现实世界中的数据结构: Part II 报告

袁无为

1. 导言

在本次作业中，我们需要在提供的 B+ 树模板的基础上修改，使其支持查询区间内（不同的）key 的数量. 随后的各个章节分别介绍了我的实现方法、优化方法以及一些测试结果.

2. 实现方法

2.1. 简单的实现方法

对于 range query 操作，我们需要查询在 $[lvalue, rvalue)$ 中的 key 数量. 为了方便实现，可以将其拆分为两个前缀查询：用 $(-\infty, rvalue-1]$ 中 key 的数量减去 $(-\infty, lvalue-1]$ 中 key 的数量. 我的代码中的 count_prefix(v) 函数可用于查询 $(-\infty, v]$ 中的 key 数量.

在实现查询操作 $(-\infty, v]$ 时，需要在树上遍历，每次跳过当前结点的前若干个孩子，直到 $v < slotkey_i$（其中 $slotkey_i$ 为第 i 棵孩子子树中储存的 key 的最大值）. 在这当中跨过的子树中的所有 key 都在查询区间内，都需要被记进答案中. 因此可以考虑维护每棵子树中 key 的个数. 我的实现方法为，在 InnerNode 类中新增一个 size 变量，表示该结点对应的区间中储存了多少个 key. LeafNode 中储存的 key 个数已在 slotuse 中记录. 这样便可快速查询区间内 key 的数量，而不需要遍历所有的 key.

对于 insert 操作，下发的 btree.h 文件已经给出了 insert 的实现，我们只需要在框架中额外维护对 size 的修改. 当往树中某个结点所在的子树内成功插入了一个 key 后，需要将该子树的 size 加一. 当一个结点分裂后，也可以快速对其所有孩子的 size 求和，得到其分裂后的 size 值.

2.2. 更高效的实现方法

注意到在 2.1 中介绍的方法中，当我们进行询问操作以及分裂后维护 size 值的操作时，需要访问其所有孩子结点的 size 值. 这些值分散在不同的结点中，访问这些结点需要对内存进行大量的随机访问，违背了 B+ 树的设计理念. 因此我们可以额外地在每个内部结点中储存其所有孩子的子树大小，并在孩子的子树大小发生变化时一并修改这些信息.

3. 优化方法

3.1. 数据结构相关优化

3.1.1 Size 的存储方式

在插入的时候，由于每个结点中孩子子树的 size 是随着 childid 进行修改的，因此可将每个孩子的地址和其子树大小放在一起储存，这样在修改这两个值时只需要访问相邻的内存地址. 代码中 child_info 成员变量的 first 项为孩子的内存地址，second 项为孩子的子树大小. 这样做的代价是询问时访问的孩子子树大小在内存中不再是连续的，不过由于本次作业的数据中询问个数远比插入个数少，这样的存储方式仍然利大于弊.

3.1.2 Bulk Load

由于部分测试数据的开头会有大量的 insert 操作，我将这些插入操作替换为一次 bulk load 操作，以提升运行效率. 因此也需要在原有的 bulk load 函数中添加对 size 的维护. 特别地，我实现了基于基数排序的排序方法，其运行效率比快速排序更高.

3.1.3 Fanout

经过测试，将 B+ 树的分叉因子增大到 200 能够提升运行效率. 分叉因子的选择不仅与缓存大小相关，还与整体的时间复杂度相关. 因此其最优值有待进一步探究.

3.1.4 Load Factor

在原有的 bulk load 方法中，所有内部结点及叶子结点在插入之后都几乎是满的. 这样在随后的插入操作中容易上溢，需要进行分裂修复，降低了性能. 我选择了在 bulk load 时只将所有结点中维护的 key 个数填充到上限的 75%，留下一些位置给之后的插入操作填充.

3.2. 其他优化

3.2.1 输入输出优化

由于本次作业的输入输出量较大，我实现了基于 fread 和 fwrite 的输入输出优化，能够进一步提升程序性能.

3.2.2 编译选项

G++ 编译器提供了一些编译选项用于优化程序的性能. 除了众所周知的 -O3 优化外,我还使用了 -march=native -frename-registers -funroll-all-loops 这些选项进行优化.

4. 实验结果

在实现完 2.2 节中所介绍的方法之后，使用给出的服务器测得在下发的样例数据 1.in 上需运行约 9 秒. 在加入了第 3 章中介绍的优化方法后，各个样例数据所需的运行时间见下表. 其中在 1 号数据上的运行效率为优化前的 200%，说明优化效果显著.

样例数据编号	1	2	3	4	5	6	7
运行时间（秒）	4.553	3.248	4.346	1.762	4.548	1.101	4.708

表 1. 在样例数据上的运行时间.

在最终统一评测时，各个测试数据的运行时间如下表所示.

测试数据编号	1	2	3	4	5	6	7
运行时间（秒）	5.69	3.69	5.06	2.0	5.61	1.52	5.96

表 2. 最终评测时的运行时间.

5. 总结

总的来说，本文中介绍的实现方法和优化方法能够有效地完成给出的任务，不过仍有提升的空间. 本文作者在探究这个问题的时候对计算机系统结构相关内容的了解甚少，无法进行有效的优化，有待将来对该领域了解之后对本问题进行进一步的探究.