用停用词表(stop list)来忽略普通的词;同时,对英文词进行抽词干(stemming)处理,对中文进行切词(chinese segmentation)处理;最后,采用 hash 数值函数产生签名位串。保留词在文档中的位置信息,将每个词的签名串联起来就得到该文档的签名,这样可以产生无重叠编码的签名文件。

此外,人们还提出了具有更高检索速度的二级签名文件、签名树以及基于签名的分割等。

签名文件方法的优点在于实现简单,插入操作效率高,能够处理词的部分查询,能够支持不断增长的文件以及拼写错误的容错性,而且该方法易于并行化。其主要缺点就是对大文件的操作时间较长,不易处理日益增长的词汇表;签名集合的设计需要巨大的开销。

11.6 红 黑 树

红黑树(red - black tree)是一种扩充的二叉搜索树 BST,树中的每一个结点的颜色是黑色或红色。它利用对树中结点红黑着色的要求达到局部平衡,其插入和删除算法性能好、易实现,同时检索效率也比较高。

11.6.1 红黑树的定义

满足下列条件的二叉搜索树是红黑树:

(1) 每个结点要么是红色,要么是黑色(后面将对颜色进行说明)。

(2) 根结点永远是黑色的。

(3) 所有的扩充外部叶结点都是空结点,并且是黑色的。

(4) 如果一个结点是红色的,那么它的两个子结点都是黑色的(不允许两个连续的红色结点)。

(5) 结点到其子孙外部结点的每条简单路径都包含相同数目的黑色结点。

结点 X 的阶(rank,也称"黑色高度"):从该结点到其子树中任意外部结点的任一条路径上的黑色结点数量(不包括 X 结点,包括叶结点)。外部结点的阶是零,根的阶称为该树的阶。

在图11.16所示的红黑树中,黑方块是外部结点,黑圈是黑色结点(粗线是指向黑色结点的

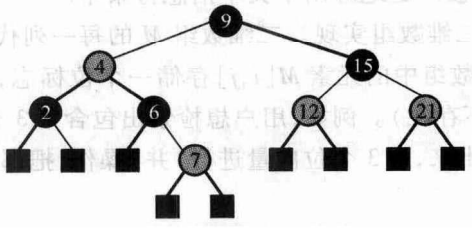

图 11.16 红黑树示意图

指针),浅色圈代表红色结点(细线是指向红色结点的指针)。该树中没有双红结点,根结点是黑色的。任何一条根结点到外部结点的简单路径上,都有两个黑结点,因此该树的阶为 2。根的左子结点 4 的阶是 2,右子结点 15 的阶是 1。

11.6.2 红黑树的相关性质

红黑树具有以下性质:

(1) 红黑树是满二叉树(空叶结点也看做结点)。

(2) 阶为 n 的红黑树,从根结点到叶结点的简单路径最短长度是 n,最长长度是 $2n$;或者说该树的树高(即根到最深的叶结点所经过的结点数)最小是 $n+1$,最大是 $2n+1$。

(3) 阶为 n 的红黑树,其含有的内部结点最少时是一棵完全满二叉树,此时内部结点数是 2^n-1。

(4) 含有 n 个内部结点的红黑树树高最大是 $2\log_2(n+1)+1$。

性质(1)从定义即可得出。

性质(2)容易证明:根据红黑树定义条件(4),结点到其子孙结点的每条简单路径中不可能有两个连续的红色结点。根据定义条件(5),最短的简单路径中全是黑色结点,此时路径长度是 n;由于最长的简单路径是红黑交替的,且根结点和叶结点必是黑色的,因此最多有 n 个红色结点,$n+1$ 个黑色结点(含根结点),路径长度是 $2n$。

性质(3)证明:当红黑树含有的内部结点最少时,则均为黑色结点;根据定义条件(4)和性质(1)易知,此时红黑树是一棵完全满二叉树,且树高是 n;又根据完全满二叉树性质可知,所有结点总数是 $2^{n+1}-1$,叶结点数是内部结点数加 1,可算得内部结点数是 2^n-1。

性质(4)证明:设红黑树的阶为 x,红黑树的树高是 h,内部结点数是 n。由性质(2)得 $h\leqslant 2x+1$,则 $x\geqslant(h-1)/2$;由性质(3)得 $n\geqslant 2^x-1$;从而得 $n\geqslant 2^{(h-1)/2}-1$,可得出 $h\leqslant 2\log_2(n+1)+1$。

虽然在定义红黑树时,将外部结点包括进来非常方便,但在执行过程中往往用空指针而不用物理结点来描述这些外部结点。由于指针颜色与结点颜色是紧密联系的,因此对于每个结点,需要存储的只是该结点的颜色或指向其子结点的指针颜色。

