RFN-B$^+$树索引文件及其有效性*

姚卿达 杨桂桢 张俊欣

(中山大学信息科学与技术学院 广州 510275)

摘要 在对比传统的B树和B$^+$树的定义和操作算法的基础上,定义了一种新的B$^+$树:RFN-B$^+$树,以获得更高的空间利用率和可用性.首先比较和分析了RFN-B$^+$树与传统B$^+$树的空间效率,然后讨论了RFN-B$^+$树索引文件的有效性以及支持这种有效性的全链接指针结构和两个备用模块:基于虚拟根结点的随机检索算法和重构结点的算法.

关键词 B$^+$树,索引,有效性,算法.

中图法分类号 TP311

1 RFN-B$^+$树的背景

LNDBMS是我们设计的一个面向数据采集、基于Client/Server计算体系结构的高效的数据库管理系统.[1]在为LNDBMS设计一个B$^+$树索引文件管理系统BPTIFMS(B plus tree index file management system)的时候,我们采用了自己定义的RFN-B$^+$树索引结构.

访问B$^+$树索引文件的基本单位是B$^+$树中的一个结点,在磁盘上是一个连续存取的数据块.在为BPTIFMS设计一个通用的B$^+$树索引文件结构的时候,B$^+$树的结点设计为定长的数据块,阶数为变量.

设B$^+$树的阶数为m,一个结点数据块的大小为L字节,索引项中一个关键字的长度为k字节,系统中的磁盘空间地址指针的长度为a字节.根据参考文献[2,3],B$^+$树中的每个叶结点可以容纳$\left\lfloor \frac{L}{k+a} \right\rfloor$个索引项,每个内部结点可以容纳$\left\lfloor \frac{L}{k+a} \right\rfloor$个指向子树的指针,B$^+$树的阶$m=\left\lfloor \frac{L}{k+a} \right\rfloor$.每个结点数据块的内碎片的长度$F=L \bmod (k+a)$.

B$^+$树索引集(所有内部结点的集合)的主要功能是起导航作用.如果能在相同的空间下增加每个内部结点的分支数目,就可以提高查询效率和空间利用效率.通过计算发现,$F \geq a$的概率$P\{F \geq a\}$在80%以上.因此,有必要修改一般B$^+$树的定义[2,3],以便尽可能地增加B$^+$树中内部结点的分支数目.本文将BPTIFMS中采用的B$^+$树称为RFN-B$^+$树(Refined B$^+$-tree,精细化的B$^+$树,简称为RFN-B$^+$树).

2 RFN-B$^+$树的定义

RFN-B$^+$树的定义如下:

定义1. 与一般B$^+$树相比,一棵具有m阶内部结点和n阶叶结点的RFN-B$^+$树应该满足以下限制条件:

(1) 每个内部结点至多有m个指针;

(2) 除了根结点外,每个内部结点至少有$\left\lceil \frac{m}{2} \right\rceil$个指针;

(3) 如果根结点不是叶结点,则根结点至少有两个指针;

(4) 有k个指针的内部结点必有$k-1$个关键字;

(5) 每个叶结点至多有n个指针;

(6) 除了根结点外,每个叶结点至少有$\left\lceil \frac{n}{2} \right\rceil$个指针;

(7) 有k个指针的叶结点必有k个关键字;

---

* 作者姚卿达,1937年生,教授,主要研究领域为数据库,知识库.杨桂桢,1972年生,硕士,主要研究领域为数据库,知识库.张俊欣,女,1973年生,硕士生,主要研究领域为数据库,知识库.

本文通讯联系人:姚卿达,广州 510275,中山大学信息科学与技术学院

本文1997-07-02收到原稿,1997-10-17收到修改稿

(8) 所有的叶结点包含了有关全部关键字的索引项,并且叶结点本身按照关键字的大小自小到大顺序链接起来.

RFN-B⁺树的内部结点与 B 树的内部结点具有相似结构,而它的叶结点与 B⁺树的叶结点具有相似结构.对 RFN-B⁺树的插入关键字和删除关键字的操作,应该区分内部结点和叶结点的情形.对于 RFN-B⁺树的内部结点,遵循对 B 树的内部结点的操作;而对于 RFN-B⁺树的叶结点,遵循对 B⁺树的叶结点的操作.相关的操作,在参考文献[2,3]中有详细的描述.

