Project_4 实验报告

王若琪 18340166

一、实验目的

完成一个B+树仿真器程序的系统设计、实现和报告 。

二、实验环境

- 开发工具：Dev-C++
- 编程语言：C++

三、实验原理

3.1 B+树的概念

A **B+ tree** is an N-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children.

(以上信息来自维基百科 [https://en.wikipedia.org/wiki/B%2B_tree])

3.2 B+树的逻辑结构

B+树分为两大部分,一部分是非叶子结点,非叶子结点存储着大量的关键码,这些相当于索引表,并且存储在外存中;另外一部分是叶子结点,叶子结点中存储的是关键字的数据层(有可能是主键,也有可能是数据库中的记录),这些结点通过链表链接,并且有序存储在内存中。简单地说,B+树是平衡多叉树和有序数组链表的组合。它的一些特性如下：

- 数据项储存在树叶上。
- 非叶节点存储最多M-1个关键字以指示搜索方向；关键字i代表子树i+1中的最小关键字。
- 树的根的儿子数在2和M之间。
- 除根以外的非叶节点的儿子数在[M/2]和M之间。
- 所有的树叶都在相同的深度上，并且每片树叶拥有的数据项的个数在[L/2]和L之间。

四、分析与设计

4.1 题目要求分析（L和M的计算）

实验要求用一个大小为 40Bytes 的内存单元模拟一个外部存储块，规定关键字大小为 4Bytes，地址大小为 4Bytes，记录信息数据大小为 8Bytes。确定上述B+树的M值（用于内部M-路搜索树）和L值（用于每个叶子块存储的记录数目）。

- 根据课本《数据结构与算法分析》(Mark Allen Weiss)中文版137页的计算方法：一个区块最多容纳40字节，有M-1个关键字，关键字大小4字节，M个地址，地址大小为4字节。这样，一个非叶节点总的空间需求为8M-4字节，使8M-4<=40的最大M值为5，故求得M为5。并且L=40/8=5。(对于这种算法我个人存在疑惑，因为B+树的节点中还需要存有父母指针，可是课本上完全没有考虑这

一点，开始我以为书上的算法是一种特别的B树，但后来发现137页下方注释里写着这就是B+树，真是令人不解。）

4.2 需求分析

- 题目要求设计存储上述B+树的数据结构设计 （程序设计语言描述） ，在这里我们设计的B+树不仅能满足题目要求，还可以自定义M及L值。
- 设计B+树用于记录查找、插入和删除的算法 。
- 包含一个自行设计的20ms延时器模拟一次外部存取的时间延迟 。
- 能够做到将树直观的输出，便于检查和分析。

4.4 结构设计

在节点结构设计我们借鉴了一位哥根廷大学的学生的思路，参考网址见报告末尾。节点的结构为如下所示，其中信息数据 `value[]` 和孩子指针 `child[]` 都是以数组形式来储存的，并且以最大整数 `INT_MAX` 来填满空位，表示这个节点已满。这种储存方法参考了一个哥根廷大学的大佬的思路(见文末参考网址)，用 `INT_MAX` 可以在之后的查找、插入、删除时思路和代码都变得非常简单。既然是要求模拟题目要求的储存块，那么当程序刚开始要求输入M、L值时，用户可自定义为5,5，这样虽然实际Node的大小并不是40Bytes，但根据如下的定义抽象一下，完全可以做到**模拟**一个40Bytes的储存单元。

```
struct Node{
    int count; //节点中的数据个数
    Node *parent; //指向父母的指针
    int value[MAX]; //开一个存值的数组
    Node *child[MAX]; // 指向孩子的指针数组
    Node(int num, int val, Node* child0, Node* child1){
        count=num;
        parent=NULL;
        value[0]=val;
        child[0]=child0;
        child[1]=child1;
        for(int i=1; i<MAX; i++){
            value[i] = INT_MAX;
            if(i>1) child[i] = NULL;
        }
    }
    Node(){
        count = 0;
        parent = NULL;
        for(int i=0; i<MAX; i++){
            value[i] = INT_MAX;
            child[i] = NULL;
        }
    }
};
```

4.5 算法设计

1. **查找算法：**
    - 从根节点开始查找 `search(root, val)` ，将目标值和当前节点中的值依次比较，如果找到，则返回 `true` 。
    - 如果没有找到，并且当前节点有孩子节点，就用递归的方法， `search(current->child[i], val)` 在指示的孩子节点中继续寻找。

