B+树算法的Java实现方法研究

时亚南
(新疆维吾尔自治区特种设备检验研究院,新疆 乌鲁木齐 830011)

摘 要:随着人们对移动数据处理和管理需求的不断提高,与各种手持设备能够紧密结合在一起的嵌入式数据库逐渐成为人们研究的热点。而B+树作为一种成熟的数据结构,在数据库索引构建以及文件索引数据组织方面具有极其广泛的应用。为深入研究嵌入式数据库中B+索引的构建机制,文中使用Java语言实现了基于内存的B+树,并对其性能进行了评估测试。测试结果表明,该B+树具有良好的数据处理能力。

关键词:嵌入式数据库;B+树;索引;Java
中图分类号:TP302.1 **文献标识码**:A **文章编号**:1673-629X(2015)01-0111-04
doi:10.3969/j.issn.1673-629X.2015.01.025

Study on Java Implementation Method of B+ Tree Algorithm

SHI Ya-nan
(Xinjiang Uygur Autonomous Region Inspection Institute of Special Equipment, Urumqi 830011, China)

Abstract: As the mobile data processing and management needs continue to improve, the embedded database which can closely link with a variety of handheld devices has become a hot research of people. While B+ tree as a mature data structure, has an extremely wide range of applications in database indexes building and index data file organization. To further study the embedded database's B+ index building mechanism, use Java language to achieve a memory-based B+ tree in this paper, and give a evaluation tests for its performance. The test results have showed that the B+ tree has a good data processing capabilities.

Key words: embedded database; B+ tree; index; Java

0 引 言

移动互联网时代人们期望能够通过各种手持移动设备(手机、PDA、平板电脑等)随时随地获取自己想要的信息,而嵌入式数据库技术的出现使人们这种愿望成为现实。目前,关系数据库管理系统,如IBM DB2、Informix、Microsoft SQL Server、Oracle、Sybase ASE、MySQL、PostgreSQL及SQLite等均支持为表构建B+ tree索引。键-值数据库管理系统如CouchDB、Tokyo Cabinet、CodernityDB等均支持通过B+ tree索引访问数据。NTFS、ReiserFS、NSS、XFS、JFS及ReFS等文件系统均采用B+ tree对文件元数据进行索引。因此,对B+ tree这种数据结构进行深入的研究具有十分重要的意义。

文献[1-3]从理论上阐述了使用B+ tree构建的数据库索引可以极大程度地提高数据检索的效率;文献[4-14]主要从如何利用B+ tree实现对文件索引数据的组织、索引库的构建及跨数据范围的检索等方面对B+ tree进行阐述。文中从中抽象出了B+树实现的形式化定义,重点从工程实践的角度对B+树加以技术实现,以期对其有一个更为深入的理解。

1 B+树简介

B+ tree是一种多叉树,每个节点可以拥有大量子节点,同一个树中允许不同的节点可以拥有不同数量的子节点。一棵B+ tree由一个根节点、若干个内部节点和若干个叶节点组成。根节点要么是叶节点(此时整个树只有一个节点),要么至少包含两个子节点。

B+ tree是平衡树,即所有叶节点的深度均相等,键和其对应的数据只保存在叶节点,内部节点只出现键和指向下层子树的指针,同一层节点内的键均从左

收稿日期:2014-04-17 **修回日期**:2014-07-17 **网络出版时间**:2014-10-27
基金项目:新疆维吾尔自治区科技攻关项目(200931103)
作者简介:时亚南(1988-),男,硕士研究生,研究方向为智能信息检索、数据库技术及应用。
网络出版地址:http://www.cnki.net/kcms/detail/61.1450.TP.20141027.1429.009.html

至右排序。内部节点的键只起到查找导向作用,可以不是实际存储在叶节点的键。

B+ tree 的阶(或分支因子)记为 b,它是树中内部节点容量的度量(即每个内部节点可以存储子节点指针的数量)。若内部节点实际存储的子节点指针数量记为 m,实际存储的键数量记为 k,则 $k=m-1$,且 m 必须满足 $\lceil b/2 \rceil \leqslant m \leqslant b$ 约束,而对于根节点(有子节点时)则必须满足 $2 \leqslant m \leqslant b$ 约束。叶节点中只存储键和对应的值,没用指向子节点的指针,当然其实际存储的键数量 k 必须满足 $\lfloor b/2 \rfloor \leqslant k \leqslant b-1$ 约束。

在 B+ tree 只有很少记录的情况下,整个树只有一个节点,它是一个叶节点(因为只存储了键和对应的值),当然它也是整个树的根节点,此时 k 必须满足 $0 \leqslant k \leqslant b-1$ 约束。

