

# SimpleDB Lab3 实验报告

2010239 李思凡

## 一、Lab3 概述

通过 lab3，将主要实现 B+数索引用于有效地查找和范围扫描。在本次实验中需要实现查找（`findLeafPage`）、插入时的分裂节点操作（`splitLeafPage`、`splitInternalPage`）、删除节点时的从左右节点取元素（`stealFromLeafPage`、`stealFromLeftInternalPage`、`stealFromRightInternalPage`）和合并两个节点（`mergeLeafPages`、`mergeInternalPages`）。

首先简单描述一下 B+树应该遵循的结构以及本实验中建立的 B+树索引的特点。一个  $n$  阶 B+树的每个节点至多有  $n$  个 child 节点，非根节点中值的个数为  $\frac{n}{2} \leq num \leq n - 1$ ，相邻叶子节点通过指针双向连接。本实验中，B+树用于索引，所有内部节点都是索引值，一个 `BtreeEntry` 的对象包括一个值和它的左孩子引用和右孩子引用；叶子节点保存数据，为该表中的元组。一个节点中的值可以通过正向迭代器或反向迭代器获取。内部节点与其 `parent` 节点值不能相同，叶子节点与其 `parent` 节点的值可以相同。

## 二、Git Commit History

08 May, 2022 1 commit		
Chris Lab3	Christine-0107 authored 1 day ago	428825fa
First Commit: delete		
07 May, 2022 1 commit		
Chris Lab3	Christine-0107 authored 1 day ago	3abb043f
First Commit: insert		
03 May, 2022 1 commit		
Chris Lab3	Christine-0107 authored 5 days ago	4da1131d
First Commit: findLeafPage		

## 三、查找（Search）

实现 `BtreeFile.java` 中的 `findLeafPage()`方法。

1.设计思路：

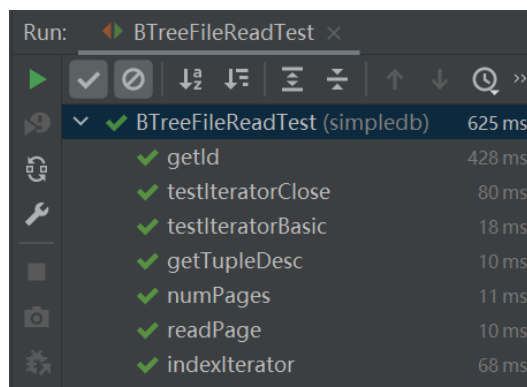
B+树中有 4 种类型的页，BTreeLeafPage 的页对象表示叶子节点，BtreeInternalPage 的页对象表示内部节点，此外还有 BtreeRootPtrPage 对象指向根节点，以及 BtreeHeaderPage 对象记录哪些页在被使用。在 findLeafPage()方法中，需要根据给定值寻找它所在叶子节点表示的页，若不存在该值，需要返回它能够插入的叶子节点。

## 2.重难点:

函数是一个递归调用的过程。若当前页的类型是 LEAF 类型，则直接通过 BtreeFile 类中定义的 getPage()方法来取这一页，许可类型为传进来的 perm 参数。若当前页为内部节点，通过只读的方式获取该页，获取它的迭代器。若迭代器的下一个元素存在，循环执行，判断所查找的 key 值是否为空，如果为空，则均递归归到其左侧孩子，最终返回最左侧叶子节点；如果不为空，来比较当前 entry 的值和要查找的值的大小，若当前值更大，则递归查找其左孩子节点。若迭代器中已无下一个元素说明该值比当前节点所有值都大，递归查找其右孩子结点。

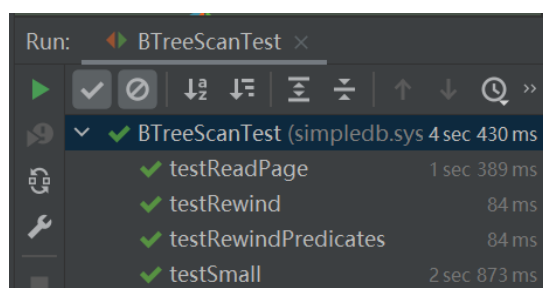
## 3.改动部分：未改动 API 和测试代码。

## 4.单元测试结果:



Run:	BTreeFileReadTest	
✓	BTreeFileReadTest (simplifiedb)	625 ms
✓	getld	428 ms
✓	testIteratorClose	80 ms
✓	testIteratorBasic	18 ms
✓	getTupleDesc	10 ms
✓	numPages	11 ms
✓	readPage	10 ms
✓	indexIterator	68 ms

## 系统测试结果:



Run:	BTreeScanTest	
✓	BTreeScanTest (simplifiedb.sys)	4 sec 430 ms
✓	testReadPage	1 sec 389 ms
✓	testRewind	84 ms
✓	testRewindPredicates	84 ms
✓	testSmall	2 sec 873 ms

