大作业：简易数据存储系统

5140379061

唐宇新

1. 概述

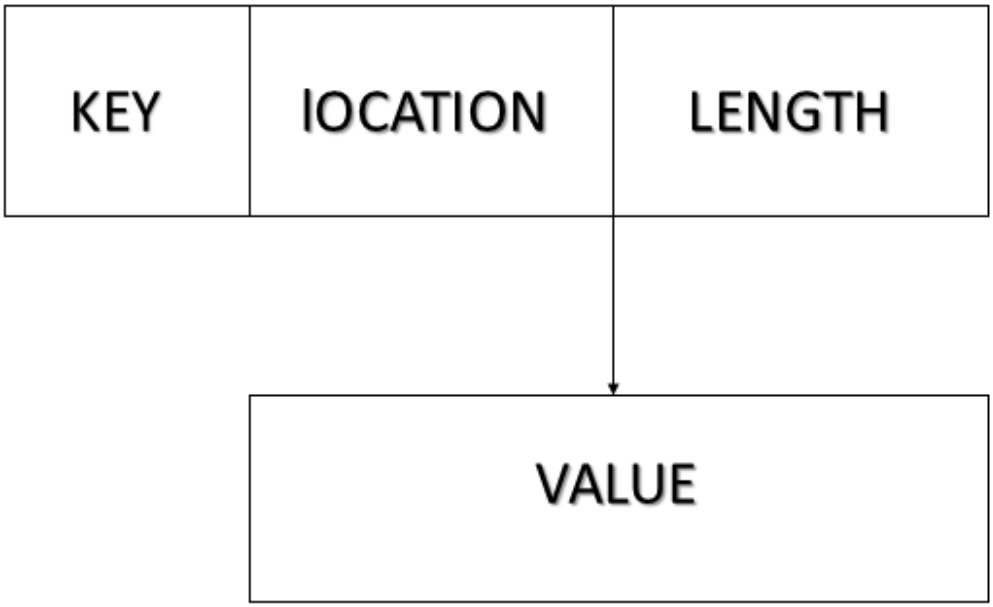
本次大作业我使用的是 B 树来实现数据库的存储和增删改查功能。

我将数据库的部分单独写成类，提供了方便的接口，使得操作磁盘就像操作内存。

最后对数据库的各类功能进行了测试。并且采用对比的方式，在性能的差别上进行了分析。

关键词：组织形式、B 树的实现、测试样例的设计、测试结果的分析、改良意见

2. 组织形式



索引是由 key，location 和 length 三个整数元素组成。

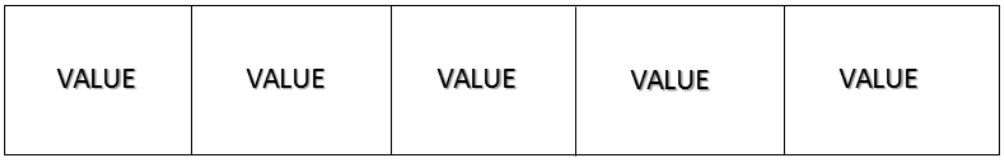
Location 代表着 key 所对应的 value 在数据文件中的起始位置，即 value 的头字符距离文件头部的位置。