3　RFN-B⁺树与一般 B⁺树的空间开销对比

考虑一棵 n 阶的一般 B⁺树 T_N 和一棵具有 m 阶内部结点和 n 阶叶结点的 RFN-B⁺树 T_R. T_N 的顺序集和 T_R 的顺序集有着完全相同的结构.也就是,它们叶结点的数目和内容以及叶结点之间的顺序链接关系完全相同.因为 T_R 与 T_N 的叶结点的阶相同,所以对于同一个关键字的插入或删除操作,在 T_R 的顺序集和 T_N 的顺序集上所操作的结点对象以及所进行的操作步骤都完全相同.而且,对索引集中结点的操作并不改变顺序集的结构.所以,在进行相同的关键字的插入或删除操作之后,T_N 的顺序集的结构与 T_R 的顺序集的结构仍然相同.

根据以上的讨论和数学归纳法,可以得到定理 1.

定理 1. 设 T_N 是一棵初始为空的 n 阶的一般 B⁺树,T_R 是一棵初始为空的具有 m 阶内部结点和 n 阶叶结点的 RFN-B⁺树,则在对 T_N 和 T_R 分别进行相同的插入和删除关键字的操作序列之后,T_R 的顺序集的结构与 T_N 的顺序集的结构完全相同.

根据定理 1 可知,RFN-B⁺树与一般 B⁺树的区别主要在于索引集的结构上.下面我们就来比较一下在顺序集结构相同的情况下,一般 B⁺树和 RFN-B⁺树在索引集上的空间开销.

对于给定的结点数据块大小 L,关键字长度 k,地址指针长度 a,如果一般 B⁺树的阶为 m,则对于相应的 RFN-B⁺树,它的叶结点的阶为 m,内部结点的阶为 m 或者 $m+1$.对于相应的 RFN-B⁺树,内部结点的阶为 $m+1$ 的概率在 80%以上.所以,我们在计算 RFN-B⁺树的索引集中的结点数目时,一般就认为它的内部结点的阶是 $m+1$.

设顺序集中有 N 个索引项.B⁺树结点的平均充满率为 α,索引集中有 I_N 个结点.RFN-B⁺树的结点的平均充满率为 β,索引集中有 I_R 个结点.为了估计 I_N 和 I_R 的大小,根据 S. B. Yao[4], D. DeJong[4]和 Andrew Yao[5]做出的结论:B 树的结点的平均充满率为 69%,我们取 $\beta=69\%=\alpha$.因为顺序集中有 N 个索引项,所以顺序集中有 $\frac{N}{\alpha m}$ 个结点.这样,就得到

$$I_N=\frac{N}{\alpha m}\cdot\frac{1}{\alpha m}+\frac{N}{\alpha m}\cdot\frac{1}{(\alpha m)^2}+\ldots+\frac{N}{\alpha m}\cdot\frac{1}{(\alpha m)^n}+\ldots=\frac{N}{\alpha m}\cdot\frac{1}{\alpha m-1}, \tag{1}$$

$$I_R=\frac{N}{\alpha m}\cdot\frac{1}{\alpha(m+1)}+\frac{N}{\alpha m}\cdot\frac{1}{[\alpha(m+1)]^2}+\ldots+\frac{N}{\alpha m}\cdot\frac{1}{[\alpha(m+1)]^n}+\ldots=\frac{N}{\alpha m}\cdot\frac{1}{\alpha(m+1)-1}. \tag{2}$$

定义 B⁺树的索引集相对于 RFN-B⁺树的索引集的额外开销 $OVR=\frac{I_N-I_R}{I_R}\times100\%$,从公式(1)和(2)可以得到

$$OVR=\frac{\alpha}{\alpha m-1}\times100\%. \tag{3}$$

当 $\alpha=69\%$ 时,对不同的 m 计算而得到的 OVR 称为理论值,如表 1 的第 7 栏所示.此外,在设计 BPTIFMS 的前期论证阶段,我们分别实现了一般 B⁺树的索引文件结构和 RFN-B⁺树的索引文件结构,并且对这两种不同的实现方案进行了对比测试.将部分实验结果一起列于表 1 中.

