B+ Tree Report

杨嘉成 林伟鸿 谭博文

April 5, 2016

Contents

1 功能介绍

B+ Tree 是一种树数据结构，通常用于数据库和操作系统的文件系统中。B+ Tree 的特点是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度。B+ Tree 的应用相当广泛，尤其是在数据库中，例如 MySql Database 就是以 B+ Tree 为基础实现的。在我们组的 B+ Tree 实现中，我们将 B+ Tree 封装成了一个类，这个类可以支持维护一个索引文件，通过这个类，我们可以支持插入、删除、查找等功能。

2 原理说明

B+ Tree 的特点是能够保持数据稳定有序，其插入与修改拥有较稳定的对数时间复杂度。B+ Tree 在节点访问时间远远超过节点内部访问时间的时候，比可作为替代的实现有着实在的优势。这通常在多数节点在次级存储比如硬盘中的时候出现。通过最大化在每个内部节点内的子节点的数目减少树的高度，平衡操作不经常发生，而且效率增加了。这种价值得以确立通常需要每个节点在次级存储中占据完整的磁盘块或近似的大小。B+ Tree 背后的想法是内部节点可以有在预定范围内的可变数目的子节点。因此，B+ 树不需要象其他自平衡二叉查找树那样经常的重新平衡。对于特定的实现在子节点数目上的低和高边界是固定的。

3 正确性分析

不难发现 B+ Tree 是一种 M 叉平衡树，我们在插入和删除的时候只需要保证每个节点 key 值的正确性和 child 的正确性就能够保证整个数据结构的正确性。

4 时空复杂度证明

不难发现 B+ Tree 是一种 M 叉平衡树，因此我们只需要在维护的时候保证每个节点满足平衡条件：每个节点的 key 值大于等于 $M/2$，小于等于 M，也就能够保证树的高度是 $O(\log_m n)$ 的（因为第 h 层的节点数量至少是 $(M/2)^h$，深度一定是对数级别的）。因此插入、删除、查找等复杂度也是对数级别的复杂度。

5 具体实现方式及细节

5.1 一些定义

为了方便之后的描述，我们先做一些定义：

Definition 5.1. *一些 B+ Tree 的定义：*

- *key 表示关键值，child 表示关键值对应的儿子节点在索引文件中的存储地址；*
- *关键值是从小到大排放好的，查找的时候可以像普通平衡树那样顺次查找关键值；*
- *B+ Tree 树的叶子节点中存放的是 key 所对应的文件在磁盘上的地址，内节点中的 child 中存放的是儿子在索引文件中的位置；*

Definition 5.2. *假设 B+ Tree 是 M 叉树，那么：*

- *M 是偶数；*
- *每个内节点至少有 $M/2$ 个儿子，至多有 M 个儿子；*
- *每个节点中存放的 key 都是儿子中 key 最小的那一个；*

5.2 查询操作

- 从根节点开始查找，每次 key 值所在的区间，然后走到其对应的儿子；
- 如果当前节点是叶子节点，那么如果 key 值等于当前儿子节点的 key 那么返回找到，否则返回没有找到。

Algorithm 1 Find Operation

```
1: function FIND(pCurrent, key)
2:     repeat
3:         current ← READ(pCurrent)
4:         for all child do
5:             if current.isLeaf & child.key = key then
6:                 return True
7:             end if
8:         end for
9:         for all child do
10:            if key ∈ child.Range then
11:                pCurrent ← child
12:            end if
13:        end for
14:    until current.isLeaf = True
15: end function
```

5.3 插入操作

- 首先查找元素是否存在，如果存在那么抛出异常；
- 否则，我们从根开始往下面找，判断一下如果当前节点插入之后会使得当前节点的儿子个数超过 M，那么将当前的儿子分裂成两半，然后再插入；
- 注意在插入之前如果根节点超过限制，那么就新开一个点作为根节点，然后分裂；
- 当然步骤一和步骤二可以合起来，在插入的时候顺便判断是否存在；

5.4 删除操作

- 首先，查找要删除的值。接着从包含它的节点中删除这个值。
- 如果节点处于违规状态则有两种可能情况：
  - 它的兄弟节点，就是同一个父节点的子节点，可以把一个或多个它的子节点转移到当前节点，而把它返回为合法状态。如果是这样，在更改父节点和两个兄弟节点的分离值之后处理结束。
  - 它的兄弟节点由于处在低边界上而没有额外的子节点。在这种情况下把两个兄弟节点合并到一个单一的节点中，而且我们递归到父节点上，因为它被删除了一个子节点。持续这个处理直到当前节点是合法状态或者到达根节点，在其上根节点的子节点被合并而且合并后的节点成为新的根节点。

具体一点讲，删除操作大致有 22 种情况，限于篇幅，这里不描述。

6 前端代码

前端我们分成了 6 个类，其中一个是基础类 BPlusBase，其他 5 个类都是继承这个类的，另外 5 个类是 BPlusResult、BPlusIO、BPlusTree、BPlusNode、BPlusError，分别表示返回结果类，输入输出控制类、主要操作的 B+ Tree 类、节点类、错误类，其中所有的核心操作都在 BPlusTree 中实现，剩下的边角余料在剩下的 4 个类中实现。我们的 IO 类对文件的控制如下：

- 第一个 int 类型是 root 所在的地址，因为在插入和删除过程中 root 可能会改变；
- 接下来若干都是节点类型的数据，每个节点存放了 M 个 key 和 M 个 child，虽然造成了一定的空间浪费，但是对于实现却很有好处。

7 提升空间

针对我们组的代码，我们组还可以作出如下提高：

- 空间回收技术：我们组并没有实现这个功能，在删除的时候我们只是将索引文件中的指针置成 0，在下次插入的时候并没有利用这些空间，造成了空间浪费；
- 可持久化技术：显然这个数据结构是可以可持久化的，如果能够实现，说不定可以在现实中对数据恢复有所帮助；
- 一些常数优化：例如在函数的变量传递的时候，应该尽量用实际的点而非用文件偏移量来传递变量，这样能够减少一半的磁盘访问；
- 度数可奇数化：我们实现的 B+ Tree 并不支持度数是奇数的 B+ Tree；

8 参考文献

- An implementation of B+ Tree
- B+ Tree in wikipedia
- B+ Tree Visualization
- 《数据结构：思想与实现》