Length 代表着 key 所对应的 value 在数据文件中的长度，即 value 所占的长度。

我是将所有的 value 一次性输入到一个 data 文件中，并且根据他索引所附带的 location 和 length 进行定位，然后继而进行操作。

最后将所有的索引组织成 B 树的形式。

而在文件中，数据则是用一种紧密的排列方式组织在一起。



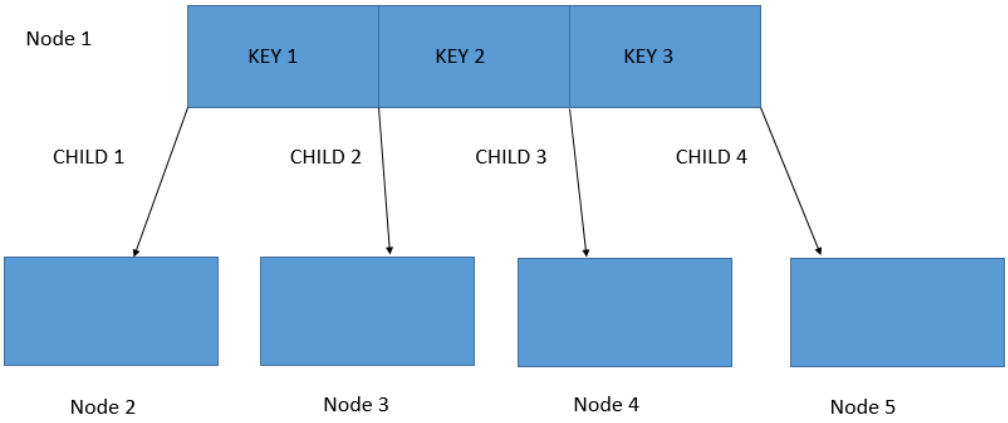
3. B 树实现

我使用的是自定义叉数的 B 树，可以测试不同叉数 B 树的性能情况。

然后结点中所存放的是 [M-1, 2M-1]个 key 值和[M,2M]个指针。

还有一个判断是否为叶子的 bool 值。

至于 B 树中各类操作的原理，这里不再赘述。



4. 同步性实现

在对硬盘上的数据文件进行修改的时候，尽量采用简洁的方式来实现。

在完成了对 B 树索引的更新之后，立即对硬盘数据进行更新。

插入操作是直接在文件的末端进行写入数据，删除操作是直接将数据替换为空格，查找则是将文件流中的指针直接移到数据所在位置，修改则是通过先删除后插入的方式，对数据进行覆盖和替换。

5. 数据库维护

在同步性模式下，数据库文件可能在大面积删除操作后，会产生较多的空闲空间，所以我使用了一个类似 vector 自动扩展的功能。

也就是说，当空闲空间的大小占总空间的大小超过一定的比例之后，会将原文件中的内容进行整理。

具体来说，B 树维持不变，新开一个 data 文件，在 data 文件中将非空闲数据写入，同时更新 B 树索引所对应的 length 和 location 数据。最后再将原 data 文件删除。

6. 测试设计

一共有 5 个测试。

正确性测试，性能测试(写入测试，查询测试，修改查询测试，删除测试，随机性测试)，打开关闭测试，数据库整理测试以及与 Map 比较测试。

每个测试测试最少 10000 组数据，最大 1000000 数据，并设置了错误抛出机制。

正确性测试就是与 Map 中的存储值进行比较，如果不同，则输出 ERROR,

都相同则输出 NO ERROR。

性能写入测试，对下面四个操作的测试(随机性)轮流进行。

```
bool InsertData(key,value)
bool deleteData(key)
bool setData(key,value)
bool getData(key)
```

打开关闭测试，读取索引文件和硬盘数据文件，重新组织成数据库。

数据库整理测试，在空余空间过度后，进行数据库的整理。

(我采用的是将所有数据进行一一删除，然后观察数据库整理过程)

Map 比较测试，对比 Map 的查询，插入，删除，修改四大操作，总结数据库的优劣之处。

最后设计了一个用户交互界面。

用来小规模测试。

7. 测试结果

(1) 不同的数据规模 在 Debug 模式和 Release 模式下

size	Debug(ms)	Release(ms)
100	5	1
1000	29	3
10000	320	16
100000	3055	159
1000000	31262	1513
10000000	384211	15182

(2) 在不同数据规模的情况下　相同操作使用的时间差异

size	getData(ms)	Delete(ms)
100	12	21
1000	25	86
10000	62	176
100000	106	587
1000000	1076	6861
10000000	12184	19356

(3) M 值改变对相同数据规模操作的影响

M size	pushData(ms)
8	1557
18	1529
58	1547
108	1590
508	1928
1008	2276
5008	4642
10008	7962

(4) 存储数据大小的不同

Data size	pushData(ms)	deleteData(ms)
0-100	1229	251
100-500	2142	829
500-1000	5995	1583
5000-10000	108458	19632