可以使用对普通 BST 的检索算法来完成对红黑树的检索,由性质(4)可知其检索长度为 $O(\log_2 n)$。

11.6.3 插入结点算法

与二叉树检索树(BST)的插入算法类似,红黑树中插入新记录时,首先检索到该记录所在的位置,时间代价为 $O(\log_2 n)$。然后,插入记录,把记录着色为红色。不同的是,红黑树还要检查

插入记录是否影响树的平衡。如果新增结点的父结点是黑色，则算法结束；否则，根据红黑树定义条件(3)。此时，红黑树不平衡，需要调整，调整方法分情况讨论如下：

情况1：新增结点的叔父结点是黑色。如图11.17所示，设新增结点是X(红色)，父结点是A(红色)，祖父结点是B(黑色)，叔父结点是C(黑色)。

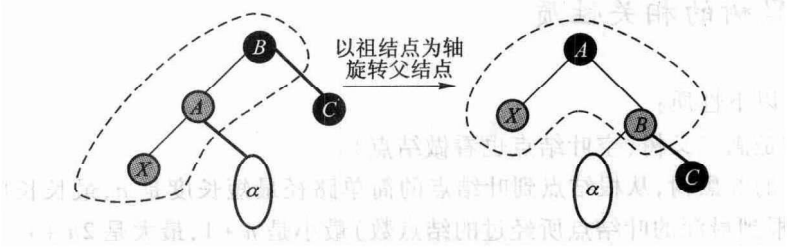

图11.17 父结点是红色，叔父结点是黑色的情况，旋转解决红红冲突

注意：图11.17中新增结点X是A结点的左子结点。若X是A的右子结点，可以把X结点和其父结点A换位，则A结点就成为X结点的左子结点，此时即可类似地处理，只是X和A符号换位而已。

在图11.17中，A结点是父结点的左子结点，如果是右子结点，则需要向左旋转。方法类似，但此时需要保证X结点是A结点的右子结点，而不是左子结点。

实际上，有4种子结构需要调整，如图11.18所示。其实质是取双红结点、祖父结点这三者的中位数作为新的子根结点，并且着色为黑色，子根的两个子结点都着色为红色。每个结点的阶都保持原值，调整完成。图11.17所示为图11.18 LL型的例子。

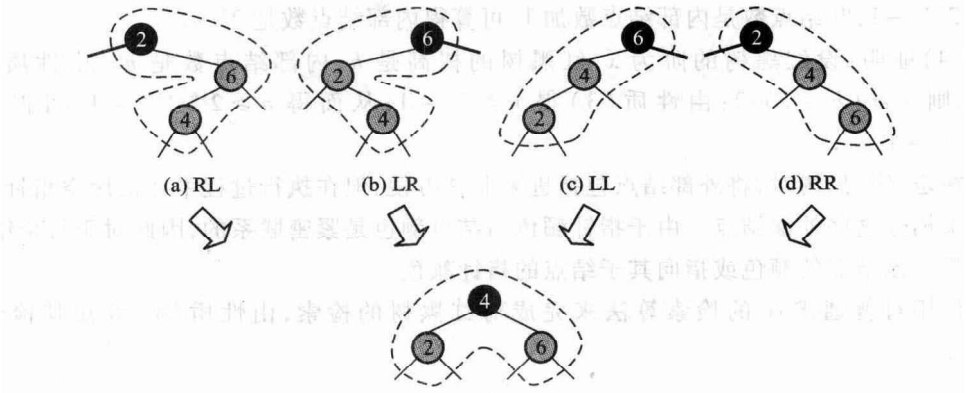

图11.18 叔父结点为黑色，进行重构调整每个结点的阶都保持原值，调整完成

情况2：新增结点的叔父结点也是红色。如图11.19所示，设新增结点是X(红色)，父结点是A(红色)，祖父结点是B(黑色)，叔父结点是C(红色)。其中，X是A的左子结点或右子结点对这种情况没有影响。

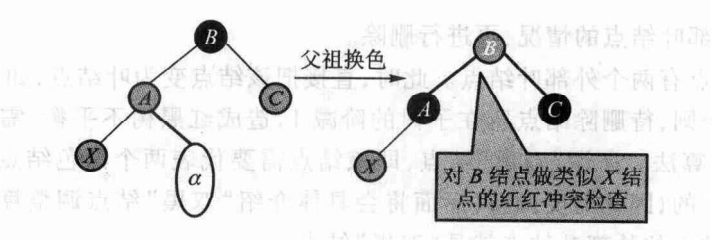

图 11.19 父结点、叔父结点均为红色，则父和祖换色，然后递归处理方法

此时，把 X 的父结点 A 和叔父结点 C 都变成黑色，其祖父结点 B 变成红色。这时只是解决了 A 结点以下的红红冲突，但有可能会引起 B 结点与其父结点的冲突，因此要像处理 X 结点一样，对 B 结点进行递归调整。最糟糕的情况会引起根结点变成红色，此时只需把根结点的着色改成黑色即可，但会引起树的阶增加 1。

