特点。最先被发明的平衡二叉查找树是AVL 树，它严格符合我刚讲到的平衡二叉查找树的定义，即任何节点的左右子树高度相差不超过 1，是一种高度平衡的二叉查找树。

但是很多平衡二叉查找树其实并没有严格符合上面的定义（树中任意一个节点的左右子树的高度相差不能大于 1），比如我们下面要讲的红黑树，它从根节点到各个叶子节点的最长路径，有可能会比最短路径大一倍。

我们学习数据结构和算法是为了应用到实际的开发中的，所以，我觉得没必去死抠定义。对于平衡二叉查找树这个概念，我觉得我们要从这个数据结构的由来，去理解“平衡”的意思。

发明平衡二叉查找树这类数据结构的初衷是，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，出现时间复杂度退化的问题。

所以，**平衡二叉查找树中“平衡”的意思，其实就是让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、右子树很矮的情况。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些。**

所以，如果我们现在设计一个新的平衡二叉查找树，只要树的高度不比 $\log_2 n$ 大很多（比如树的高度仍然是对数量级的），尽管它不符合我们前面讲的严格的平衡二叉查找树的定义，但我们仍然可以说，这是一个合格的平衡二叉查找树。

如何定义一棵“红黑树”?

平衡二叉查找树其实有很多，比如，Splay Tree（伸展树）、Treap（树堆）等，但是我们提到平衡二叉查找树，听到的基本都是红黑树。它的出镜率甚至要高于“平衡二叉查找树”这几个字，有时候，我们甚至默认平衡二叉查找树就是红黑树，那我们现在就来看看这个“明星树”。

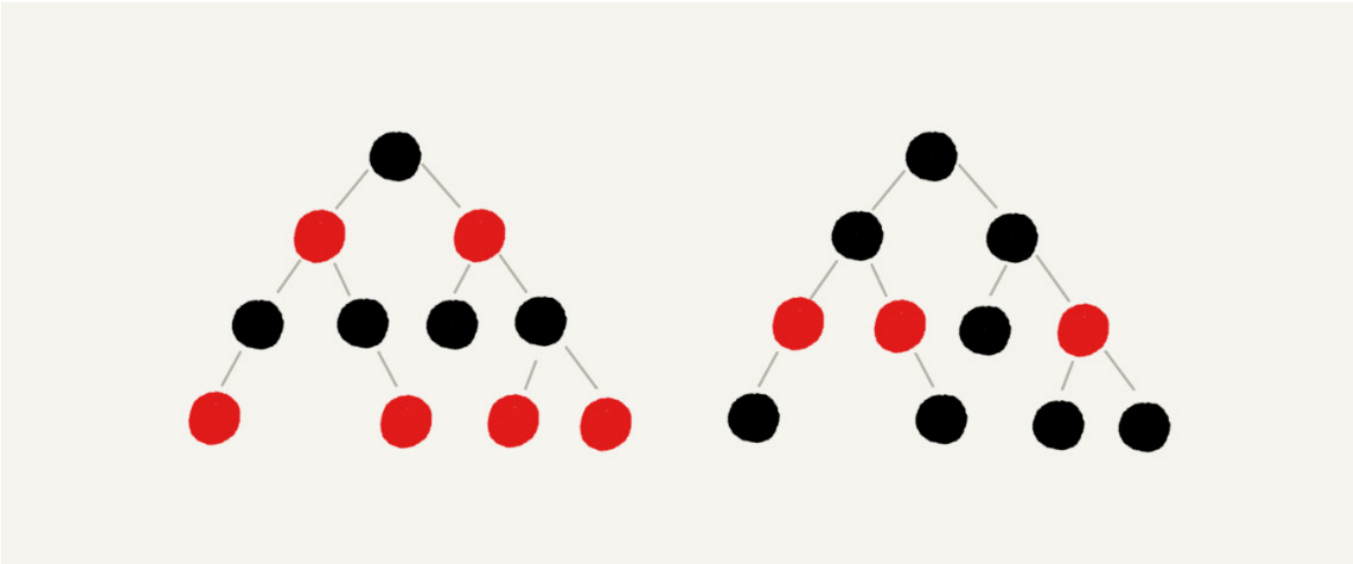
红黑树的英文是“Red-Black Tree”，简称 R-B Tree。它是一种不严格的平衡二叉查找树，我前面说了，它的定义是不严格符合平衡二叉查找树的定义的。那红黑树究竟是怎么定义的呢?

