中国人民大学信息学院

数据库系统原理与实现

实验二《索引》实验报告

小组成员：张恒、陶友贤、赵旺、修晔良

目录

1 实验内容

- 索引的初始建立
- 索引的维护（增、删、改）
- 索引的查找

2 实验要求

2.1 索引结构

至少实现以下索引中的一种：

- B+ 树
- 线性 Hash
- 可扩展 Hash

2.2 外存索引

- 实现的索引是外存索引。（数据地址可用逻辑地址模拟）

2.3 索引展示

- 图形化地展示索引结构（Web UI、客户端、本地文件或命令行等）

3 实验分工安排

任务名称	人员
系统设计	
程序实现	
系统测试	
文档编写	
构图设计	
文档修改	

4 实验环境

- 操作系统平台：windows 10/win 8/win 7
- 编程语言：C 语言
- 程序开发环境：Visual Studio 2013

5 实验原理

5.1 B+树定义与特点

B+树是一种树数据结构， B+树都有一个参数 n，它决定了 B+树的所有存储块的布局，每个存储块存放 n 个查找键和 n+1 个指针， B+树把它的存储块组织成一棵树，这棵树是平衡的，即从树根到树叶的所有路径都一样长。通常 B+树有三层：根、中间层和叶，但也可以是任意多层。

- 根节点至少有两个指针被使用，所有指针指向位于 B+树下一层的存储块。

- 叶节点中的键值是排好序的，从左到右分布在叶节点中，叶节点中最后一个指针只指向它右边的下一个叶节点存储块，即指向下一个键值大于它的块，在叶块的其他 n 个指针当中，至少有(n+1)/2 向下取整个数个指针被使用指向数据记录，未使用的指针可看做空指针且不指向任何地方，如果第 i 个指针被使用，则指向具有第 i 个键值的记录。
- 内层节点中，所有的 n+1 个指针都可以用来指向 B+树中下一层的块，它们中至少有（n+1/2）向上取整个的指针被使用，如果 j 个指针被使用，那该块中将有 j-1 个键值。
- 所有被使用的键值和指针通常都存放在数据块的开头位置，叶节点第(n+1)指针是个例，它用来指向下一个叶节点。

5.2 B+树的操作

5.2.1 B+树的查找

假设有一棵 B+树索引，找出键值为 K 的记录，我们需要从根到叶递归查找，查找过程为：若我们处于叶节点上，就在其键值中查找，若第 i 个键值为K，则第 i 个指针可以让我们找到所需记录。如若我们处于某个内部节点，且它的键值为 K1，K2，...，Kn， 则只有一个子节点可使我们找到具有键值 K 的叶节点，如果 K<K1，则为第一个子节点， 如果 K1<=K<K2，则为第二个子节点.....在这一子节点上递归的运用查找过程。

5.2.2 B+树的插入

插入在原则上也是递归的，我们设法在适当的叶节点中为新键找到空闲空间，如果有的话，我们就把键放在那里，如果在适当的叶节点中没有空间，我们就把该叶节点分裂成两个并且其中的键分到这两个新节点中，是每一个新节点有一般厚刚好超过一般的键。某一层的节点分裂在其上一层看来相当于是要在这一较高的层次上插入一个新的键值对，因此，我们这可以在这一较高层次上递归的使用这个插入策略：如果有空间，则插入；如果没有，则分裂这个父节点且继续向树的高层推进。

例外的情况是：如果我们试图插入键到根节点中并且根节点没有空间，那么我们就分裂根节点成两个节点且在更上一层创建一个新的根节点，这个新的根节点有两个刚分裂的节点作为它的子节点。

