**B+树实验报告**

**一：组织形式**（5阶为例）

节点key存储叶子节点中的最小值。

日历

描述已自动生成

**二：基本操作**

**Ⅰ:插入**

**叶子节点插入**：

1. 叶子直接插入。

叶子插入分裂为两个节点：

1. 插入点在分裂点左边：left = 新节点right= 旧节点

图片包含 文本

描述已自动生成

1. 插入点在分裂点右边：left = 旧节点 right = 新节点

分裂产生的关键字插入父节点的key里面（非叶子节点插入）

**非叶子节点插入**：

1. 非叶子节点足够空间可直接插入

非叶子节点分裂为两个节点：

1. 插入点在分裂点左边：
2. 插入点等于分裂点：
3. 插入点在分裂点右边：

leaf溢出之后，把split\_key向上插入，父节点分裂，把lch和rch挂在key左右，然后递归的向上插入父节点分裂的key。

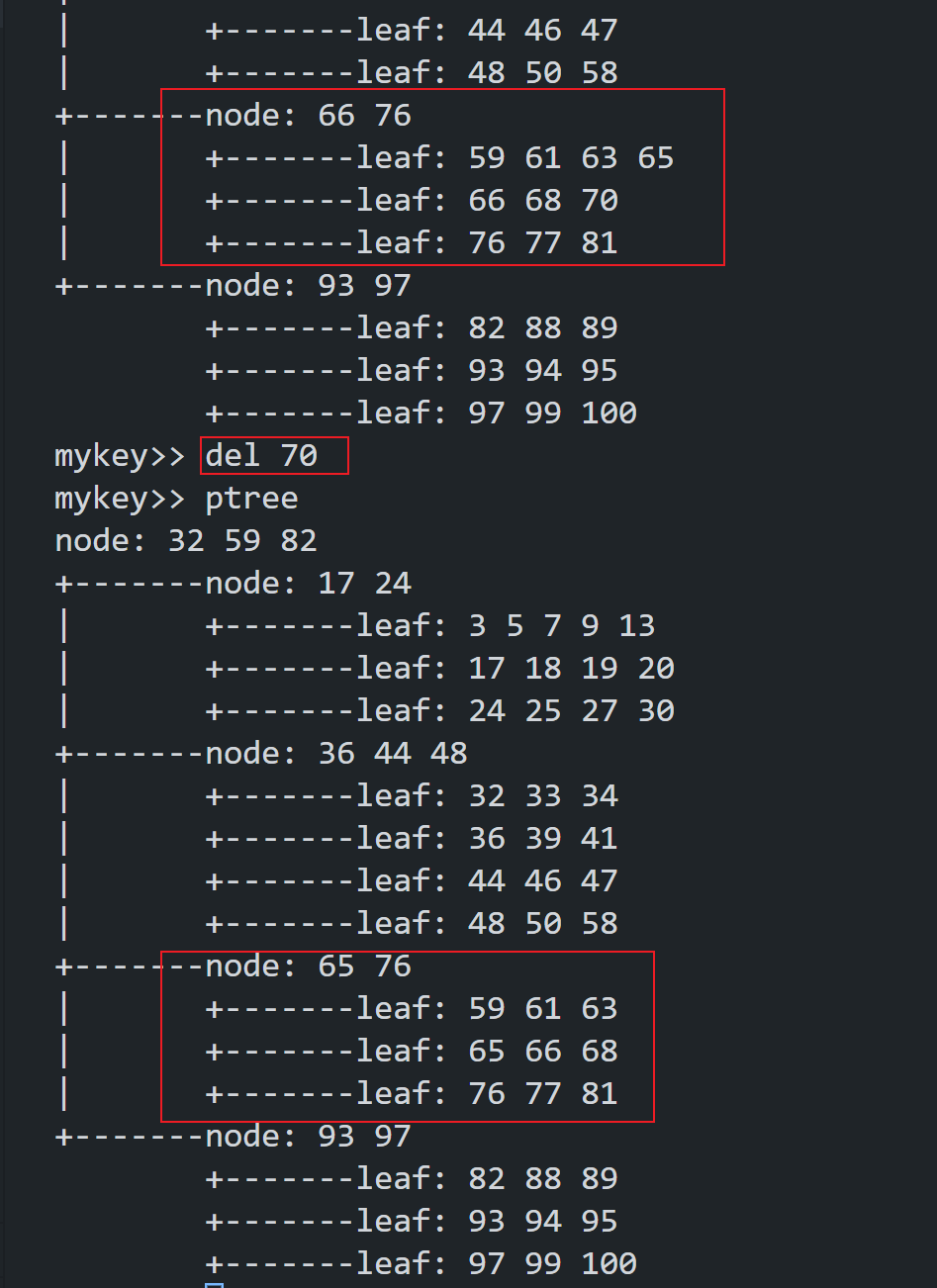
屏幕上有字

描述已自动生成

**Ⅱ：删除**

**叶子中key值删除**：

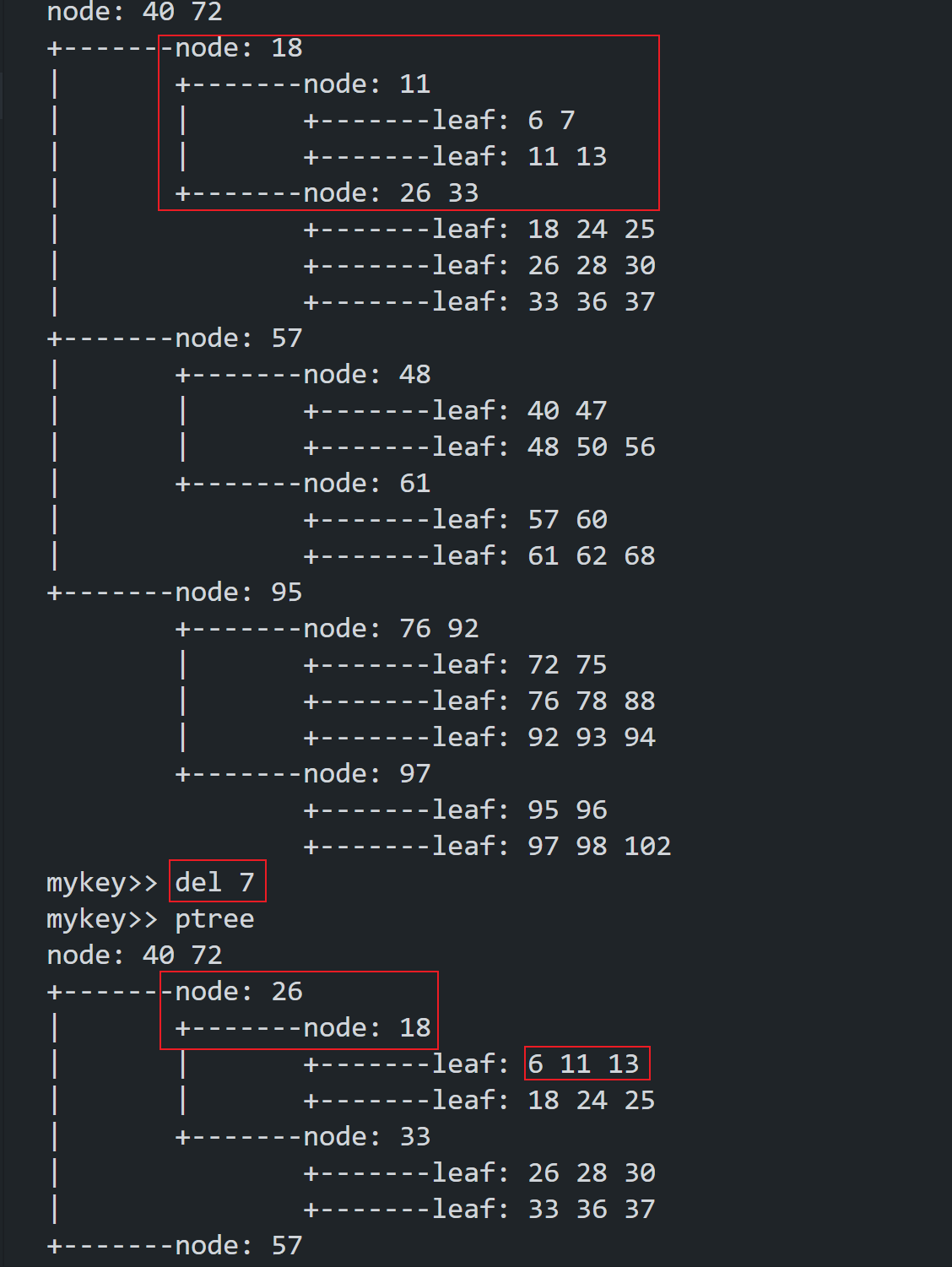
1. 简单删除
2. 左右兄弟够借，从左右节点借前驱或者后继，同时替换父节点中key值