顾名思义，红黑树中的节点，一类被标记为黑色，一类被标记为红色。除此之外，一棵红黑树还需要满足这样几个要求：

- 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；

这里的第二点要求“叶子节点都是黑色的空节点”，稍微有些奇怪，它主要是为了简化红黑树的代码实现而设置的，下一节我们讲红黑树的实现的时候会讲到。**这节我们暂时不考虑这一点，所以，在画图和讲解的时候，我将黑色的、空的叶子节点都省略掉了。**

为了让你更好地理解上面的定义，我画了两个红黑树的图例，你可以对照着看下。



为什么说红黑树是“近似平衡”的?

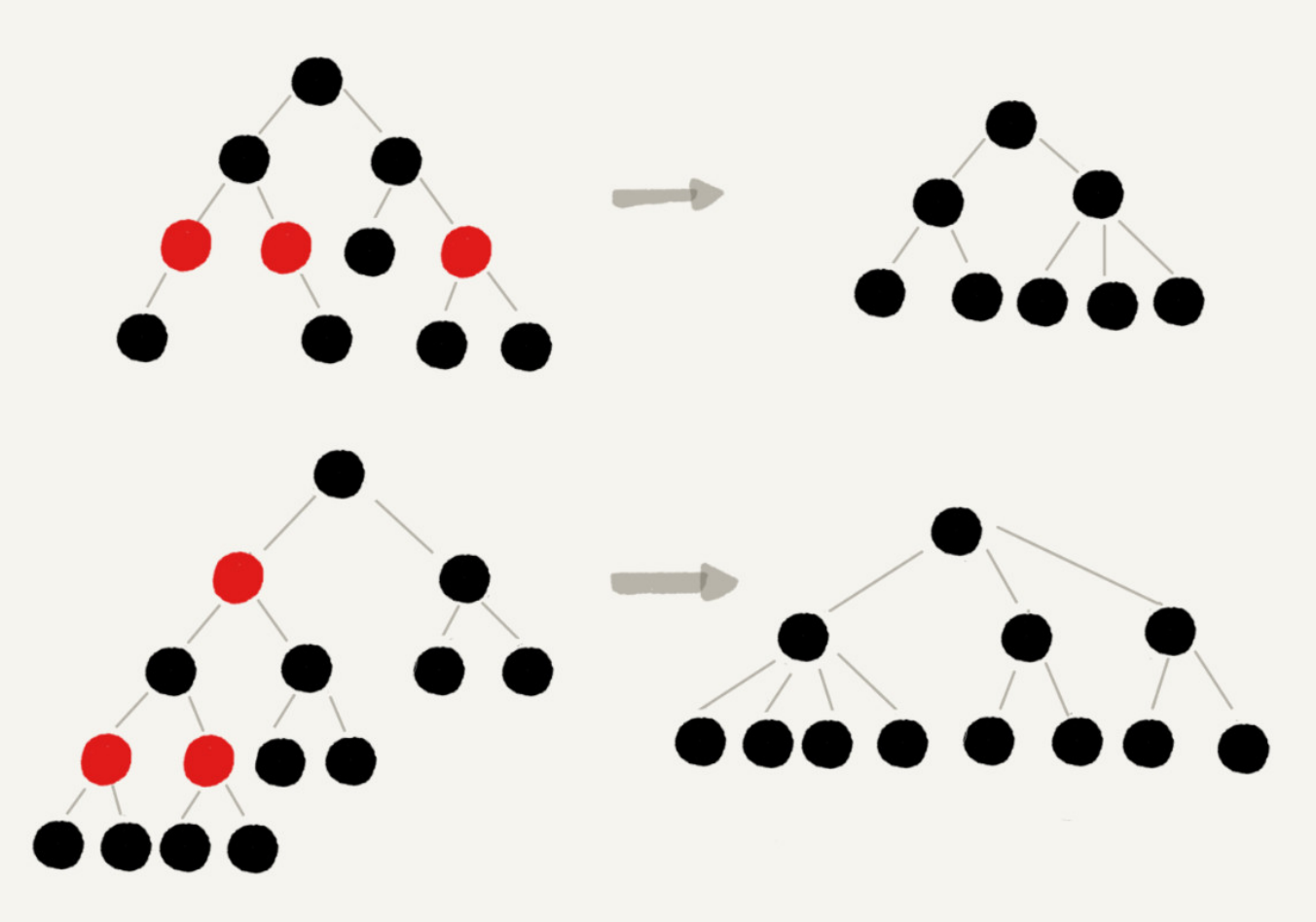
我们前面也讲到，平衡二叉查找树的初衷，是为了解决二叉查找树因为动态更新导致的性能退化问题。所以，**“平衡”的意思可以等价为性能不退化。“近似平衡”就等价为性能不会退化的太严重。**