5.2.3 B+树节点的分裂

假定 N 是一个容量为 n 个键值和 n+1 个指针的内部节点，并且由于下层几点的分裂N正好又被分配给第(n+2)个指针，则需执行：创建一个新节点 M，它将是N节点的兄弟且紧挨在 N 的右边；按排序将前(n+2)/2向上取整个的指针留在节点N中，把剩下的(n+2)/2向下取整个的指针移到节点M中；将n/2 向上取整个键保留在节点N中，而后n/2向下取整个键移动节点M中，注意，在中间的那个键总是被留出来，它既不在节点N中也不在节点M中，这一留出的键K指明通过M的第一个子节点可访问到最小键，尽管K不在出现在N中也不出现在 M 中，但它代表通过M能到达的最小键值，从这种意义上来说它与 M 相连，因此，K 将会被节点N和M的父节点用来划分这两个节点之间的查找。

5.2.4 B+树的删除

如果发生删除的 B+树节点在删除后至少还有最小数目的键和指针，那就不需要再做什么。但是，节点有可能在删除之前正好具有最小的充满度，因此在删除后，就违背了 键数目的约束，这时，我们需要为这个键的数目仅次于最小数目的节点 N 做下面两件事之一，其中有一种情况需要沿着树往上递归的删除。

如果与节点 N 相邻的兄弟中有一个键和指针超过了最小数目，那她的一个键指针对可以移到节点N 中并保持键的顺序，节点N 的父节点的键可能需要调整以反映这个新的情况。例如，如果节点 N 的右边的兄弟 M 可提供一个键指针对，那么从节点 M 移到节点 N 的键一定是节点M 的最小键，再节点 M 和节点N 的父节点有一个表示通过 M可访问到的最小键，必须增大该键值以

反映新的 M。

困难的情况适当相邻的两个兄弟节点中没有一个能提供键值给节点 N 时，不过，在这种情况下，我们有节点 N 和它的一个兄弟节点 M 这两个相邻接点，其中后者的键数刚好为最小数，而前者的键数小于最小数，因此，它们合在一起也没有超过单个节点所允许的键和指针数，我们合并这两个节点，实际就是删除它们中的一个，我们需要调整父节点的键，然后删除父节点的一个键和指针，如果父节点现在足够满，那么我们就完成了删除操作，否则，我们需要在父节点上递归的运用这个删除算法。

6 程序实现

6.1 结构体

定义几个常数：

```
#define MAX 5                    //最大节点数为 5
#define MIN 3                    //最小节点数为 3
#define NODE 0                   //内部节点为 0
#define LEAF 1                   //叶节点为 1
```

B+树节点上的记录：

```
typedef struct
{
    int key;                     //记录的键值，
    int pos;                     //记录的物理位置
}Record;
```

B+树的节点：

```
typedef struct
{
  int type;                  //节点在树中的角色，叶子节点，左孩子，右孩子。
  int count;
  Record pair[MAX];
  int parent;
}Node;
```

6.2 主要的函数

- **插入函数**

  实现功能：对于给定的一个“键值—指针对”，按照 B+树的定义，将其插入到规定的叶节点。

  函数声明：int insert(FILE *index, Record record);

- **查找函数**

  实现功能：找出给定的键值在 B+树中的位置

  函数声明：int search(FILE *index, int key);

- **删除函数**

  实现功能：对于给定的一个键值，若 B+树中存有该键值，则从B+树中删除该

键值及其相应的指针。

函数声明：int del(FILE *index, int key);