例如,图 1 为一个 4 阶 B+ tree 的实例,而对于 7 阶 B+ tree,根节点以外的内部节点存储的子节点指针数量在 4 和 7 之间,而根节点存储的子节点指针数量在 2 和 7 之间。

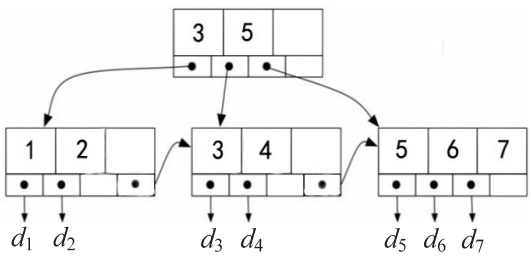

图 1 一个 4 阶 B+ tree 的实例

2 B+树的特性

对于 b 阶 B+ tree,如其层数为 h,存储的记录数为 n,则它具备以下特性:

(1)最多能够存储的记录数量为:

$$n_{max} = b^h - b^{h-1}$$

(2)能够存储的记录数量下限为:

$$n_{min} = 2\lceil \frac{b}{2} \rceil^{h-1}$$

(3)能够存储的键数量下限为:

$$n_{kmin} = 2\lceil \frac{b}{2} \rceil^{h} - 1$$

(4)存储整棵树的空间复杂度为 $O(n)$;

(5)查找、插入及删除 1 条记录的时间复杂度均为 $O(\log_b n)$;

(6)范围查找(假设满足的记录数为 k)的时间复杂度为 $O(\log_b n + k)$。

3 B+树的 Java 实现

本实现用数组存储记录,节点内部排序效率不高,可以通过使用支持排序的容器改进。

3.1 数据结构定义及描述

本实现用到了三个类,即节点类 Node、兄弟节点类 Sibling 和节点内记录类 Record。下面对它们的数据结构进行详细阐述。

(1)节点类 Node 的数据结构描述如下:

```
public class Node {
private int keyCount; //节点中已存储的记录个数
private Record[ ] records; //存储记录的数组
private Node leftMostChild; //最左边子节点指针
private Node left; //同层左侧节点
private Node right; //同层右侧节点
private Node parent; //父节点
}
```

其中,叶节点不使用 leftMostChild,标准的 B+ tree 是由叶节点使用 right 指针把所有叶节点从左至右连接为一个链表以支持>或>=查询,为叶节点添加 left 指针则可以支持<或<=查询,parent 指针和内部节点的 left 及 right 主要用于调试。

(2)兄弟节点类 Sibling 的数据结构描述如下:

```
public class Sibling {
private boolean isLeft; //是否为左侧兄弟
private Node sibling; //节点
}
```

其中,若 isLeft 为 true,则表示 Sibling 指向的节点是某节点的左侧兄弟,否则为右侧兄弟。

(3)节点内记录类 Record 的数据结构描述如下:

```
public class Record {
private Comparable key; //键
private Object value; //叶节点使用,保存键对应的值
private Node child; //内部节点使用,保存子节点指针
}
```

其中,内部节点只使用 key 和 child,叶节点只使用 key 和 value。

3.2 B+树须满足的约束

该 B+树的实现具有以下约束:

(1)MAX_KEYS≥2;

(2)每个节点最多可以保存 MAX_KEYS 条记录,当记录数超过 MAX_KEYS,节点发生分裂;

(3)节点中,指针数量比记录数量多 4 个,其中 leftMostChild 用来指向键小于该节点内最小键的子树,left 用来指向同层左侧的节点,right 用来指向同层右侧的节点,parent 用来指父节点;

(4)当树只有一层,即只有一个节点时(此时它既是根节点,也是叶节点),该节点可以含 0 至 MAX_KEYS 条记录;

(5)当树只有两层时,由根节点和叶节点构成,没有中间层,根节点至少要有 1 条记录,叶节点记录数量的下限为 MAX_KEYS/2 上取整;

(6)当树高超过两层时,由根节点、中间节点和叶节点构成,根节点至少要有 1 条记录,每个中间节点记录数量的下限为 MAX_KEYS/2 下取整;

(7)叶节点存储的记录由键和值构成;

(8)树高超过 1 层时,内部节点(根节点和中间节点)存储的记录由键和子节点指针构成,最左边的指针指向键<k_1的子树,k_1的指针指向键>=k_1且<k_2的子树,k_i的指针指向键>=k_i且<k_{i+1}的子树,假设该节点有 n 个键,则最后一个键为 k_n,指针指向键>=k_n的子树;

(9)不允许出现重复的键。

3.3　B+树查询算法实现

查询算法的输入为查找键 Key,算法输出为键对应的值 Value。它的实现流程为:首先调用递归查找函数 treeSearch 找到可能包含该键的叶节点,然后在叶节点内部查找键对应的值。递归查找函数 treeSearch 输入为节点指针 nodePointer,查找键 Key,当前层 layer,输出为可能包含键 Key 的叶节点 nodePointer,具体实现过程为:

①如果 nodePointer 是叶节点,直接返回 nodePointer,否则执行步骤②;

②如果 Key 小于当前叶节点的最小键,将当前节点最左边的子树指针赋给 nodePointer,如果 Key 大于等于当前节点中最大的键,将当前节点最后一条记录的子树指针赋给 nodePointer,否则,找到记录 Rec,使得 $K_i<=K_{rec}<K_{i+1}$,并将记录 Rec 的子树指针赋给 nodePointer;

③return treeSearch(nodePointer,Key,layer-1)。

3.4　B+树插入算法实现

插入算法实现过程为:首先调用查找函数检查要插入的键是否已存在,若已存在,返回插入失败,否则,调用递归插入函数 treeInsert 将键值对插入。递归插入算法以根节点、要插入的记录及 B+树目前的高度为参数,返回新增子节点的记录 newRec。算法背后的思想是在相应的子节点递归调用插入算法,这将导致逐层向下调用插入函数,直到在键所属的叶节点插入记录后,再逐层返回到根节点,每层都可能发生节点分裂,叶节点发生分裂后,直接把新节点的最小键复制到上层节点,而内部节点发生分裂后,则是把新节点的最小键推到上层节点。具体算法如下:

①如果层数为 1,表示当前节点是叶节点,执行步骤②,否则当前执行步骤④;

②如果当前节点有可用空间,直接在当前节点插入记录 R,返回空,否则执行步骤③;

③分裂当前节点(首先创建一个新节点,使新节点成为当前节点的右侧兄弟,然后将当前节点中的记录加上要插入的记录平均分配到当前节点和新节点中,最后用新节点最小的键和指向新节点的指针设置 newRec),如果当前节点是根节点,树长高 1 层,返回 newRec;

④将当前节点的子树赋值给 pointerNode,调用 treeInsert(P,R,layer-1),并用其返回值设置 newRec,如果 newRec 值为空,表示下层没有发生节点分裂,直接返回 newRec,否则执行步骤⑤;

⑤如果当前节点未满,直接在当前节点插入记录 R,返回空,否则执行步骤⑥;

⑥分裂当前节点(首先创建一个新节点,新节点成为当前节点的右侧兄弟,将当前节点中的记录加上要插入的记录平均分配到当前节点和新节点中,用新节点最小的键和指向新节点的指针设置 newRec,将新节点的最左边指针设置为新节点最小键对应的指针,然后删除新节点最小键记录),如果当前节点是根节点,树长高 1 层,返回 newRec。

3.5　B+树删除算法实现

删除算法也是首先调用查找函数检查要删除的记录是否存在,不存在的话,返回删除失败,否则调用递归删除函数 treeDelete 进行删除。调用递归删除函数 treeDelete 算法的输入为当前节点 N,父节点 P,记录 R,当前层 layer,返回被合并子节点的记录 oldRec。算法实现的具体流程为:

①如果当前层 layer 为 1,表示当前节点是叶节点,执行步骤②,否则当前执行步骤⑤;

②如果 N 为根节点,直接删除 R,返回空,如果 N 的记录数大于叶节点应当保留的记录下限,直接删除记录 R,设置 oldRec 为空,否则执行步骤③,如果删除的记录是 N 的最小键记录且 N 不是父节点最左侧孩子,将父节点指向 N 的那条记录的键修改为 N 节点中的最小键,返回 oldRec;

③删除 N 中记录 R,设置 S 为 N 的兄弟,如果 S 的记录数大于叶节点应当保留的记录下限,从 S 借 1 条记录(如果从左兄弟借,借来的是兄弟的最大键记录,必须将父节点中指向 N 的那条记录的键修改为借来键;如果从右兄弟借,借的是兄弟的最小键记录,必须将父节点指向右兄弟的那条记录的键修改为右兄弟新的最小键),返回空,否则执行④;

④将 N 和 S 合并(始终向左合并,如果和左兄弟合并,把 N 的记录移到 S,设置 oldRec 为父节点中指向 N 的那条记录;如果和右兄弟合并,则把 S 中的记录移到 N,设置 oldRec 为父节点中指向 S 的那条记录),返回 oldRec;

⑤将当前节点的子树赋值给 pointerNode,调用 treeInsert(P,R,layer-1),并用其返回值设置 oldRec,如果 oldRec 为空,表示下层没有发生节点合并,直接

返回空,否则执行步骤⑥;

⑥删除 N 中的记录 oldRec,如果 N 是根节点,树高降低 1 层,返回空,如果 N 的记录数大于内部节点应当保留的记录下限,返回空,否则执行步骤⑦;