我们在上一节讲过，二叉查找树很多操作的性能都跟树的高度成正比。一棵极其平衡的二叉树（满二叉树或完全二叉树）的高度大约是 $\log_2 n$，所以如果要证明红黑树是近似平衡的，我们只需要分析，红黑树的高度是否比较稳定地趋近 $\log_2 n$ 就好了。

红黑树的高度不是很好分析，我带你一步一步来推导。

首先，我们来看，如果我们将红色节点从红黑树中去掉，那单纯包含黑色节点的红黑树的高度是多少呢？

红色节点删除之后，有些节点就没有父节点了，它们会直接拿这些节点的祖父节点（父节点的父节点）作为父节点。所以，之前的二叉树就变成了四叉树。



前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的黑色节点。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，仅包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。

上一节我们说，完全二叉树的高度近似 $\log_2 n$，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”的高度也不会超过 $\log_2 n$。

我们现在知道只包含黑色节点的“黑树”的高度，那我们现在把红色节点加回去，高度会变成多少呢？

从上面我画的红黑树的例子和定义看，在红黑树中，红色节点不能相邻，也就是说，有一个红色节点就要至少有一个黑色节点，将它跟其他红色节点隔开。红黑树中包含最多黑色节点的路径不会超过 $\log_2 n$，所以加入红色节点之后，最长路径不会超过 $2\log_2 n$，也就是说，红黑树的高度近似 $2\log_2 n$。

所以，红黑树的高度只比高度平衡的 AVL 树的高度（$\log_2 n$）仅仅大了一倍，在性能上，下降得并不多。这样推导出来的结果不够精确，实际上红黑树的性能更好。

解答开篇

我们刚刚提到了很多平衡二叉查找树，现在我们就来看下，为什么在工程中大家都喜欢用红黑树这种平衡二叉查找树?

我们前面提到 Treap、Splay Tree，绝大部分情况下，它们操作的效率都很高，但是也无法避免极端情况下时间复杂度的退化。尽管这种情况出现的概率不大，但是对于单次操作时间非常敏感的场景来说，它们并不适用。

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 树为了维持这种高度的平衡，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，对于有频繁的插入、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，要面对各种异常情况，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。

内容小结

很多同学都觉得红黑树很难，的确，它算是最难掌握的一种数据结构。其实红黑树最难的地方是它的实现，我们今天还没有涉及，下一节我会专门来讲。

不过呢，我认为，我们其实不应该把学习的侧重点，放到它的实现上。那你可能要问了，关于红黑树，我们究竟需要掌握哪些东西呢?