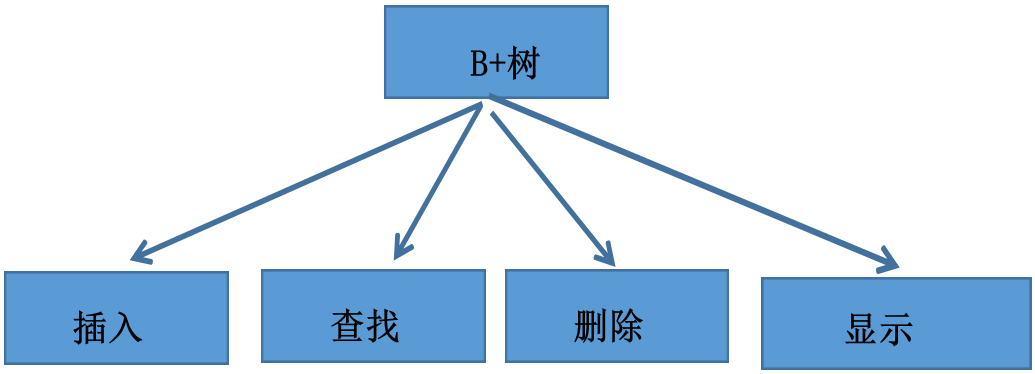
- **显示函数**

  实现功能：用图形的方式直观地显示所建立的整棵 B+树

  索引结构函数声明：

  void showTree(FILE *index, Node&node)