图 11.20 所示为在红黑树中插入一个关键码之后引起的红红冲突调整示例。

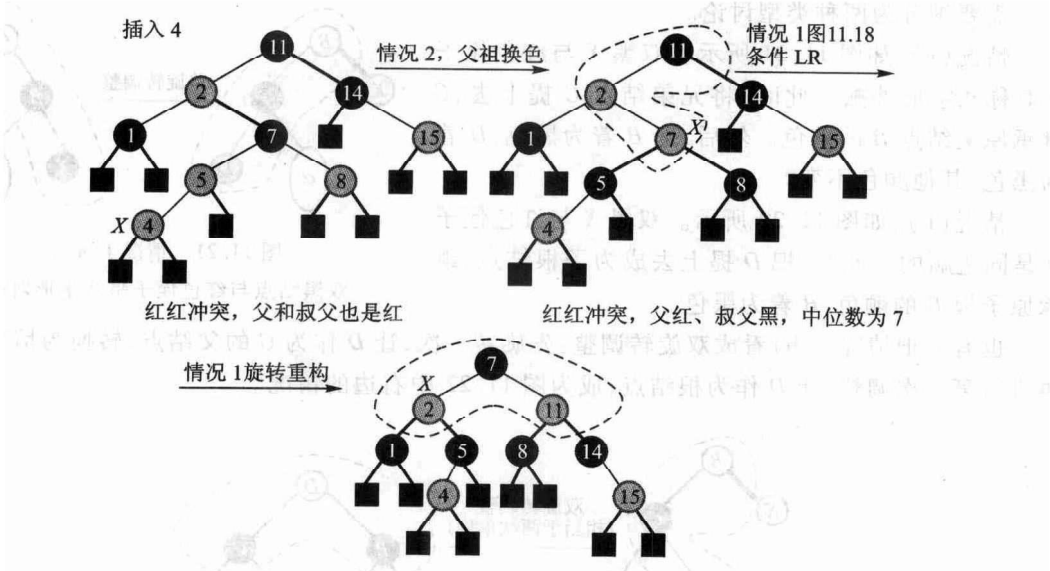

图 11.20 红黑树插入示例

11.6.4 删除结点算法

红黑树的删除操作比插入操作复杂一些，基本删除算法与 BST 类似，但需要检查红黑平衡性。首先，找到待删除的结点，如果该结点有一个以上的空指针（有一个或两个外部叶结点），则直接删除；如果有两个非叶子结点，则在右子树中找到最小值结点（其后继结点）与该结点进行值交换（结点位置的着色不变），然后再对交换后的位置调用删除结点算法，直到转换位置成为

具有一个或两个外部叶结点的情况，再进行删除。

（1）待删除结点有两个外部叶结点。此时，直接把该结点变为叶结点，如果待删除结点是红色，则算法结束。否则，待删除结点所在子树的阶减1，造成红黑树不平衡，需要调整，该算法称为“双黑”结点调整算法。所谓“双黑”结点，即该结点需要代表两个黑色结点，才能维持树的平衡，事实上是不允许的，因此需要调整（后面将会具体介绍“双黑”结点调整算法）。删除上述黑结点后，替代被删结点的外部叶结点就是“双黑”结点。

（2）待删除结点只有一个外部叶结点，可知其非空子结点肯定为红色，待删结点肯定为黑色，此时只需把其子结点提升到删除结点位置，颜色变为黑色即可。

上述删除算法把待删除结点已删掉，此时只需考虑解决“双黑”结点现象即可，这也是删除算法中最复杂的地方。下面分情况讨论，注意所讨论情况是针对“双黑”结点是左子结点的情况；若是右子结点，可参照左子结点的处理方法，注意左右对称地修改。

情况1：双黑结点的兄弟结点是黑色，且子结点有红色。

需要细分为两种类型讨论。

情况（a），如图11.21所示。双黑 X 与红色侄子 D 对称八字形外撇。此时，将兄弟结点 C 提上去，C 继承原父结点 B 的着色。然后，把 B 着为黑色，D 着为黑色，其他颜色不变。