测试步骤如下:

(1) 对于选定的阶数 m,生成一棵空的 m 阶的一般 B⁺树以及一棵空的具有 $m+1$ 阶内部结点和 m 阶叶结点的 RFN-B⁺树;

(2) 随机生成 N 条一定长度的关键字,N 的值列于表 1 的第 2 栏;

(3) 分别对于在步骤(1)中生成的一般 B⁺树和 RFN-B⁺树,连续插入在步骤(2)中得到的 N 条索引项;

(4) 在步骤(3)结束后,分别统计一般 B⁺树的顺序集中的结点数目 S_N,索引集中的结点数目 E_N 以及 RFN-B⁺树的顺序集中的结点数目 S_R,索引集中的结点数目 E_R.由定理 1,$S_N=S_R$,列于表 1 的第 3 栏.E_N 和 E_R 分别列于表 1 的第 4 栏和第 5 栏;

(5) 计算 $\frac{E_N-E_R}{E_R}\times100\%$,即 OVR 的测试值,列于表 1 的第 6 栏.

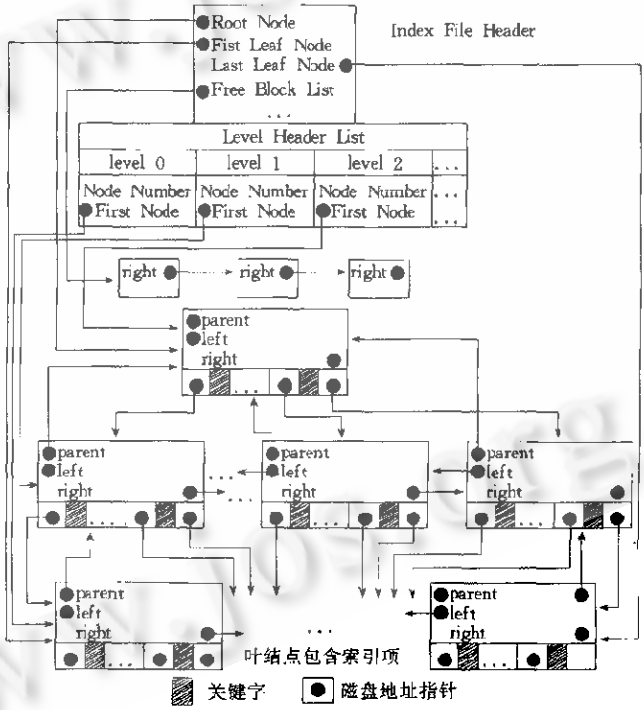
表1 RFN-B⁺树与一般B⁺树在索引集上的空间开销对比

阶数	索引项数目	顺序集	索引集		OVR	
m	N	$S_N(S_R)$	E_N	E_R	测试值	理论值
6	12 000	2 772	811	661	22.69	21.97
9	15 000	2 341	424	382	10.99	13.24
12	20 000	2 374	315	288	9.38	9.48
18	30 000	2 360	202	187	8.02	8.04
24	55 000	3 262	204	192	6.25	4.43
28	80 000	4 088	234	222	5.41	3.77

从表1可以看到,OVR的测试值在理论值的上下范围内有一定的波动.这主要是由于对结点的平均充满率α的估计上的误差引起的.但是,从数量级的观点来看,可以认为OVR的测试值与理论值相符.

4 RFN-B⁺树索引文件的结构

图1所示是一个典型的RFN-B⁺树索引文件的逻辑结构,文件由两部分组成:文件头和数据区.



注:如果叶结点的阶是m,则内部结点的阶一般是$m+1$

图1 一个典型的RFN-B⁺树索引文件的逻辑结构

4.1 文件头

文件头(File Header)主要包含以下几个比较重要的数据结构:

(1) 指向根结点数据块的指针;

(2) 指向顺序集中的第1个和最后1个叶结点数据块的指针;

(3) 指向自由结点数据块链的指针;

(4) 每个关键码字所占用的字节数;

(5) 内部结点的阶数和叶结点的阶数;

(6) 顺序集中索引项的总数目、RFN-B⁺树中结点的总数目以及索引文件中自由结点的总数目;

(7) 层次头的列表. 与 RFN-B$^+$树的每一个层次相联系,层次头(Level Header)由两个数据项构成:指明该层次中所包含的结点数据块的总数目以及指向该层次中的第 1 个结点数据块的指针.