还记得我多次说过的观点吗？**我们学习数据结构和算法，要学习它的由来、特性、适用的场景以及它能解决的问题。对于红黑树，也不例外。你如果能搞懂这几个问题，其实就已经足够了。**

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，复杂度退化的问题而产生的。红黑树的高度近似 $\log_2 n$，所以它是近似平衡，插入、删除、查找操作的时间复杂度都是 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、查找数据的场景，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，这个时候，我们其实更倾向用跳表来替代它。

课后思考

动态数据结构支持动态地数据插入、删除、查找操作，除了红黑树，我们前面还学习过哪些呢？能对比一下各自的优势、劣势，以及应用场景吗？

欢迎留言和我分享，我会第一时间给你反馈。

©版权归极客邦科技所有，未经许可不得转载

上一篇 24 | 二叉树基础（下）：有了如此高效的散列表，为什么还需要二叉树？

写留言

精选留言

Smallfly 7

动态数据结构的动态是什么意思？

动态是指在运行时该数据结构所占的内存会扩大或缩小。

数组是一种静态的数据结构，在创建时内存大小已经确定，不管有没有插入数据。而链表是动

态的数据结构，插入数据 alloc 内存，删除数据 release 内存。

栈、队列、散列表、跳表、树都是抽象的数据结构，它们在内存中存在的形式都要依赖于数组或者链表。

如果用链表实现，很明显它们是动态的数据结构；如果用数组实现，那么在扩容的时候会创建更大内存的数组，原数组被废弃，从抽象角度看，它们仍旧是动态的。

2018-11-16

作者回复

我看smallfly大牛好像对动态数据结构有些误解，可能其他同学也会有。所以，我解释一下：动态数据结构是支持动态的更新操作，里面存储的数据是时刻在变化的，通俗一点讲，它不仅仅支持查询，还支持删除、插入数据。而且，这些操作都非常高效。如果不高效，也就算不上是有效的动态数据结构了。所以，这里的红黑树算一个，支持动态的插入、删除、查找，而且效率都很高。链表、队列、栈实际上算不上，因为操作非常有限，查询效率不高。那现在你再想一下还有哪些支持动态插入、删除、查找数据并且效率都很高的的数据结构呢？？

2018-11-16

失火的夏天 4

动态数据结构有链表，栈，队列，哈希表等等。链表适合遍历的场景，插入和删除操作方便，栈和队列可以算一种特殊的链表，分别适用先进后出和先进先出的场景。哈希表适合插入和删除比较少（尽量少的扩容和缩容），查找比较多的时候。红黑树对数据要求有序，对数据增删查都有一定要求的时候。（个人看法，欢迎老师指正）

2018-11-16

作者回复

👍 刚看错了。写的不错

2018-11-16

🐱您的好友William🐱 3

老师做图的软件用的是什么啊？我看了不少的极客时间的blog，感觉老师这个是最好看的哈哈。

2018-11-16

null 2

红黑树的节点颜色，是如何确定的，如何知道在新增一个节点时，该节点是什么颜色？

从红黑树需要满足的四个要求来看：
1. 根节点为黑色
2. 所有叶子节点为黑色，且不存储数据
3. 相邻两个节点不能都为红色
4. 从某节点到其所有叶子节点的路径中，黑色节点数相同

从这四点要求，好像我一根树全是黑色，也是满足其定义的。这里用红黑两色区分各节点，意

义是啥?

谢谢老师

2018-11-16

作者回复

新插入的节点都是红色的。全黑不可能的。红黑区分的意义你等下一节课看看能懂不

2018-11-16

朱月俊 1

动态数据结构比如本篇的平衡二叉查找树，还有就是跳表，跳表也支持动态插入，删除，查询，也很快，不同点是跳表还能支持快速的范围查询。比如leveldb中的memtable，redis都是使用跳表实现的，而也有用红黑树实现的memtable。
除此之外，跳表还支持多写多读，而红黑树不可以，这些场景下显然用跳表合适。