## 四、插入

实现 BtreeFile.java 中 splitLeafPage()、splitInternalPage()方法。实验中 B+树的插入过程为：

首先调用 findLeafPage()方法找到要插入的节点，并调用相应页对应的插入函数进行插入。接着判断此时节点中个数和最大个数  $n$  的关系，若未超过最大，则插入结束；否则要将该节点分裂为两个，左侧个数为  $\text{floor}(n/2)$ ，右侧为  $\text{ceil}(n/2)$ 。对于叶子节点，需要将右侧第一个节点的值“复制”到其 parent 节点中；对于内部节点，则是将右侧第一个节点的值“剪切”到其 parent 节点中。

此时要再判断 parent 节点中值的个数与最大个数的关系，若未超过最大个数，插入结束；否则递归进行分裂，直到所有节点均满足 B+树的结构。

注意，如果原来的根节点产生分裂，生成了新的根节点，需要改变 root 引用的指向。对于所有被影响的节点表示的页，都要将它们放到 dirtypages 列表中。

### （一）splitLeafPage()

#### 1.设计思路：

（1）调用 getEmptyPage()方法，在当前页的右侧添加一个新的页（节点）。

（2）获取当前页的反向迭代器，目的是将后一半元素移动到新的页中。设立一个存放右侧元组的数组，通过迭代器遍历将它们放到数组里。遍历数组，调用 deleteTuple()方法，将各个元组从当前页中删除，新页调用 insertTuple()方法将它们插入。

（3）获取右侧页的第一个值 midKey，调用 getParentWithEmptySlots()方法找到该页的 parent 节点，传入 midKey 用于 parent 节点也需要分裂的情况。

（4）更新因为新增一个节点后带来的左右引用指向的变化。

（5）更新当前页和新页的 parent 引用，让他们都指向（3）中获取的 parent 节点。

（6）为 midKey 设置一个 BtreeEntry 对象，左右孩子引用分别为当前页和新页，并将该对象插入 parent 节点中。

（7）将有改动的页设置为 dirtypages。

（8）比较新插入的值和中间值，确定插入的页是当前页还是右侧页，返回。

#### 2.重难点：

(1) 将右半部分元组移动到新页时，需要先删除再插入。因为插入会导致 Id 的变化，先插入会导致无法在当前页正常删除。

(2) 叶节点分裂时，中值是“复制”到 parent 中。中值插入 parent 节点时，需要为它设置左右引用，通过新创建 BtreeEntry 对象实现。

## (二) splitInternalPage()

### 1. 设计思路：

整体思路与 splitLeafPage()大致相同，不同点见重难点。

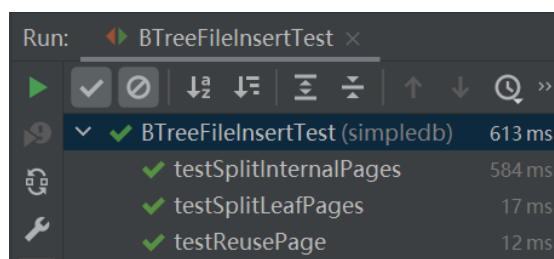
### 2. 重难点：

(1) 将当前页中后一半的 entry 移动到右侧新页时，中间的那个 entry 只从当前页删除，但不添加到新页。获取它的 key，将它的左右孩子引用分别设置为当前页和右侧新页，将它插入到 parent 节点中。

(2) 由于是内部节点，节点不止有 parent 引用需要更新，右半部分被移动的 entry 的孩子节点的 parent 引用也需要更新。调用 updateParentPointers()方法可以方便地实现。

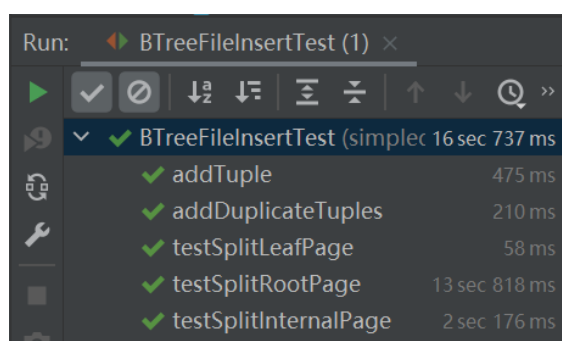
本部分未改动 API 和测试代码。

单元测试结果：



Run: BTreeFileInsertTest x	
✓	BTreeFileInsertTest (simplifiedb) 613 ms
✓	testSplitInternalPages 584 ms
✓	testSplitLeafPages 17 ms
✓	testReusePage 12 ms

系统测试结果：



Run: BTreeFileInsertTest (1) x	
✓	BTreeFileInsertTest (simplec 16 sec 737 ms
✓	addTuple 475 ms
✓	addDuplicateTuples 210 ms
✓	testSplitLeafPage 58 ms
✓	testSplitRootPage 13 sec 818 ms
✓	testSplitInternalPage 2 sec 176 ms

## 五、删除

实现 BtreeFile 中的 `stealFromLeafPage()`、`stealFromLeftInternalPage()`、`stealFromRightInternalPage()`、`mergeLeafPages()`、`mergeInternalPages()`方法。B+树种的删除过程如下：