- 因为用极大值填充空余位置，所以算法用来确定要寻找的孩子节点时需要考虑的情况大大减少，给编写代码带来极大的方便。

2. **插入算法：**
   - 整体采用递归方法，先找到合适的插入位置，不管是否符合要求，先将这个数据插入这个位置，在插入过程中，要注意数组的有序性，这里用到了 `swap` 函数来交换数值，确保有序。并且由于B+树的索引的特性，既需要考虑在叶节点上的插入情况，又需要判断是否需要在非叶节点上插入此值用作索引。
   - 接下来判断插入后的节点是否合法，如果合法，则删除完成，如果不合法，有两种情形：
     - 如果插入后叶节点中的数据数目大于等于L，即 `current->count==L+1` ，此时需要将叶节点分裂，调用 `splitLeaf(current)` 。
     - 如果插入后非叶节点中的数据数目大于等于M，即 `current->count==M` ，此时需要将非叶节点分裂，调用 `splitNonLeaf(current)` 。在分裂时需要考虑根与非根的情况。
3. **删除算法：**
   - 整体采用递归方法，从上向下寻找，如果找到这个值，不管是否符合要求，先将这个值删除，因为是用数组方式储存每个节点中的值，所以在删除过程中，需要用到很方便的 `memcpy()` 函数，能使数组中的值整体移动位置。并且需要分非叶节点和叶节点两种情况来讨论。
   - 接下来判断删除后的节点是否合法，如果合法，则删除完成，如果不合法，有两种情形：
     - 如果删除后非叶节点中的数据数目小于[M/2]-1，则需要向兄弟借数值或者与兄弟合并。当他有相邻兄弟的值的数量大于[M/2]-1时，就像兄弟借一个值，并领养一个孩子。调用 `borrowNonLeaf()` 函数进行借操作。当他的相邻兄弟的值的数量都小于等于[M/2]-1时，意味着他不能向兄弟借了，这时应该与他的相邻兄弟进行合并，调用 `mergeNode()` 函数。进行这两个操作时，要注意的是重新分配被移动过的节点的父母指针也要更新。
     - 如果删除后叶节点中的数据数目小于[L/2]，则需要向它的兄弟叶子借或者与兄弟叶子合并。当他有相邻兄弟的值的数量大于[L/2]时，就向兄弟叶子借一个值，调用 `borrowLeaf()` 函数进行借操作。当他的相邻兄弟的值的数量都小于等于[L/2]时，意味着它不能向兄弟借了，这是应该与他相邻的兄弟合并，调用 `mergeNode()` 函数。进行这两个操作时，也要注意重新分配被移动过的节点的父母指针也要更新。

4.6 代码主模块命名清单

源代码中除 `main()` 函数之外的各个函数命名清单如下：

- 此函数可实现半可视化打印树的功能：

```
1 void print();
```

- 此函数可实现查找功能：

```
1 bool search(Node* current, int val);
```

- 此函数可实现对叶节点的分裂：

```
1 void splitLeaf(Node *current);
```

- 此函数可实现对非叶节点的分裂：

```
1 void splitNonLeaf(Node *current);
```

- 此函数可实现插入操作：

```
void insert(Node *current, int val);
```

- 此函数可实现对叶子节点的借操作：

```
void borrowLeaf(Node *left, Node *right, int leftNumber, bool isRight);
```

- 此函数可实现对非叶节点的借操作：

```
void borrowNonLeaf(Node *left, Node *right, int leftNumber, bool
isRight);
```

- 此函数可实现对叶节点的合并：

```
void mergeLeaf(Node *left, Node *right,int rightNumber);
```

- 此函数可实现对非叶节点的合并：

```
void mergeNonLeaf(Node *left, Node *right,int rightNumber);
```

- 此函数可实现删除操作：

```
void deleteValue(Node* current, int val, int curNodePosition);
```

- 此函数可实现对树清空操作：

```
void clear(Node* root);
```

五、程序运行操作

- 先根据提示自定义输入M和L的值（按题目要求可输入5,5）。
- 开始读取初始化树文件*input1.txt*，进行树的初始化操作。
- 树初始化完成之后，按照提示可选择自定义的删除、插入、查找、打印、输出树等操作。

六、测试样例和运行结果分析

M=5，L=5时，插入154后叶子分裂：

```
-----After inserting 76:-----

[ 56 ]
[ 4 29 43 45 ]  [ 56 61 76 90 890 ]

Time for inserting 76: 0.042 s

-----After inserting 154:-----

[ 56 90 ]
[ 4 29 43 45 ]  [ 56 61 76 ]  [ 90 154 890 ]

Time for inserting 154: 0.103 s
```

