简易数据库

设计文档

516030910375

蔡一凡

2017.07

目录

I．基本信息

·项目名称： 简易数据库

·使用数据结构： B+ 树

·开发环境： Visual Studio 2015 Community

·测试环境：

Windows 10 家庭版 14393.1198

处理器： Intel® Core™ i5-6300HQ CPU @ 2.30GHz

内存： 8GB

系统类型： 64 位操作系统

·功能概述：

实现了<key, value>型数据的查找、插入、删除、修改等功能。

II. 提供接口

本数据库所有数据均采用 c++ 标准库中的 string 类型存储。

·创建数据库：Database();

·检查目录的正确性：bool checkFile();

如果 data 目录不存在，则会报出错误信息,返回 false。

·新建数据库：bool refresh();

新建一个空的数据库，会覆盖原有的所有数据。

·插入数据：string insert(const string &key, const string &value);

key 为索引，value 为值。

如果插入成功，返回“Success.”，否则返回错误信息。

·查找数据：string search(const string &key);

key 为所要查找的数据的索引。

如果找到，返回“Found：”+找到的 value；如果没有，返回“Not Found.”。

·删除数据：string del(string &key);

key 为所要删除的数据的索引。

如果删除成功，返回“Success.”，否则返回错误信息。

·修改数据：string modify(const string &key, const string &value);

key 为所要修改的数据的索引，value 为修改后的值。

如果修改成功，返回“Success.”，否则返回错误信息。

·保存：`void save();`

将内存中的数据与文件中同步。当程序正常关闭时，会自动进行保存。

III. 功能实现

i. 模块实现：

本简易数据库由以下部分的代码构成：

·BPlustree.h & BPlustree.cpp：

包含了 B+树及其内部　节点的声明以及实现。

·functions.h & functions.cpp：

包含了一些需要用到的字符串处理等根据需要自己编写的函数。

测试部分代码：

·AutoTest.cpp：

自动进行数据量为 100 万的插入删除测试，并且输出所需要的时间。

·ManualTesting.cpp：

提供用户自定义的使用，有相关的使用提示和错误处理。

ii. 功能实现：

本简易数据库使用的数据结构是 B+树。

由于考虑到数据库需要存储的文件规模很大，而指针在文件中的存储并不方便，因此本简易数据库并没有直接采用指针，而是在每个 Node 中记录了其父节点和子节点在文件中的位置，来代替指针的作用。节点的数据在文件中和内存中都有保存。其中文件中分为三行，第一行为该节点所储存的所有 key，第二行为该节点的所有子节点在文件中的位置，第三行为该节点

的父节点在文件中的位置。每个节点在文件中的位置记为index，取代指针，起到了访问这个节点的作用，在内存中也是如此。

在查找、插入、修改、删除等操作中，如果需要访问一个节点的信息，程序会先在内存中根据 index 查找是否之前从文件中读取过该节点，如果有，那么直接调用内存中的数据；如果没有，那么将文件指针移到 index 的位置，读取数据后，将数据存到内存中。这样可以避免反复从文件中读取数据，也避免了每次修改了节点的数据后，还要修改文件，大大节省了时间。由于总数据量很大，因此只在内存中存储一部分节点。如果在不断读取节点的过程中，内存中的节点的数量超过设定的值（在#define 中），那么程序将会自动进行保存，即将内存中的数据写入文件中，并清空内存中的数据。程序在用户执行查找操作或正常退出程序时也会自动进行保存，用户也可以用手动进行保存（利用 save()接口）。

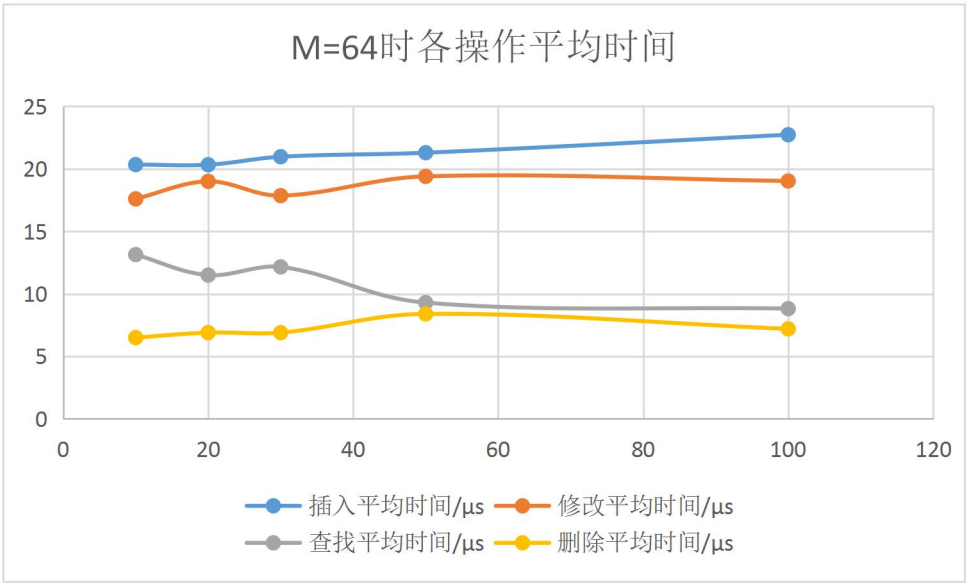
当进行插入、删除、修改等会修改各节点的数据的操作时，为了节省时间，所有的数据只会在内存中进行修改，而后一次性通过自动或是手动的保存操作，一次性同步到文件中。