节点容纳的值的的最小数量为  $\text{ceil}(n/2)-1$ ，若删除后的数量小于该值，且其左右节点中存在大于最小值的节点，则将多余的量分给当前节点一半。对于叶子节点，更新其 `parent` 中的 `entry`；对于内部节点，将 `parent` 中的 `entry` 值拉下来，相邻两个节点中其中一个值上去。若左右节点均只达到最小数量，则需要将当前节点和其中一个相邻节点合并。对于叶子节点，删除 `entry`；对于内部节点，将该 `entry` 拉下来。

### （一）`stealFromLeafPage()`

#### 1.设计思路：

（1）根据 `isSiblingRight` 参数判断相邻页在左侧还是右侧。若在左侧则采用反向迭代器，若在右侧则采用正向迭代器。

（2）将当前页的相邻页多的元组的一半存到一个元组数组里。从相邻页中删除，再在当前页添加。

（3）重新设置 `parent` 中相应 `entry` 的值，为右侧页的第一个元素的值。将该 `entry` 在 `parent` 中更新。

#### 2.重难点：

在更新 `parent` 的时候需要用到右侧页的首元，所以需要新建一个页表示右侧页，在使用 `isSiblingRight` 时为它赋值。

### （二）`stealFromLeftInternalPage()`

#### 1.设计思路：

（1）采用左侧页的反向迭代器，将多出来的一半元素存在 `BtreeEntry` 数组里。先将 `parent` 中的 `entry` 的值插入到当前页，为其设置新的左右引用。再将数组中的元素（除了下标为 0 的元素）从左侧页删除，插入到当前页中。下标为 0 的元素仅从左侧页删除，不添加到右侧页。

（2）将下标为 0 的元素的值设置到 `parentEntry` 中，更新 `parent`。

（3）更新被移动节点的孩子节点的 `parent`，更新为 `page`，采用

updateParentPointers()。

(4) 将有改动的页设置为脏页。

## 2.重难点:

将原来 parentEntry 中的值拿下来时, 需要为设立一个新的 BtreeEntry 来存这个值。此时左孩子应该指向 leftSibling 页最右边 entry 的右孩子, 右孩子应该指向 page 页最左边 entry 的左孩子。

### (三) stealFromRightInternalPage()

与 stealFromLeftInternalPage()方法的设计思路相同, 操作上是“对称”的。

### (四) mergeLeafPages()

设计思路:

(1) 设立一个元组数组, 通过右侧页的正向迭代器取右侧页的所有元素, 存在数组中。将右侧页这些元素删除, 插入到左侧页中。

(2) 更新因为删除一个节点后带来的左右引用指向的变化。

(3) 通过 setEmptyPage()方法将右侧页置为空。

(4) 通过 deleteParentEntry()方法删除 parent 中相应的 parentEntry。

(5) 将左侧页和 parent 页添加到脏页列表中。

### (五) mergeInternalPages()

1.设计思路:

(1) 将 parent 中的 parentEntry 删除, 添加到左侧页中。

(2) 与 mergeLeafPages()方法相同, 将右侧页中的所有元素移动到左侧中, 将右侧页置空。

(3) 更新所有移动的节点的孩子节点的 parent。

(4) 将左侧页和 parent 页添加到脏页列表中。

## 2.重难点:

将原来 parentEntry 中的值拿下来时, 需要为设立一个新的 BtreeEntry 来存这个值。此时左孩子应该指向左侧页最右边 entry 的右孩子, 右孩子应该指向右侧页最左边 entry 的左孩子。

本部分没有改动 API 和测试代码。

单元测试结果如下：

Run: BTreeFileDeleteTest

Test Case	Duration (ms)	Status
BTreeFileDeleteTest (simplifiedb)	950	Pass
deleteTuple	413	Pass
testStealFromLeftLeafPage	20	Pass
testMergeLeafPages	30	Pass
testStealFromLeftInternalPage	173	Pass
testStealFromRightLeafPage	10	Pass
testStealFromRightInternalPage	140	Pass
testMergeInternalPages	164	Pass

系统测试结果如下：

The screenshot shows the Visual Studio Test Explorer interface. At the top, it says "Run:" followed by a red arrow icon and the test name "BTreeFileDeleteTest (1)". Below this is a toolbar with icons for running tests (green play button), passing/failing status (checkmark, X), and various sorting options (arrows). The main area displays a tree view of the test results:

- BTreeFileDeleteTest** (simple 17 sec 166 ms) - Passed (green checkmark)
  - testDeleteRootPage** - 442 ms - Passed (green checkmark)
  - testReuseDeletedPages** - 160 ms - Passed (green checkmark)
  - testRedistributeLeafPages** - 50 ms - Passed (green checkmark)
  - testMergeLeafPages** - 80 ms - Passed (green checkmark)
  - testRedistributeInternal** - 14 sec 712 ms - Passed (green checkmark)
  - testDeleteInternalPages** - 1 sec 722 ms - Passed (green checkmark)