4.2 结点头和索引数据区

文件头之后的连续线性地址空间就是 RFN-B$^+$树索引文件的数据区. 数据区主要是存储 RFN-B$^+$树中的结点,结点为固定长度的数据块. 数据区也以固定长度来分段. 每一段就是一个结点数据块. 数据块之间通过相应的指针链接起来.

每个结点数据块又由两部分组成:结点头和索引数据区. 结点头(Node Header)主要包含以下比较重要的数据结构:(1) 指向父母结点数据块的指针;(2) 指向左兄弟和右兄弟结点数据块的指针;(3) 指明该结点所处的层次;(4) 指明该结点的索引数据区中当前所包含的关键字的总数目.

结点头之后的区域就是索引数据区,主要用于存放关键字和磁盘地址指针. 对索引数据区的解释依赖于文件头和结点头中的信息:(1) 每个关键字所占用的字节数;(2) 地址指针的长度;(3) 结点中当前所包含的关键字的总数目;(4) 结点所处的层次.

这里请注意对一个结点所处的层次的定义. 由于要将叶结点和所有其他内部结点区分开来. 叶结点所处的层次被定义为第 0 层,而叶结点以上的各层逐层加 1. 这样,无论 RFN-B$^+$树的结构如何增减,叶结点所处的层次都是固定的,便于判断.

5 有效性概念的背景

对于 B$^+$树的操作,通常只需要从父母结点指向子女结点的指针就足够了. 对于一般 B$^+$树而言,从根结点到叶结点只有一条路径.

RFN-B$^+$树可以说是一种全链接的 B$^-$树. 从前面的图 1 可以看到,RFN-B$^+$树中包含有十分丰富的指针信息:指向父母结点的指针;指向左右兄弟结点的指针;以及层次头(Level Header)里包含的指针信息. 为 RFN-B$^+$树设计这种全链接的指针结构,主要是为了提高 RFN-B$^+$树的容错(Fault Tolerant)能力以及系统的有效性(Availability).

按照 Jim Gray 的定义[6]:一个系统不是有效的(Unavailable),如果在一个确定的时间限制内,它不能完成正确的操作. 高有效性对系统设计提出了一些新的概念,特别是对于那些提供不间断服务(Continuoys Scrvice)的系统.[6]因此,在对 BPTIFMS 的需求定义中,一方面要求系统处理故障的开销要尽可能地小;另一方面要求系统在故障处理期间,应该尽可能地保持对用户请求的响应.

试想由于磁盘介质故障,B$^+$树中某一个层次 L(非叶结点所在的层次)中的结点数据块遭到了破坏,从而丢失了从父母结点指向子女结点的指针,而其他层次中的结点数据块则保持完好无损. 对于一般的 B$^+$树(只有从父母结点指向子女结点的指针)而言,层次 L 以下各层中的结点都丢失了,尽管从它们中任意一个结点开始的导航都可以正确地进行. 通常,系统这时将不得不把整个 B$^+$树都关闭掉,转入故障处理模块,根据叶结点层次进行整个 B$^+$树的重构工作. 而对于 RFN-B$^+$树,由于在文件头中引进了层次头,RFN-B$^+$树中的任意一个层次都是可以进行顺序检索的. 这时,就可以暂时将层次 L 及其以上的层次全部关闭掉,而将层次 L 下面的那个层次看做一个虚拟的根结点. 在故障恢复工作完成之前,RFN-B$^+$树上的随机检索操作,都将从这个虚拟的根结点开始进行导航. 而与此同时,系统将启动一个故障处理模块,进行 RFN-B$^+$树中结点数据块的重构工作. 所以,只要不是在叶结点数据块中出现了故障,就可以保持对大部分操作请求的响应. 这种故障处理对于事务管理而言是透明的.

这样,至少需要构造两个备用模块:(1) 以某一个层次作为虚拟根结点的随机检索算法;(2) 重构结点数据块的算法.

6 基于虚拟根结点的随机检索算法

6.1 RFN-B$^+$树中指针导航的二义性

以 RFN-B$^+$树中的某一个层次 L 作为虚拟根结点的随机检索,大致可以分为两个步骤:(1) 在层次 L 内进行顺序检索;(2) 以指针为导航进行随机检索.