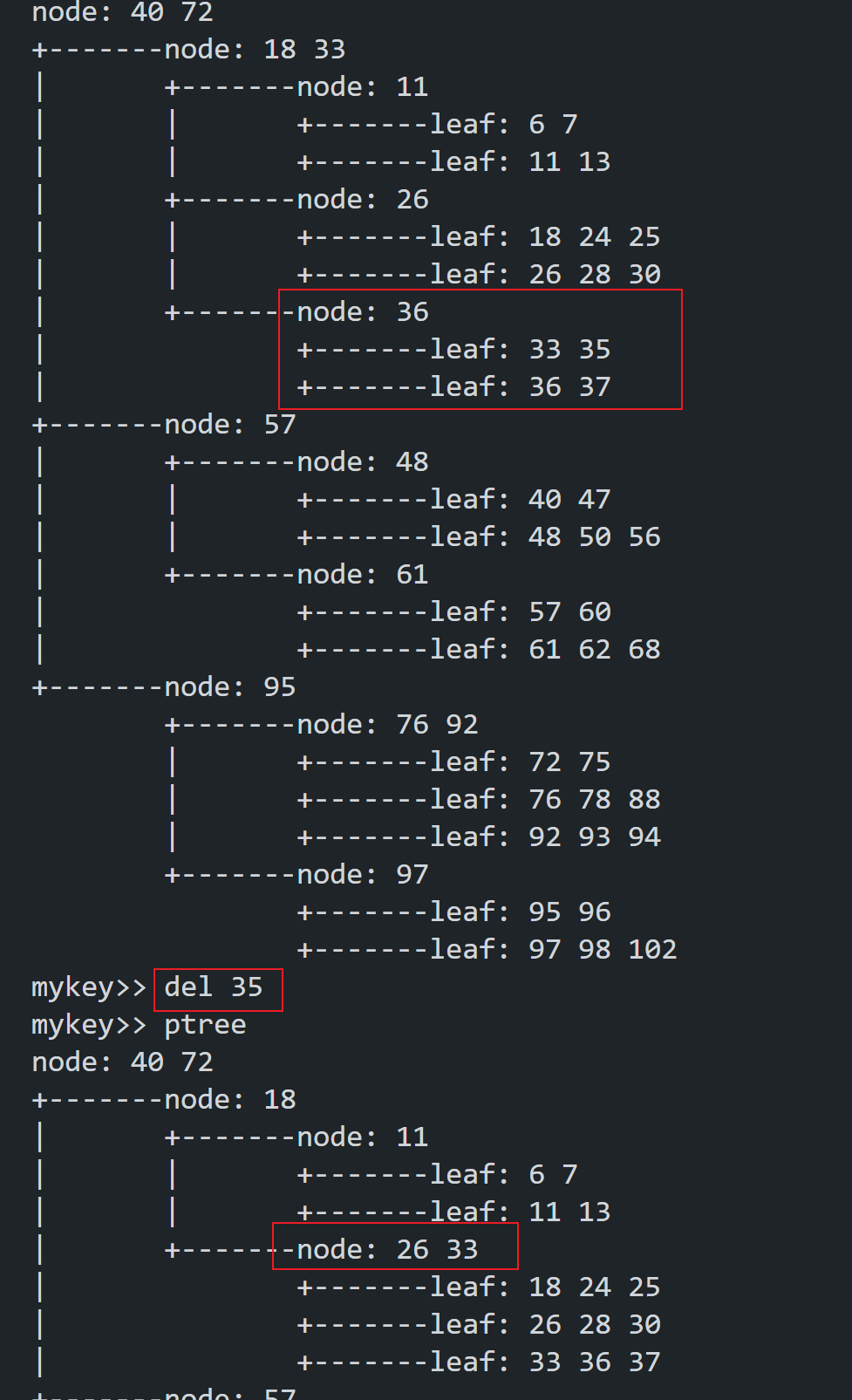
1. 左右兄弟不够借，合并兄弟节点

**非叶子节点中key删除**：（三阶为例）

1. 简单删除
2. 兄弟节点有富余，父节点key下移，兄弟节点key上移



1. 兄弟节点没富余，当前节点，父节点key和兄弟节点合并成一个节点，递归删除父节点key



**三：序列化，反序列化**

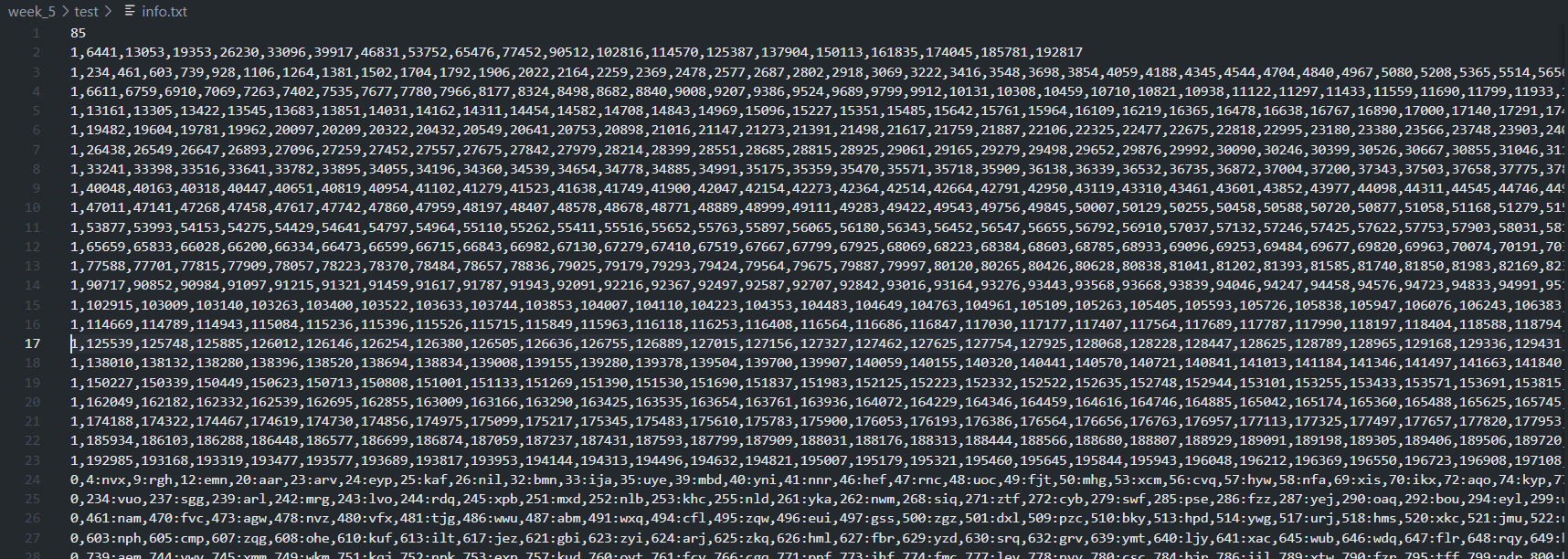
数据存储格式：

|  |  |  |  |
| --- | --- | --- | --- |
| 是否是叶子(0叶子，1分支) | key1 | key2 | key3 |

序列化：层次遍历存储每个节点

反序列化：根据文件每一行重新创建节点建树

使用info.txt存储数据，每一行为层序遍历的一个节点。第一行为树的阶数。



**四：大数据测试（时间单位s，操作取25次平均值）**

数据量1000：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 度 | 建树耗时 | 反序列化耗时 | 插入耗时 | 删除耗时 | 查询耗时 |
| 5 | 0.000714 | 0.000475 | 0.0000006 | 0.000001 | 0.00001216 |
| 15 | 0.000465 | 0.000378 | 0.00000064 | 0.00000044 | 0.00001628 |
| 25 | 0.000417 | 0.000279 | 0.00000064 | 0.00000064 | 0.00001072 |
| 35 | 0.000371 | 0.000278 | 0.00000048 | 0.00000056 | 0.00001316 |
| 45 | 0.000554 | 0.000271 | 0.00000048 | 0.00000040 | 0.00001072 |
| 55 | 0.000643 | 0.000365 | 0.00000076 | 0.00000052 | 0.00001604 |
| 65 | 0.000574 | 0.000440 | 0.00000056 | 0.00000060 | 0.00001792 |
| 75 | 0.000416 | 0.000304 | 0.00000040 | 0.00000056 | 0.00002544 |
| 85 | 0.000393 | 0.000366 | 0.00000064 | 0.00000044 | 0.00001772 |