⑦设置 S 为 N 的兄弟,如果 S 的记录数大于叶节点应当保留的记录下限(可以借),从 S 借 1 条记录(如果从左兄弟借,借的是兄弟的最大键记录,N 的最左侧指针必须修改为借来记录指向的节点,借来的记录必须修改为 N 原来最小记录的指针,N 的父节点中指向 N 的记录的键必须和借来的键调换;如果从右兄弟借,S 的最左侧指针必须修改为借走的那条记录指向的节点,借来的记录的指针域必须修改为 S 的最左侧节点,S 的父节点中指向的记录的键必须和借走的键调换),返回空,否则执行步骤⑧;

⑧将 N 和 S 合并(始终向左合并,如果和左兄弟合并,把 N 的记录移到 S,设置 oldRec 为父节点中指向 S 的那条记录,向左兄弟补充 1 条记录,键为 N 的父节点中指向 N 的那条记录的键,而指针为 N 的最左侧指针;如果和右兄弟合并,则把 S 中的记录移到 N,设置 oldRec 为父节点中指向 S 的那条记录,向 N 补充 1 条记录,键为 S 的父节点中指向 S 的那条记录的键,而指针为 S 的最左侧指针),返回 oldRec。

4 性能测试

文中测试环境为:Centos 6.4 操作系统,2 G 内存,ext3 文件系统。测试数据来源为:100 万条随机生成的数据。测试结果为:查询每条数据平均耗时 0.103 3 ms,插入每条数据平均耗时 0.286 7 ms,删除每条数据平均耗时 0.373 1 ms。

由此可见,该 B+树对每条数据进行查询、插入和删除操作平均耗时不到 0.4 ms。因此,该 B+树具有较好的吞吐性能。

5 结束语

文中首先结合 B+树的特性及约束,抽象出了实现 B+树查找、插入及删除等操作的数据结构定义,然后对这三种常见操作的实现流程进行了详细阐述,最后对这三种操作的综合性能进行了测试评估。测试结果表明,B+树在数据组织方面表现出的综合性能良好,从而为深入研究其在数据库索引及文件索引方面的应用奠定了扎实的实践基础。

参考文献:

[1] 王英强,石永生. B+树在数据库索引中的应用[J]. 长江大学学报(自然科学版),2008,5(1):233-235.

[2] 张学琴. 嵌入式数据库 B+树索引机制研究及其改进[J]. 计算机与现代化,2009(12):68-71.

[3] 刘彩苹,李仁发,刘喜苹. 面向嵌入式数据库的改进 B~+-树索引机制[J]. 计算机工程与科学,2007,29(1):101-102.

[4] 张 华,顾红飞,刘 涛. 基于 B+树的文本信息检索技术[J]. 皖西学院学报,2010,26(2):31-35.

[5] 长孙妮妮,张毅坤,华灯鑫,等. 一种基于 B+树的混合索引结构[J]. 计算机工程,2012,38(14):35-37.

[6] 陆志峰,陈新建. B~+树索引文件结构的优化设计[J]. 计算机工程与设计,2000,21(3):40-44.

[7] 徐德智,郭玉珂,孙 莹,等. XML 数据 B+树存储索引研究[J]. 计算机工程与应用,2004,40(22):168-170.

[8] 孙建锋,冯 超,张 权. 基于 B+树的 RFID 防碰撞算法分析与改进[J]. 计算机工程,2013,39(9):49-51.

[9] 吴伟民,卢 琦,王振华,等. NTFS 目录下索引 B+树结构动态解析[J]. 计算机工程与设计,2010,31(22):4843-4846.

[10] Park J,Hong B,Ban C. A query index for continuous queries on RFID streaming data[J]. Science in China,2008,51(12):2047-2061.

[11] Kalay U,Kalipsiz O. A comparison study of moving object index structures[J]. Journal of Computer Science & Technology,2009,24(6):1098-1108.

[12] On S T,Hu Haibo,Li Yu,et al. Flash-optimized B+-tree[J]. Journal of Computer Science & Technology,2010,25(3):509-522.

[13] Ye Xiaoping,Tang Yong,Chen Luowu,et al. Study and application of temporal index technology[J]. Science in China Series F:Information Sciences,2009,52(6):899-913.

[14] Li Guoliang,Feng Jianhua,Zhou Lizhu. Keyword searches in data-centric XML documents using tree partitioning[J]. Tsinghua Science and Technology,2009,14(1):448-461.

B+树算法的Java实现方法研究

作者： 时亚南， SHI Ya-nan
作者单位： 新疆维吾尔自治区特种设备检验研究院,新疆 乌鲁木齐,830011
刊名： 计算机技术与发展 ISTIC
英文刊名： Computer Technology and Development
年，卷(期)： 2015(1)