这里应该注意的是,在 RFN-B$^+$树中,基于指针的导航可能会带来二义性.

根据定义 1,在 RFN-B$^+$树中,定义一个有 m 个关键字的内部结点必有 $m+1$ 个指针. 考虑层次 L 中的两个结点 M 和 N,它们的结构如图 2 所示,其中结点 N 是结点 M 的右兄弟结点.

设待检索的关键字为 K. 对层次 L 的顺序检索,首先是从相应的结点头中找到该层次中的第 1 个结点,然后在结

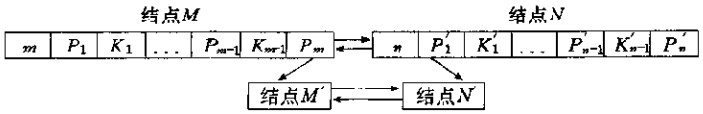


图2 RFN-B$^+$树中指针导航的二义性

点内部进行查找.对于结点 M,如果存在 i,$1\leq i\leq m-1$,$K_{i-1}<K\leq K_i$($K_0=-\infty$),那么,我们只要沿着指针 P_i 访问下一层中的相应结点就可以了.但是,当 $K>K_{m-1}$,而结点 M 又有右兄弟结点 N 时,我们就不能肯定沿着指针 P_m 就是正确的检索路径.因为对于结点 N,如果 $K\leq K'_1$,则沿着指针 P'_1 也可能是一条正确的检索路径.于是就产生了二义性.

对于二义性的情形,设在层次 L 的下面一个层次 L' 中,P_m 所指向的结点为 M',P'_1 所指向的结点为 N'.如果在层次 L' 中二义性仍然存在,那么,二义性只能由结点 M' 和结点 N' 产生.如果在对层次 L 进行顺序检索的过程中没有发现二义性,那么,在对层次 L 以下各层次的随机检索过程中,都不需要二义性检查.

6.2 算法的描述

在以某一个层次 L 作为虚拟根结点的随机检索算法1中,调用了一个函数 search(M,K).search(M,K)的主要功能是在结点 M 内查找关键字 K,并返回相应的指针.对于 M 是叶结点的情形,如果 M 中所有关键字的值都小于 K,则 search(M,K)返回一个空指针.否则,无论对于内部结点还是叶结点,search(M,K)都将返回一个非空的指针,该指针指示了在下一个层次中将要检索的结点.同时,在算法1中,还考虑了层次 L 就是叶结点所在层次的情形.用类C语言的算法语言来描述,该算法的主要思想如下.

算法1. 以一个层次 L 作为虚拟根结点的随机检索算法.

```
输入:一棵 RFN-B+树及其层次 L,待检索的关键字 K
输出:对关键字 K 进行随机检索的结果
算法的描述:
(1) 根据层次 L 所对应的层次头中的信息,将层次 L 中的第1个结点读出来,并且存放到 M;
(2) CheckAmbiguity=TRUE; /* 用于指示是否应该进行二义性检查 */
(3) AmbiguityFound=FALSE; /* 用于指示是否发现了二义性 */
(4) p=search(M,K);
(5) if (M 是叶结点)
      if (p 不是一个空指针,或者,
          CheckAmbiguity 的值为 FALSE,或者,
          M 没有右兄弟结点 N)
        return 检索的结果;
      else /* p 是一个空指针,并且,
              M 有右兄弟结点 N 要进行二义性检查 */
      if (AmbiguityFound 的值为 TRUE)
      {p'=search(N,K);
        return 检索的结果;} /* 根据引理 5.2 */
      else /* 叶结点所在的层次为虚拟根结点 */
      {M=M 的右兄弟结点 N;
       转(4);}
(6) if (M 是内部结点)
      if (p 不是 M 中最大的指针,或者,
          CheckAmbiguity 的值为 FALSE,或者,
          M 没有右兄弟结点 N)
      {AmbiguityFound=FALSE; /* 根据引理 5.1 */
        CheckAmbiguity=FALSE; /* 根据引理 5.3 */
        M=指针 p 所指向的结点;
        转(4);}
      else /* p 是 M 中最大的指针,并且,
              M 有右兄弟结点 N 要进行二义性检查 */
      {p'=search(N,K);
        if (p'是 N 中最小的指针)
        {AmbiguityFound=TRUE;
          M=指针 p 所指向的结点;
          转(4);}
        else if (p'不是 N 中最大的指针)
        {AmbiguityFound=FALSE; /* 根据引理 5.1 */
```

```
  CheckAmbiguity=FALSE; /* 根据引理 5.3 */
  M=指针 p′所指向的结点;
  转(4);}
else /* p′是 N 中最大的指针 */
if (AmbiguityFound 的值为 TRUE)
{AmbiguityFound=FALSE; /* 根据引理 5.1 和引理 5.2 */
  CheckAmbiguity=FALSE; /* 根据引理 5.3 */
  M=指针 p′所指向的结点;
  转(4);}
else /* 结点 M 所在的层次作为虚拟根结点 */
{M=N;
  p=p′;
  转(6);}}
```