数据量10000：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 度 | 建树耗时 | 反序列化耗时 | 插入耗时 | 删除耗时 | 查询耗时 |
| 5 | 0.014498 | 0.019580 | 0.00000292 | 0.00000212 | 0.00002612 |
| 15 | 0.011900 | 0.003636 | 0.00000176 | 0.00000144 | 0.00003156 |
| 25 | 0.009940 | 0.002938 | 0.00000124 | 0.00000112 | 0.00000684 |
| 35 | 0.008228 | 0.002202 | 0.00000112 | 0.00000116 | 0.00002484 |
| 45 | 0.006913 | 0.003359 | 0.00000144 | 0.00000116 | 0.00001452 |
| 55 | 0.007187 | 0.002426 | 0.00000104 | 0.00000084 | 0.00000584 |
| 65 | 0.006473 | 0.002307 | 0.00000100 | 0.00000112 | 0.00002544 |
| 75 | 0.007335 | 0.002347 | 0.00000084 | 0.00000112 | 0.00001392 |
| 85 | 0.007636 | 0.002832 | 0.00000136 | 0.00000092 | 0.00001568 |

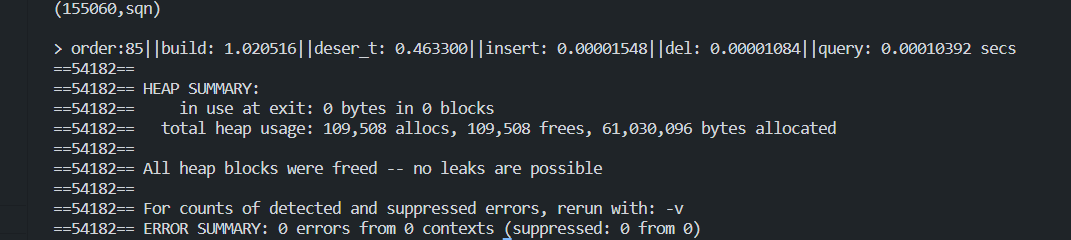
数据量100000：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 度 | 建树耗时 | 反序列化耗时 | 插入耗时 | 删除耗时 | 查询耗时 |
| 5 | 0.159609 | 0.027138 | 0.00000252 | 0.00000276 | 0.00001656 |
| 15 | 0.084342 | 0.015940 | 0.00000140 | 0.00000136 | 0.00000704 |
| 25 | 0.071520 | 0.020796 | 0.00000144 | 0.00000148 | 0.00001748 |
| 35 | 0.078471 | 0.017325 | 0.00000136 | 0.00000132 | 0.00000908 |
| 45 | 0.068301 | 0.017756 | 0.00000148 | 0.00000136 | 0.00001152 |
| 55 | 0.068027 | 0.016931 | 0.00000116 | 0.00000132 | 0.00000980 |
| 65 | 0.066678 | 0.018439 | 0.00000108 | 0.00000096 | 0.00002148 |
| 75 | 0.073867 | 0.017280 | 0.00000120 | 0.00000140 | 0.00001184 |
| 85 | 0.068550 | 0.017302 | 0.00000124 | 0.00000132 | 0.00002132 |

结论当大数据(>10000)时候，度为55附近效果较好。

**五：内存泄漏测试**

valgrind --tool=memcheck --leak-check=full --show-reachable=yes ./bplusTree



没有内存泄漏问题。

**六：学习体会**

实验使用B+树完成键值对的查询，删除。学习了B+树的结构和作用。锻炼了数据的定义和组织能力。

过程中遇到的问题，比如在使用大数据插入树结构时，树结构出错。把每一步插入后的树结构打印到log.txt，手推一次出错地方检查程序出错地方。

了解了程序测试工具，在使用valgrind检查时候发现使用链队列头节点忘记释放问题。