实验5：B+树索引实现

邹兆年，哈尔滨工业大学计算学部，znzou@hit.edu.cn

一、实验目的

1. 掌握B+树查找操作的实现方法。
2. 掌握B+树插入操作的实现方法。
3. 掌握B+树删除操作的实现方法。

二、相关知识

1. B+树
2. 并发编程

三、实验内容

本实验包括4项任务。

任务0：了解B+树数据结构的设计

（1）阅读代码

阅读`src/index`目录下的代码。

- `src/index/ix_defs.h`
- `src/index/ix_index_handle.h`
- `src/index/ix_index_handle.cpp`

了解`IxFileHdr`、`IxPageHdr`、`IxNodeHandle`和`IxIndexHandle`类的设计，并回答下列问题：

- `IxNodeHandle`类与B+树节点是什么关系?
- `IxIndexHandle`类与B+树是什么关系?
- `IxPageHdr`类与B+树节点是什么关系?
- `IxFileHdr`类与B+树是什么关系?

（2）设计的特殊性

本实验的B+树具有如下特殊性：

- 本实验的B+树索引是唯一索引（unique index），该索引不支持重复的键。
- 该B+树中任意节点能容纳的键值对数量小于最大值，大于等于最小值。这相当于留出了一个多余的空位，方便B+树进行插入和删除操作。
- 在逻辑上，B+树非叶节点中指针的数量比键的数量多1。在物理设计上，令B+树非叶节点中指针的数量与键的数量相等。设非叶节点N的n个儿子，$p_0, p_1, \ldots, p_n$是指向这n个儿子的指针，则非叶节点N亦有n个键$k_0, k_1, \ldots, k_n$。对任意$i = 1, 2, \ldots, n$，k_i左侧的儿子指针是p_{i-1}，k_i右侧的儿子指针是p_i。k_0有什么

用呢？k_0等于p_0指向的N的最左儿子中存储的第一个键，即k_0等于以N的最左儿子为根的子树中存储的最小键。因此，当分裂或合并节点N时，可能需要更新k_0的值。

（3）重要的辅助函数

`ix_index_handle.cpp`文件中提供了一些已经实现好的辅助函数，对本实验非常重要，可直接调用。

`IxNodeHandle`类的辅助函数

```
class IxNodeHandle {
  // 辅助函数（本实验提供，无需实现）
  char *get_key(int key_idx) const;
  Rid *get_rid(int rid_idx) const;
}
```

- `get_key`函数返回当前节点键数组中指定位置`key_idx`的键的地址。
- `get_rid`函数返回当前节点键数组中指定位置`rid_idx`的值的地址。

`IxIndexHandle`类的辅助函数

```
class IxIndexHandle {
  // 辅助函数（本实验提供，无需实现）
  IxNodeHandle *fetch_node(int page_no) const;
  IxNodeHandle *create_node();
  void maintain_parent(IxNodeHandle *node);
  void maintain_child(IxNodeHandle *node, int child_idx);
  void erase_leaf(IxNodeHandle *leaf);
  void release_node_handle(IxNodeHandle &node);
}
```

- `fetch_node`函数获取页号为`pange_no`的B+树节点对应的`IxNodeHandle`对象指针。
- `create_node`函数创建一个`IxNodeHandle`对象，并返回其指针。
- `maintain_parent`函数从`node`节点开始更新其父节点的第一个key，一直向上更新直到根节点。
- `maintain_child`函数将`node`节点的第`child_idx`个儿子节点的父节点指针置为`node`对应节点。
- `erase_leaf`函数用于在删除叶节点`leaf`前更新其前驱节点的`next_leaf`指针和后继节点的`prev_leaf`指针。
- `release_node_handle`函数用于在删除节点`node`后更新索引文件头记录的页面个数。

```
int ix_compare(const char *a, const char *b, ColType type, int col_len);
```

- `ix_compare`函数比较两个键`a`和`b`的大小，`a`和`b`的长度均为`col_len`。`type`表示`a`和`b`的类型（`int*`/`float*`/`char*`）。

任务1：B+树查找操作的实现

补全`IxNodeHandle`类的部分函数，实现B+树上的查找操作，具体完成下列任务。

（1）实现`IxNodeHandle::lower_bound`函数

函数声明：

```
int IxNodeHandle::lower_bound(const char *target) const;
```

功能：

- 该函数适用于任何类型B+树节点。
- 在当前节点中查找第一个大于或等于`target`的索引键值所在的位置。思考：该函数返回值的值域是什么？
- 如果`target`大于当前节点中最后一个索引键值，则返回当前节点中存储的索引键值数量（该数量存储在`page_hdr->num_key`中）。思考：`page_hdr`的作用是什么？
- 例：如果索引键值数组为$[1, 3, 5, 7, 11]$，则`lower_bound(3)`的返回结果为1，`lower_bound(13)`的返回结果为5。