情况（b），如图11.22所示。双黑 X 与红色侄子 D 是同边顺的。此时，把 D 提上去成为子根结点，继承原子根 B 的颜色，B 着为黑色。

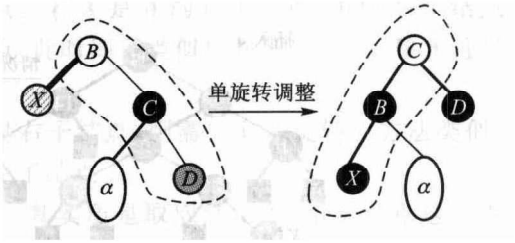

图11.21 情况1（a）：
双黑结点与红色侄子呈八字形对称

也有人把情况1（b）看成双旋转调整：先旋转一次，让 D 作为 C 的父结点，转换为情况1（a）；再进行第二次调整，让 D 作为根结点，成为图11.22中右边的情况。

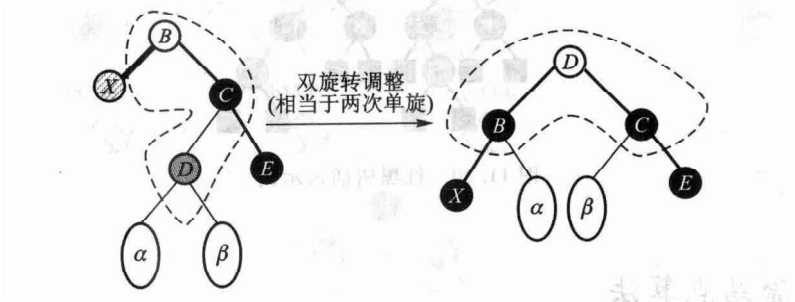

图11.22 情况1（b）：双黑结点与红色侄子同边顺

情况2：双黑结点的兄弟结点是黑色，并且兄弟结点有两个黑色子结点。

如图11.23所示，双黑结点的兄弟结点 C 是黑色，且有两个黑色子结点 D 和 E。进行换色处理，把 C 着红色，B 着黑色，如果 B 原为红色，则算法结束。否则，对 B 继续做双黑调整。

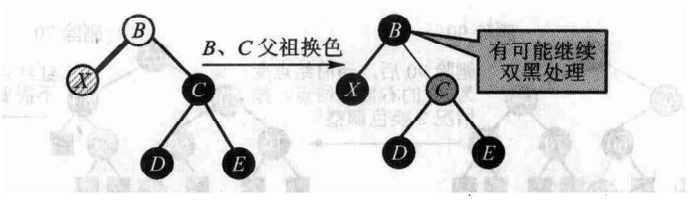

图 11.23 情况 2 双黑的兄弟结点为黑色，且兄弟结点有两个黑色子结点

情况 3：双黑结点的兄弟结点是红色。

如图 11.24 所示，双黑的父结点肯定为黑色（否则与红色子结点 C 冲突），双黑的兄弟结点 C 的子结点 α 和 β 肯定都是黑色。因此，可做旋转，此时 X 结点仍是双黑结点，只是转化为前面两种情况之一继续处理。

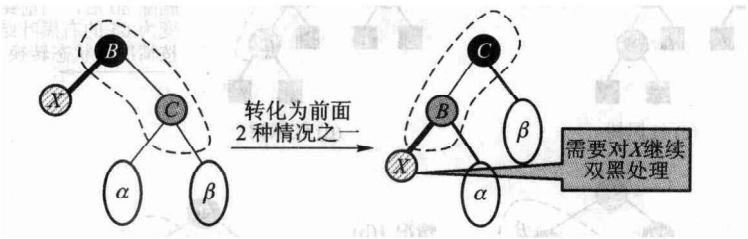

图 11.24 情况 3 双黑的兄弟结点为红色

以上各种情况讨论了删除结点所作的操作，重点在于对删除黑色结点时的调整操作，由以上步骤可知，调整“双黑”结点最多有三步旋转操作，其他操作则是换色、递归处理，相对简单。

图 11.25 所示为在红黑树中连续删除 3 个关键码之后引起的调整示例，其调整运用到了双黑冲突处理策略。

从图 11.25(a)的红黑树中删除 90，结点 80 的右子结点——黑的空叶结点变成当前结点，其兄弟 70 是带两个黑子女的黑结点，按情况 2 进行“双黑”调整，对 70 和 80 进行换色处理。由于 80 原来是红色的，因此改变颜色后使树重新恢复了图 11.25(b)所示的平衡状态。