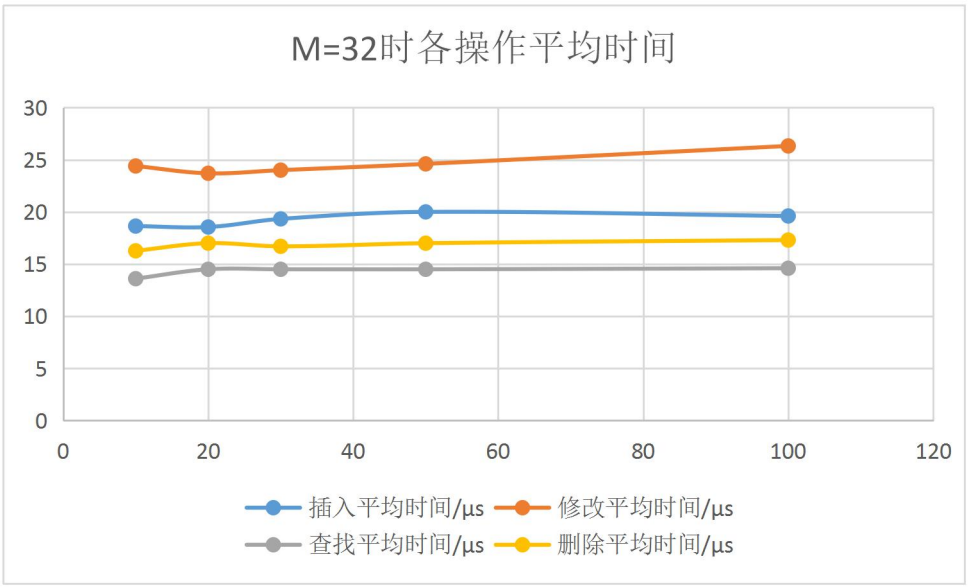
IV．特点及其分析

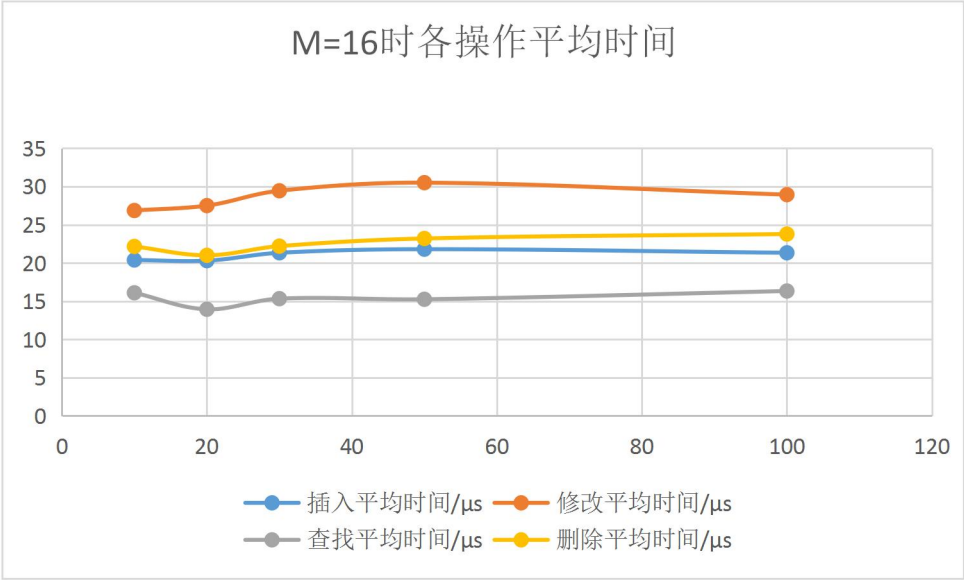
如前文所述，本简易数据库并没有直接采用指针类型，而是使用节点在文件中的位置，即 index，作为访问节点的工具。这样避免了所有数据堆叠在内存中的情况。并且这种方式的存储，使得每个节点直接被储存在了文件中，对应关系明确。读取数据时可以直接读取各个节点的信息，而不是在运行程序后要在内存中重新构建一棵 B+树。

V．测试及其分析

- 在 M=16，32，64 时分别进行了各操作，统计时间







根据图表中的数据，可以发现，当每个节点容纳的 key 的个数为 64 时，操作显然更快，原因是树的深度更少，只需要查找数量较少的节点。另外，当内存的节点数量大于一定值时，需要释放内存，更新文件。M 的值越小，节点数越多，需要更新的频率也越高。

另外，随着数据量的变多，发现平均时间并没有明显变长。原因可能是，程序刚运行时的数据需要从文件中获得，花费时间多；而后面数据从内存中读取，大大节省了时间。这种影响和深度的影响大致抵消了。

VI．待改进的地方

为了防止修改一段数据后，导致后面所有数据需要后移，造成极大的时间浪费，因此每个节点被设置为定长。这样的做法，在数据量较小时，会造成磁盘空间的部分浪费，而如果一段数据过长，将无法存下，因此本简易数据库对数据的长度做出了限制。另外，随着数据量的变多，节点的文件中的位置变大，位数的增多也会导致定长的空间不够存储，导致数据库无法存储 130 万以上条数据。

以上是本数据库做的主要的不足的地方，已经想到了好的方法，即将修改后的数据放在文件末尾，然后将原来的位置腾出，当有符合该长度的数据时填充该空间。由于时间不足，暂时没有改进。