实现：

- 参考代码注释。基本实现逻辑如下：使用二分查找。思考：为什么可以使用二分查找？
- 获得第i个索引键值需要调用`IxNodeHandle::get_key()`函数。
- 比较键值大小时需要调用`ix_compare()`函数。

（2）实现`IxNodeHandle::upper_bound`函数

函数声明：

```
int IxNodeHandle::upper_bound(const char *target) const;
```

功能：

- 该函数适用于任何类型B+树节点。
- 在当前节点中查找第一个大于`target`的索引键值所在的位置。思考：该函数返回值的值域是什么？
- 如果`target`大于等于当前节点中最后一个索引键值，则返回当前节点中存储的索引键值数量（该数量存储在`page_hdr->num_key`中）。
- 例：如果索引键值数组为$[1, 3, 5, 7, 11]$，则`upper_bound(3)`的返回结果为2，`upper_bound(13)`的返回结果为5。

实现：

- 参考代码注释。基本实现逻辑如下：使用二分查找。
- 获得第i个索引键值需要调用`IxNodeHandle::get_key()`函数。
- 比较键值大小时需要调用`ix_compare()`函数。

（3）实现`IxNodeHandle::leaf_lookup`函数

函数声明：

```
bool IxNodeHandle::leaf_lookup(const char *key, Rid **value);
```

功能：

- 该函数只适用于B+树叶节点。思考：如何判断一个B+树节点是叶节点？提示：借助`page_hdr`。
- 如果当前叶节点中存在等于`key`的索引键值，则函数返回`true`，并且输出参数`value`的值赋值为与该索引键值对应的元组rid指针；否则，函数返回`false`。

实现：

- 参考代码注释。
- 查找等于`key`的索引键值时可以调用`IxNodeHandle::lower_bound`函数。思考：如何根据`IxNodeHandle::lower_bound`的返回结果判断当前叶节点中存在等于`key`的索引键值？
- 获取第i个rid需要调用`IxNodeHandle::get_rid()`函数，其返回值为`Rid*`类型。

（4）实现`IxNodeHandle::internal_lookup`函数

函数声明：

```
page_id_t IxNodeHandle::internal_lookup(const char *key);
```

功能：

- 该函数只适用于B+树的非叶节点。思考：如何判断一个B+树节点不是叶节点？提示：借助`page_hdr`。
- 查找索引键值`key`在当前节点的哪个儿子节点N作为根的子树中。函数返回值为儿子节点N所在页的页号。思考：儿子节点N所在页的页号存储在哪里？提示：分析`Rid`类型的构成，思考其中域`pange_no`代表什么。
- 非叶节点中第i个索引键值k_i右边的儿子节点做为根的子树中存储的索引键值均大于等于k_i；第i个索引键值k_i左边的儿子节点做为根的子树中存储的索引键值均小于k_i。

实现：

- 参考代码注释。
- 查找大于`key`的索引键值时可以调用`IxNodeHandle::upper_bound`函数。思考：如何
- 获取儿子节点的页号需要调用`IxNodeHandle::get_rid()`函数，其返回值为`Rid*`类型。

（5）实现`IxIndexHandle::find_leaf_page`函数

函数声明：

```
std::pair<IxNodeHandle *, bool> IxIndexHandle::find_leaf_page(const char *key,
Operation operation, Transaction *transaction, bool find_first);
```

功能：

- 该函数作用于整个B+树上。
- 该函数在B+树上查找索引键值等于`key`所在的叶节点。如果存在，则返回`true`，以及该叶节点（以该节点的`IxNodeHandle`对象指针形式返回）；否则，返回`false`。
- `operation`表示上层调用该函数时执行的是何种操作（查找、插入、删除），因为查找、插入和删除操作均要先查找叶节点。不同操作下需要根据`operation`来确定对哪些节点加锁。
- `transaction`可以暂时忽略。
- 因为是要实现唯一索引，所以`find_first`默认为`false`，可以暂时忽略。

实现：

- 参考代码注释。
- 从B+树根节点开始，不断向下查找并访问儿子节点，直到找到包含索引键值`key`的叶节点。
- 在每个非叶节点上，先调用`IxNodeHandle::internal_lookup`函数找到接下来要访问的儿子节点的页号p，然后调用`IxIndexHandle::fetch_node`函数将页号为p的页读入缓冲池，并返回该儿子节点的`IxNodeHandle`对象指针。