7　重构结点数据块的算法

如果由于磁盘介质故障，层次 L 中的某一个结点数据块 N 遭到了破坏. 一般情况下，磁盘上的介质故障是难以直接修复的. 需要申请一个新的结点数据块 N'，在 N' 中重新组织关键字和指针信息，并将 N' 重新链接到原来 N 所在的位置.

下面只给出重构一个结点数据块的算法. 如果某一个层次中的结点数据块全部需要重构，可以调用此算法逐个地构造.

重构结点数据块的算法如算法 2 所示. 在算法 2 中的一些符号及其含义如图 3 所示. 首先应该注意，我们讨论的前提是：层次 L 是一个内部结点所在的层次，并且，虽然层次 L 中的结点数据块 N 遭到了破坏，但是层次 L 的上一个层次 L''以及下一个层次 L'中的数据块全都保持完好无损.

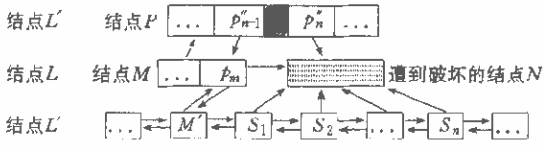


图3　重构RFN-B$^+$树中遭到破坏的结点数据块

结点 N 的父母结点为 P. 在结点 P 中，指向结点 N 的指针为 p''_n. 如果在结点 P 中，结点 N 的左兄弟结点 M 存在，那么，指向结点 M 的指针为 p''_{n-1}. 结点 M 中最大的指针为 p_m，指针 p_m 所指向的子女结点为 M'. M'的右兄弟结点 S_1，就是遭到破坏的结点 N 的第 1 个子女结点. 而且，对于结点 N 的所有子女结点，它们指向父母结点的指针都相等. 这也是在重构结点数据块时将要用到的一个重要的性质.

在算法 2 中调用了一个函数 maxkey(M). maxkey(M)的主要功能是返回以结点 M 为根的子树中最大的关键字. maxkey(M)的执行过程，主要是沿着结点中的最大指针一直搜索到叶结点，并返回该叶结点中最大的关键字.

在算法 2 中，还考虑到了结点 N 本身就是根结点的情形. 用类 C 语言的算法语言来描述，该算法的主要思想如下：

算法 2. 重构结点数据块的算法.

输入：一棵 RFN-B$^+$树；层次 L，层次 L 中遭到破坏的结点数据块 N；如果结点 N 不是根结点，则还有结点 N 的父母结点 P 以及结点 P 中指向结点 N 的指针 p''_n.

输出：重新构造的结点数据块 N'，并且结点 N'取代了结点 N 而链接在输入的 RFN-B$^+$树中.