2018-11-17

gg 1

我看smallfly大牛好像对动态数据结构有些误解，可能其他同学也会有。所以，我解释一下：动态数据结构是支持动态的更新操作，里面存储的数据是时刻在变化的，通俗一点讲，它不仅仅支持查询，还支持删除、插入数据。而且，这些操作都非常高效。如果不高效，也就算不上是有效的动态数据结构了。所以，这里的红黑树算一个，支持动态的插入、删除、查找，而且效率都很高。链表、队列、栈实际上算不上，因为操作非常有限，查询效率不高。那现在你再想一下还有哪些支持动态插入、删除、查找数据并且效率都很高的的数据结构呢？？

2018-11-16

Farflight 1

老师好，请问之后讲到heap sort这些的时候会提到斐波那契堆吗？最近在看相关但是觉得网上的资料讲得都不太清楚。

2018-11-16

Geek_022a0f 0

老师，可以教我们刷下leetcode上的算法，毕竟讲了这么多，还是练习的。这样的提升才有巨大的帮助

2018-11-18

insist 0

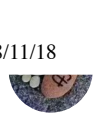
支持高效的查找、插入、删除操作的动态数据结构有跳表、散列表、平衡二叉查找树

2018-11-18

阿K 0

以前只知道概念和实现，不知道为什么这么实现。
要是早几年看到这篇文章， 我也不至于这样了

2018-11-17



0

虽然不能理解其中的精华，可能是编程的时间很短，平时也没有考虑数据结构，但是，每一节都认真看。

2018-11-17

城

0

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；这个是指父子之间的相邻吧，很容易误会成兄弟间的关系

2018-11-17

sketch2018

0

讲到的动态数据结构除了红黑树外还有HashMap和跳表;
其中HashMap查找时间复杂度为O(1),但是其扩容耗时且负载因子较大时哈希冲突概率很高会导致时间复杂度退化;
跳表是在链表的基础上建立索引提高效率(时间复杂度logn)，需要消耗多余的空间存储索引

2018-11-16

iron_man

0

跳表相对于红黑树，查询效率是同一个量级，但插入数据，红黑树更高效，旋转的节点不多，而跳表可能要移动比较多的数据?

2018-11-16

绪扬IS未知数

0

讲红黑树入门不说到二三四树都是耍流氓。
什么节点是黑，什么节点是红?
我弄一个无序的二叉树，不管里面的数字是什么，然后给它们涂上颜色，照样可以符合规定。

2018-11-16

Smallfly

0

非常感谢老师的指点，我不是大牛，我是渣渣。。

按照老师对高效动态数据结构的定义，专栏目前讲到的动态数据结构，除了红黑树，还有散列表和跳表，没有涉及到的有 B 树、树状数组等……

2018-11-16

Ricky

0

散列表和跳表应该属于动态数据结构吧，它们都支持动态更新，插入和删除操作，跳表应该跟平衡二叉查找树差不多，只是实现比较简单，比较适合大规模数据，排序的场景，散列表应该更多适于数据查询场景，如果涉及到排序，可能需要借助跳表实现

2018-11-16

有铭

0

我除了看懂红黑树是一种“性能上比较均衡”的二叉树这个结论外，完全没搞懂它为啥能获得这个“比较平衡”的结果

2018-11-16

wean 0

“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”

老师，这里的某些节点应该怎么取，才能让四叉树变成二叉树，不是很明白，希望老师讲解一下，谢谢

2018-11-16

ALAN 0

老师，红黑树删除两个红节点之间的黑节点后，应该如何处理?

2018-11-16

作者回复

下一节课会讲

2018-11-16

跬行 0

红色节点删除下面的图片(上下两部分，下部分)是不是有一个错误，有四个连续的黑结点

2018-11-16

作者回复

是的 多谢指出 马上改正

2018-11-16

往事随风，顺其自然 0

这里面的从该节点到可达的叶子节点，可达指的是可以回溯?

2018-11-16

作者回复

可达就是字面意思 可以到达。因为并不是所有的叶子节点它都能到达。所以这里加了可达两个字

2018-11-16