继续从图 11.25(b)中将 70 删除，就得到了图 11.25(c)。由于删除的是红色结点，删除后树仍然是平衡的。

从图 11.25(c)中将 80 删除，得到图 11.25(d)。这次删除的是非根黑色结点，破坏了红黑树平衡，替代 80 的是结点 X，X 结点符合情况 3，进行双黑调整，得到图 11.25(e)。调整后的当前结点还是 X，符合情况 1(b)，进行双黑调整，得到图 11.25(f)。

上述调整过程中，需要注意图的双黑结点是右子结点的情况，与前面情况 1 至情况 3 的图例是对称的。

上面介绍的插入和删除操作，为了重新恢复红黑树的平衡需要自底向根的方向调整，其平均和最差检索时间都是 $O(\log_2 n)$。因此，红黑树的结点往往由数据、左指针、右指针、颜色和父指针 组成。如果不给出父指针，就可以借用堆栈，从根结点至插入/删除结点路径上所遇到的每个

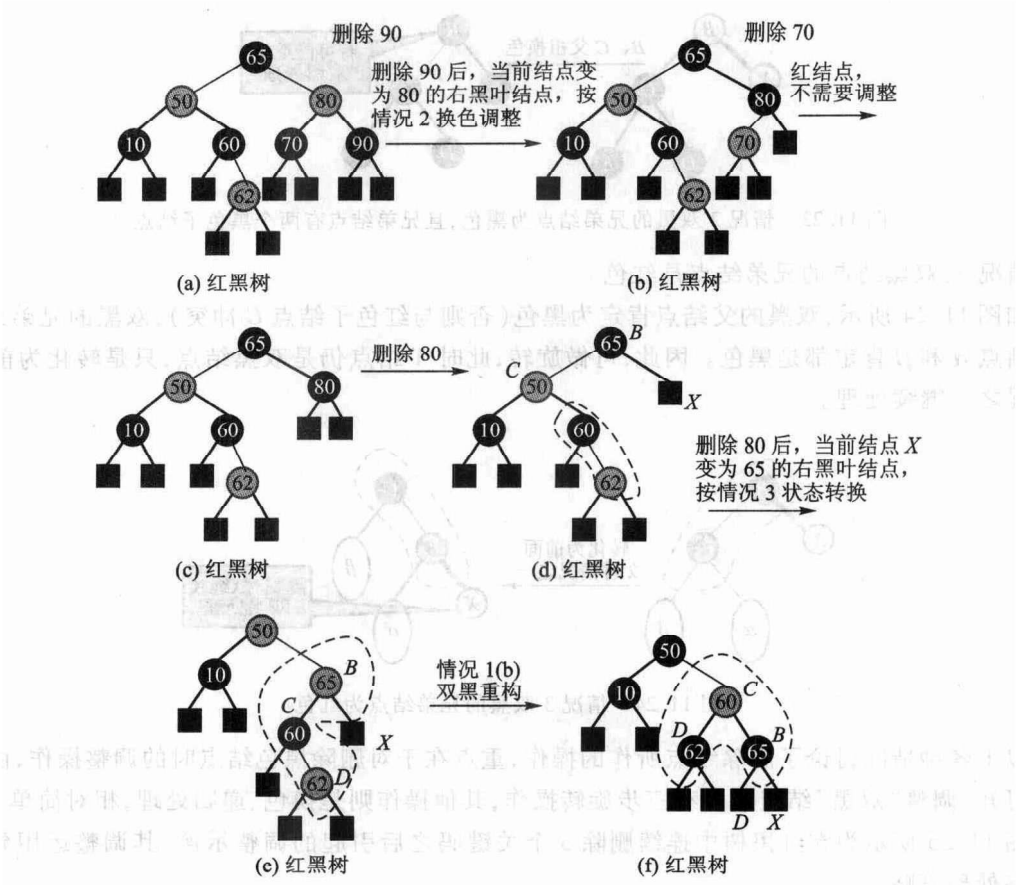

图 11.25 红黑树连续删除 90、70、80，引起的双黑调整示例

结点指针都保存在堆栈中，以方便回溯。此外，还可以采用自顶向下的迭代式插入/删除调整方法，其调整速度更快。

红黑树在很多地方都有应用，例如 C ++ STL 的 set、multiset、map、multimap 等数据结构都应用了红黑树的变体。在 Linux 内核中，用于组织虚存“区间”的数据结构也是红黑树。

本章小结

索引就是把关键码与其主文件中的数据记录位置相关联的过程，索引文件则是用于记录这种联系的文件组织结构。索引技术的目的是在支持高效数据（尤其是外存的大量数据）检索的同时，支持高效的插入和删除操作，它是数据库的核心技术之一。

实际应用中，经常需要根据属性的值来查找记录，因此采用倒排索引技术。每一项都包括一