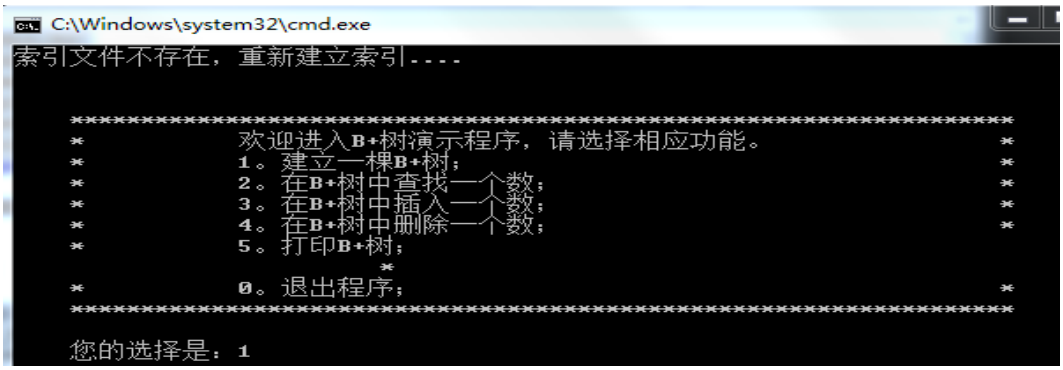
6.3 程序主要模块



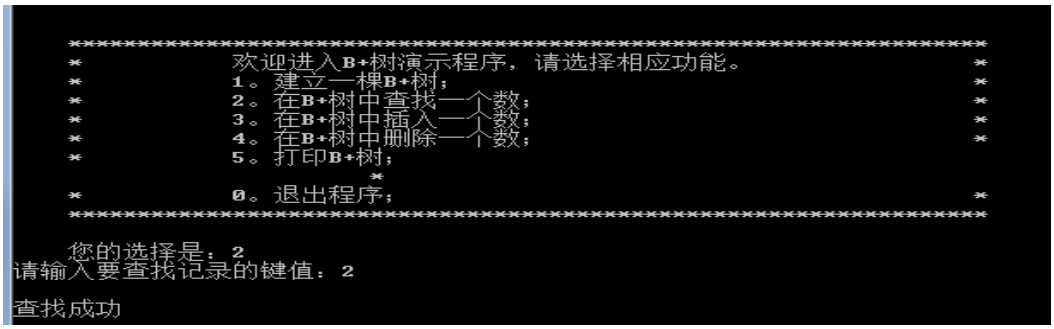
7 实验结果

7.1 实验部分操作代码结果展示如下：

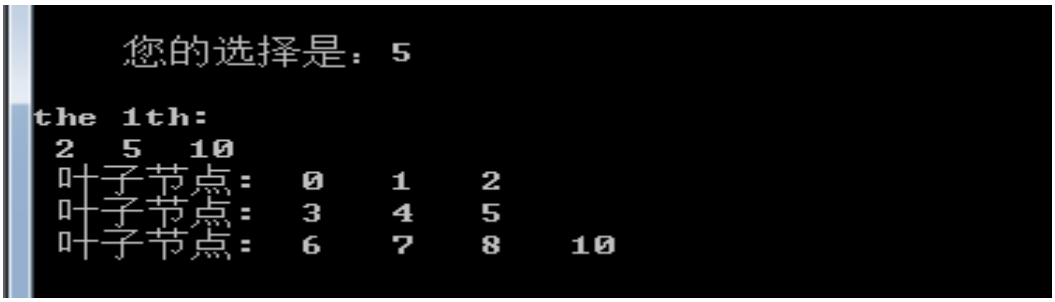
- 文件索引初始化



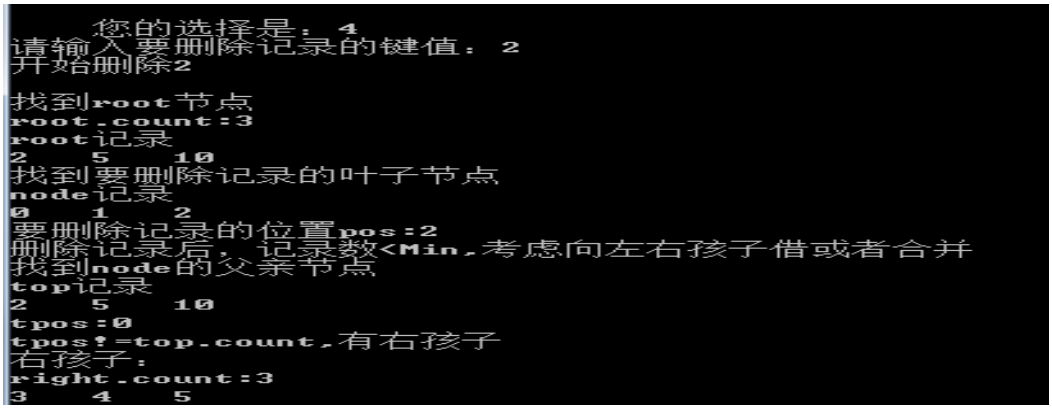
- 根据记录的键值查找：

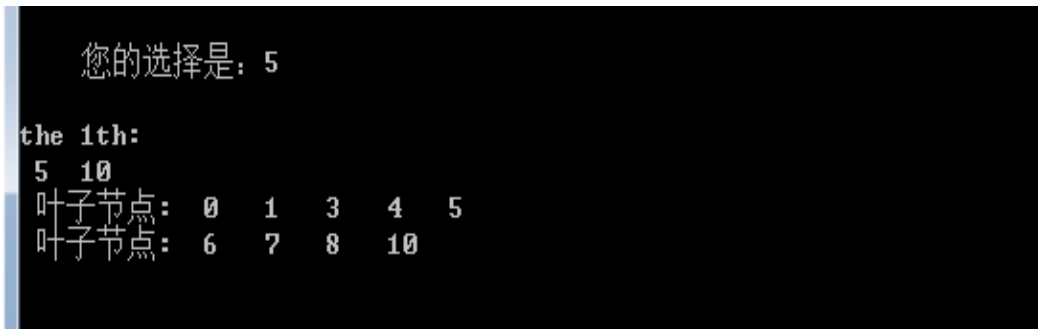


- 根据键值插入记录：



- 根据键值删除某个记录：





8 实验体会

虽然实验二相对实验一来说要容易一些，但是在实现的过程中也遇到了一些问题，其中最主要的还是怎样将实验二和实验一结合起来，或者说是怎么将一个项目的二个模块结合起来，所以我们团队在后期主要是针对这块进行了一些具体的讨论与实现。在实现了这个实验之后，我们团队对 B+树这种数据结构有了更加清楚的体会与理解。因此也懂得了只有通过实践才能够更加充分的理解理论知识。最重要的是，我们学习到了程序不同模块之间的连接也是十分重要的。我们也会不断的在今后的实验中锻炼加强这种能力。最后，最后我们这次实验也基本上完成了所要求的功能，但仍然有一些不足的地方完善和修正。