算法的描述：

```
(1) 申请一个新的结点数据块 N′;
(2) if (结点 N 是根结点,或者,
        (指针 p″n 是结点 P 中最小的指针,并且结点 P 没有左兄弟结点 Q))
      读出层次 L′中的第 1 个结点,并且存放到 S1;
    else /* 结点 N 不是根结点 */
    {if (指针 p″n 不是结点 P 中最小的指针)
       读出指针 p″n-1所指向的结点,并且存放到 M;
     else /* 指针 p″n 是结点 P 中最小的指针,并且 P 有左兄弟结点 Q */
       {读出结点 P 的左兄弟结点 Q;
        取出 Q 中最大的指针 pq;
        读出指针 pq 所指向的结点,并且存放到 M;}
     取出 M 中最大的指针 pm;
     读出指针 pm 所指向的结点,并且存放到 M′;
```

```
        读出M'的右兄弟结点,并且存放到S1;}
(3) n=0; /* n用于对结点N'中将要包含的指针数目进行计数 */
(4) p=结点S1的地址;
    n++;
(5) if (S1没有右兄弟结点S2)
      转(6);
    else if (S1的指向父母结点的指针等于S2的指向父母结点的指针)
    {K=maxkey(S1);
      将(p,K)添加到结点N';
      S1的指向父母结点的指针改为指向结点N';
      S1=S2;
      转(4);}
    else /* S1是原来的结点N的最后一个子女结点 */
      转(6);
(6) 将指针p添加到结点N';
    S1的指向父母结点的指针改为指向结点N';
    结点N'的指针数目=n;
    if (结点N是根结点,或者,
        (指针p''n是结点P中最小的指针,并且结点P没有左兄弟结点Q))
    {层次L的层次头中指向第1个结点数据块的指针=结点N';
      结点N'的指向左兄弟结点的指针=NULL;}
    else /* 原来结点N的左兄弟结点为M' */
    {结点M'的指向右兄弟结点的指针=N';
      结点N'的指向左兄弟结点的指针=M';}
    if (结点N是根结点)
    {
      结点N'的指向右兄弟结点的指针=NULL;}
    else /* 结点N不是根结点 */
    {将结点P中的指针p''n改为指向结点N';
      结点N'的指向父母结点的指针=P;
      if (指针p''n是结点P中最大的指针,并且P没有右兄弟结点O)
        结点N'的指向右兄弟结点的指针=NULL;
      else
      {if (p''n不是结点P中最大的指针)
        p=p'n右边的指针;
      else
        p=结点O中最小的指针;
      R=指针p所指向的结点;
      结点N'的指向右兄弟结点的指针=R;
      结点R的指向左兄弟结点的指针=N';}}
```

8 结束语

由于RFN-B$^+$树增加了B$^+$树构造的复杂性,特别是要维护RFN-B$^+$树的全链接结构,势必增加插入和删除算法的时间,但时间复杂度的量级不变.相对于RFN B$^+$树带来的较高的空间效率和有效性,增加的时间复杂度还是值得的.

参考文献

1 姚卿达,肖永侨,陈晓蘅. LNDBMS:一个高效的面向数据采集的数据库管理系统. 软件学报,1996,7(增刊):254～260
(Yao Qing-da, Xiao Yong-qiao, Chen Xiao-heng. LNDBMS: a data collection oriented DBMS with high performance. Journal of Software, 1996,7(supplement):254～260)

2 许卓群,张乃孝,杨冬青等. 数据结构,北京:高等教育出版社,1991
(Xu Zhuo-qun, Zhang Nai-xiao, Yang Dong-qing *et al*. Data Structure. Beijing: Higher Education Publishing House, 1991)

3 严蔚敏,吴伟明. 数据结构,北京:清华大学出版社,1992
(Yan Wei-min, Wu Wei-ming. Data Structure. Beijing: Tsinghua University Publishing House, 1992)

4 Yao S B, DeJong D. Evaluation of database access paths. In: Proceedings of the 1978 ACM SIGMOD Conference on the Management of Data. Austin, Texas, USA, 1978

5 Yao A. On random 2-3 trees. Acta Informatica, 1978,9:159～170

6 Gray J. The transaction concept: virtues and limitations. In: Proceedings of IEEE the 7th International Conference on Very Large Data Bases. Cannes, France, 1981

RFN-B⁺-Tree Index File and Its Availability

YAO Qing-da　YANG Gui-zhen　ZHANG Jun-xin

(*School of Information Science and Technology　Zhongshan University　Guangzhou*　510275)

Abstract　Based on the comparison of the conventional definitions of B-tree and B⁺-tree, and their manipulating algorithms, a new definition of B⁺-tree: RFN-B⁺-tree is presented in order to achieve higher space efficiency and higher availability. Its space efficiency compared with conventional B⁺-tree is analyzed firstly. Then the availability of RFN-B⁺-tree index file in terms of its full-link pointer structure together with two modules that support this availability: random access algorithm based on virtual root node, and node restructuring algorithm are discussed.

Key words　B⁺-tree, index, availability, algorithm.