(5) STL Map 性能比较

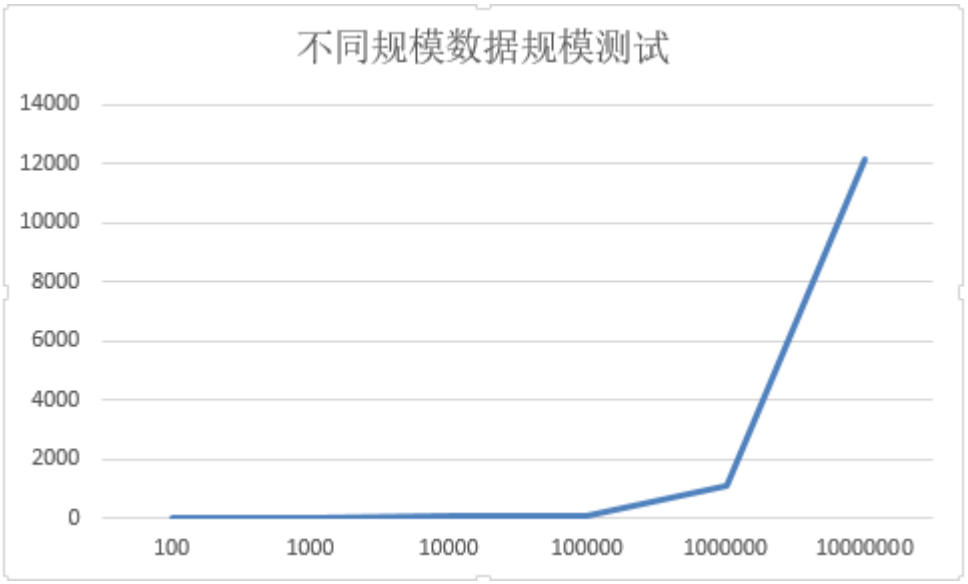
Data size	Map	DataBase
1000	3	2
10000	5	7
100000	18	28
1000000	268	311
10000000	3978	3324
100000000	47978	32747

8. 结果分析

(1)

横向分析：很显然，debug 和 release 模式下，速度的差异还是挺大的。当输入的数据到 10000000 的时候，速度整整差一个数量级。Release 状态下编译优化了很多不必要的细节。

纵向分析：

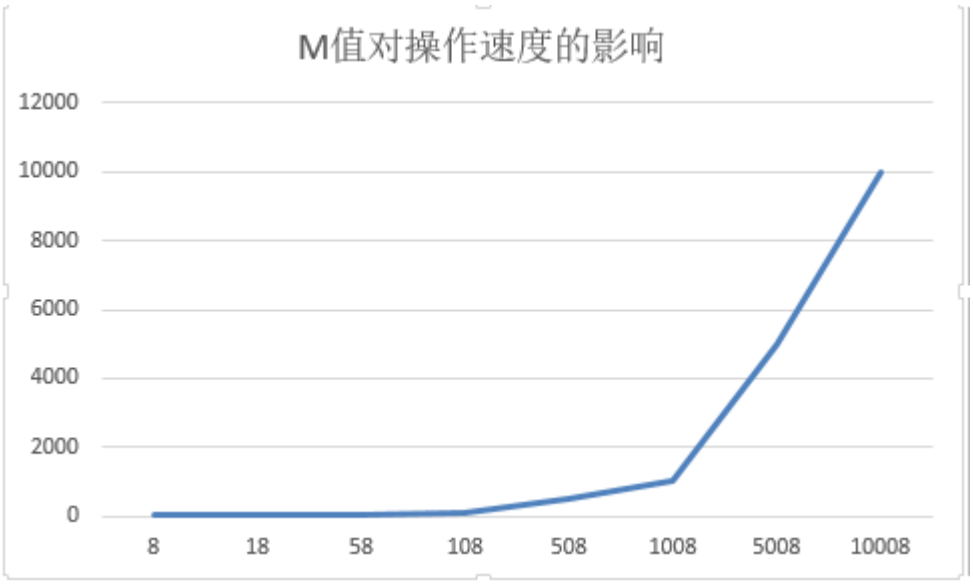


B 树在数据量很大之后，性能严重下降。我觉得从实现上来说，B 树的缺陷应该在于在单个节点中比较的操作，尤其是当数据量非常大了之后，如果数组实现的，可以换成二分查找应该会快很多。