（6）单元测试

与任务2在一起进行单元测试。

任务2：B+树插入操作的实现

补全`IxNodeHandle`类的部分函数，实现B+树上的插入操作，具体完成下列任务。

（1）实现`IxNodeHandle::insert_pairs`函数

函数声明：

```
void IxNodeHandle::insert_pairs(int pos, const char *key, const Rid *rid, int n);
```

功能：

- 该函数适用于任何类型B+树节点。
- 该函数在当前节点的指定位置`pos`插入`n`个键值对。`n`个键值对中的键（`char*`类型）存储在`key`数组中，`n`个键值对中的值（`Rid`类型）存储在`rid`数组中。思考：数组`key`中每个键值的长度是多少字节？提示：查看`file_hdr`中存储的B+树元数据。

实现：

- 参考代码注释。
- 注意：在插入`n`个键值对之前，需要先将节点中原来存储在`pos`位置及其后面的键和值均向后移动，为插入`n`个键值对腾出空间。调用`memcpy()`和`memmove()`进行数据移动。思考：需要将键向后移动多少个字节？需要将值向后移动多少个字节？

（2）实现`IxNodeHandle::insert`函数

函数声明：

```
int IxNodeHandle::insert(const char *key, const Rid &value);
```

功能：

- 该函数适用于任何类型B+树节点。
- 该函数在当前节点插入1个键值对`key:value`，并返回插入后节点中包含的键值对数量。
- 约束条件：保证插入后键值对`key:value`后，节点中的键数组仍然有序。
- 注意：因为是唯一索引，所以重复的`key`不插入。

实现：

- 参考代码注释。
- 先调用`IxNodeHandle::lower_bound`函数找到插入键值对`key:value`的位置`pos`，然后调用`IxNodeHandle::insert_pairs`函数在位置`pos`插入该键值对。

（3）实现`IxIndexHandle::insert_entry`函数

函数声明：

```
page_id_t IxIndexHandle::insert_entry(const char *key, const Rid &value, Transaction
*transaction);
```

功能：

- 该函数作用于B+树，是B+树对外的接口。
- 该函数将键值对`key:value`插入到B+树叶节点中，并返回该叶节点的页号。
- `transaction`参数可以忽略。

实现：

- 参考代码注释。
- 实现该函数时需要调用函数`IxIndexHandle::find_leaf_page()`、`IxIndexHandle::insert()`、`IxIndexHandle::split()`、`IxIndexHandle::insert_into_parent()`。

（4）实现`IxIndexHandle::split`函数

函数声明：

```
IxNodeHandle *IxIndexHandle::split(IxNodeHandle *node);
```

功能：

- 该函数分裂节点`node`，并返回新分裂出的节点的`IxNodeHandle`对象指针。

- 约束条件：如果`node`是叶节点，则需要更新`node`和新分裂出的叶节点的兄弟指针（节点页号），使B+树全部叶节点形成一个链表。如果`node`不是叶节点，则需要更新分裂出的节点的父节点指针（节点页号）。

实现：参考代码注释。

（5）实现`IxIndexHandle::insert_into_parent`函数

函数声明：

```
void IxIndexHandle::insert_into_parent(IxNodeHandle *old_node, const char *key,
IxNodeHandle *new_node, Transaction *transaction);
```

功能：

- 当旧节点`old_node`被分裂，形成新节点`new_node`后，调用该函数向旧节点`old_node`的父节点中插入键值对。

实现：

- 参考代码注释。
- 该函数可能会递归调用`IxIndexHandle::split()`和`IxIndexHandle::insert_into_parent()`函数。

（6）单元测试

单元测试代码在文件夹`src/test/index`中。

执行下列命令，进行单元测试。

```
cd build

make b_plus_tree_insert_test
./bin/b_plus_tree_insert_test
```

（7）注意事项

- 所有测试只调用`IxIndexHandle::get_value()`、`IxIndexHandle::insert_entry()`和`IxIndexHandle::delete_entry()`这三个函数。学生可以自行添加和修改辅助函数，但不能修改以上三个函数的声明，否则测试程序无法执行。
- 在测试前，学生还需完成`src/system/sm_manager.cpp`文件中的`SmManager::create_index()`函数，方可进行测试。

任务3：B+树删除操作的实现

（6）单元测试

单元测试代码在文件夹`src/test/index`中。

执行下列命令，进行单元测试。

```
cd build

make b_plus_tree_delete_test
./bin/b_plus_tree_delete_test
```