M=5，L=5时，插入45后根分裂并产生新根：

```
-----After inserting 61:-----

[ 4 29 56 61 890 ]

Time for inserting 61: 0.021 s

-----After inserting 45:-----

[ 56 ]
[ 4 29 45 ]  [ 56 61 890 ]

Time for inserting 45: 0.083 s
```

M=5，L=5时，插入21后有叶节点的分裂和非叶节点的分裂：

```
-----After inserting 524:-----

[ 56 90 394 591 ]
[ 4 25 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 186 367 ]  [ 394 461 524 ]  [ 591 890 981 ]

Time for inserting 524: 0.103 s

-----After inserting 21:-----

[ 90 ]
[ 29 56 ]  [ 394 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 186 367 ]  [ 394 461 524 ]  [ 591 890 981 ]

Time for inserting 21: 0.166 s
```

删除不存在的数时：

```
Enter your choice:2
Now erase! Enter the value:0
0 is not in this tree.
```

M=5，L=5时，删除时叶节点和非叶节点的合并：

```
Enter your choice:2
Now erase! Enter the value:4
-----After erasing 4:-----

[ 76 394 ]
[ 43 ]  [ 154 ]  [ 591 ]
[ 21 25 ]  [ 43 67 ]  [ 76 78 90 144 ]  [ 154 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for erasing 4: 0.147 s


Enter your choice:2
Now erase! Enter the value:21
-----After erasing 21:-----

[ 394 ]
[ 76 154 ]  [ 591 ]
[ 25 43 67 ]  [ 76 78 90 144 ]  [ 154 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for erasing 21: 0.207 s
```

M=5，L=5时，删除时产生层数改变：

```
Enter your choice:2
Now erase! Enter the value:154
-----After erasing 154:-----

[ 186 ]
[ 56 90 ]  [ 394 ]
[ 4 21 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 ]  [ 186 332 367 ]  [ 394 461 524 90 144 ]

Time for erasing 154: 0.166 s


Enter your choice:2
Now erase! Enter the value:367
-----After erasing 367:-----

[ 56 90 186 461 ]
[ 4 21 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 ]  [ 186 332 394 ]  [ 461 524 90 144 ]

Time for erasing 367: 0.186 s
```

M=5，L=5时，删除时叶子节点合并：

```
[ 90 ]
[ 29 56 ]  [ 186 394 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 ]  [ 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for printing: 0.248 s


Enter your choice:2
Now erase! Enter the value:25
-----After erasing 25:-----

[ 186 ]
[ 56 90 ]  [ 394 591 ]
[ 4 21 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 ]  [ 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for erasing 25: 0.186 s
```

还可以有其他的ML值，M=3，L=4时，一系列插入结果如图：

```
-----After inserting 43:-----

[ 56 ]
[ 4 29 43 45 ]  [ 56 61 90 890 ]

Time for inserting 43: 0.042 s

-----After inserting 76:-----

[ 56 76 ]
[ 4 29 43 45 ]  [ 56 61 ]  [ 76 90 890 ]

Time for inserting 76: 0.102 s

-----After inserting 154:-----

[ 56 76 ]
[ 4 29 43 45 ]  [ 56 61 ]  [ 76 90 154 890 ]

Time for inserting 154: 0.042 s

-----After inserting 367:-----

[ 76 ]
[ 56 ]  [ 154 ]
[ 4 29 43 45 ]  [ 56 61 ]  [ 76 90 ]  [ 154 367 890 ]

Time for inserting 367: 0.163 s
```

M=3，L=4时，最普通也是最简单的一种删除操作：

```
[ 76 394 ]
[ 29 56 ]  [ 154 ]  [ 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 56 61 67 ]  [ 76 78 90 144 ]  [ 154 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for printing: 0.261 s


Enter your choice:2
Now erase! Enter the value:56
-----After erasing 56:-----

[ 76 394 ]
[ 29 61 ]  [ 154 ]  [ 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 61 67 ]  [ 76 78 90 144 ]  [ 154 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for erasing 56: 0.124 s
```

清空树：

```
Enter your choice:5
The tree has been destroyed!
Now you can build your new tree!

Enter your choice:4

[ ]

Time for printing: 0.023 s
```

打印树：

```
Enter your choice:4

[ 90 ]
[ 29 56 ]  [ 186 394 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 ]  [ 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for printing: 0.246 s
```