(2) 在渐进的趋势下，getData 操作和 deleteData 操作速度慢的主要原因是 相对比其他数据结构，B 树对硬盘的操作频繁的多，数据量是依次多 10 倍，操作花时也从一开始的 4,5 倍增长到 10 多倍 增长。

(3) 存储数据大小不同比较

在硬盘每次读写的情况下，数据的大小会很大程度上影响程序的效率。

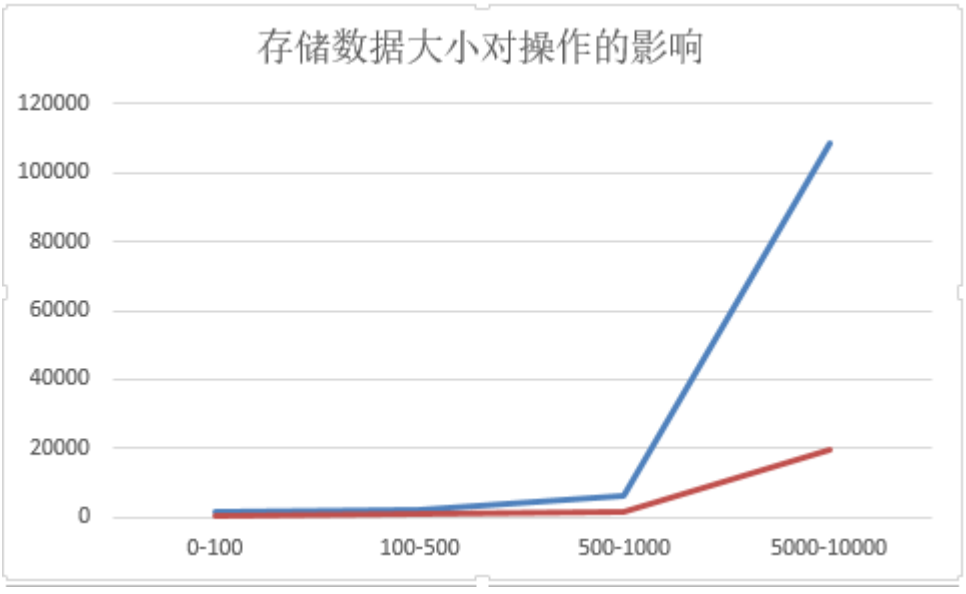


8,18,58,108 等 M 值速度差不多。

而 508，1008，5008，10008 的速度显然慢了很多很多。

推测原因还是节点内部比较查找的次数明显增加，如果用链表实现的，则复杂度无法降下来，如果用数组实现，则可以转换为二分查找。

(4)



蓝色的是插入操作，红色的是删除操作。

明显当数据量超过一定限额后，蓝色速度急剧下降。

主要原因还是和硬盘的读写有关系。一旦出现了大量的读写硬盘的操作，整个

程序的效率便会有极大的下降。

(4) 与 STLMAP 的对比当中，可以很轻易的看出,由红黑树实现的 STL Map 在初始阶段速度很快，但是一旦数据量很大之后就慢了很多。

而 B 树的优点就在于此。当数据量较小时，节点内顺序查找的效率明显很低，一开始会比红黑树慢一些，当大量数据输入时，红黑树的高度过高，导致效率变得很低很低，而 B 树能将高度维持在一个可接受的范围内。

9. 期望与改进

(1) 并没有用 HashMap 将此简易数据库实现一遍,这样更好可以与 B 树相比较，并且能更加深刻地理解，譬如 MySQL 之类的数据库为何不用 HashMap，而用 B 树 B+树实现。

(2) 二分查找可以更快地提高查找的速度。有必要及时将顺序查找换位二分查找。

(3) 有关索引，自己在本次大作业中，索引几乎在很多时间里面，全部组织在内存中，但是在真正的数据库中，索引文件的大小是很难放在内存里的，一般来说，只有 B 树的根节点会长期留在内存中，其他节点都是在硬盘上。这时候需要设计缓存。而对于此次大作业，我觉得关于缓存的设置，需要进一步的完善和修改。