查找值，发现查找时间与B+树的特点一致，推测时间复杂度在O(logN)，有的数据本身是索引值，所以很快就能找到：

```
Enter your choice:4

[ 90 ]
[ 29 56 ]  [ 186 394 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 ]  [ 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for printing: 0.244 s


Enter your choice:3
Now search! Enter the value:90
Find it!
Time for searching 90: 0.021 s


Enter your choice:3
Now search! Enter the value:29
Find it!
Time for searching 29: 0.042 s


Enter your choice:3
Now search! Enter the value:4
Find it!
Time for searching 4: 0.062 s
```

没有找到的情况，时间复杂度推测也为O(logN)：

```
[ 90 ]
[ 29 56 ]  [ 186 394 591 ]
[ 4 21 25 ]  [ 29 43 45 ]  [ 56 61 67 76 78 ]  [ 90 144 154 ]  [ 186 332 367 ]  [ 394 461 524 ]  [ 591 890 963 981 ]

Time for printing: 0.256 s


Enter your choice:3
Now search! Enter the value:1000
Can't find it!
Time for searching 1000: 0.063 s
```

七、小组分工

此项目由吴晓淳和王若琪合作完成，具体分工如下：

1. 先查阅资料，讨论并确定出所用的结构，以及总体的实现思路。
2. 由于B+树插入算法容易实现，根据一起写好的insert函数，王负责分裂非叶子节点的函数，吴负责分裂叶子节点的函数，实现后整合并一起debug。
3. 由于B+树的删除算法实属复杂，在借鉴网络上的思路后，一起讨论完成并优化。
4. 王负责打印B+树的函数，吴负责查找关键字的函数以及销毁B+树的函数；
5. 王和吴完成main函数后，王对main函数进行润色。
6. 最后吴加入了时间延迟功能。

八、体会与总结

1. 在设计B+树的时候，由于个人本身对较为复杂的递归操作的代码实现不太熟悉，于是很多递归的思想借鉴了网络，我从中收获了到了很多，比如说删除操作中使用的方法是我之前不太熟悉的，经过这次实验我熟悉了类似的操作。
2. 我学到了用数组存数据时的一种新的方法，在空位中存最大整形数据 `INT_MAX` 的方法，这样将会减少很多代码，比如判断插入位置时只需要作一种比较即可。
3. 以前不怎么使用 `memcpy()` 函数，这次由于对数组操作较多，于是使用 `memcpy()` 能够很快并且很方便地解决各种操作问题，同时也能大大减少代码量，增加代码易读性。
4. 在网络上的资料和代码良莠不齐，在编写代码的时候，我们借鉴了很多别人的思路（参考网址会附在实验报告最后），但是于此同时，我们也发现了别人各种方法存在的各种各样的bug，在研究他人思路的过程中我们又想到了许多自己新的方法。这让我明白了，编程不是一个闭门造车的过程，而是要通过多读、多看、多与组员、同学甚至互联网上的陌生人交流，才能有效的解决问题，将思路多元化。
5. B+树的设计是一个十分考验耐心和细致程度的过程，我认为它比之前任何一个大作业都要困难许多，因为我之前完成大作业很少有需要调试的时候，这次因为多处使用递归，所以在debug时必须使用调试功能，所以我又学会了许多有效的调试方法及思路。

6. 合作完成项目确实比一个人完成要顺畅的多，因为遇到困难时能够互相分享讨论，不至于被一个bug搞自闭。

九、缺陷和问题

1. 由于能力有限，打印树时只能做到分层打印，由于是B+树，经过思考可以判断出每个节点父母、孩子，所以能够简单表示出来大意。但是看上去比较丑，数据较多时会出现分行打印一层的情况，不便于查看。
2. 由于时间有限，没能够检查很多数值插入或删除的情况，只检查了有代表性的情况，可能还会存在一些我没有发现的bug，如果遇到bug请麻烦您联系我们。
3. 由于使用了数组，所以在L和M的设置上会有限制，而且在M和L较大时时间会有缺陷，但对于题目要求的较小M和L值还是比较方便的。

十、参考资料和网页

1. 《数据结构与算法分析——C++语言描述（第四版）》 Mark Allen Weiss
2. http://www.sayef.tech/posts/2018/07/16/b-plus-tree-tutorial
3. https://en.wikipedia.org/wiki/B%2B_tree
4. https://blog.csdn.net/qq_26222859/article/details/80631121