算法 project1 设计文档

姓名：崔绍杰 学号：518021911244

1. 设计

整个项目分为跳表部分和SSTable部分

1. **跳表**

跳表基本实现与书中相同，不再赘述，只介绍其中几个特别处理点。

1. 如果在跳表内重复输入相同的key值，那么将会直接将旧结点的value值替换为新结点的value值，不会重复整个插入操作。
2. 当删除结点时，不仅会删除原先的结点，还会插入同key但value值为null的结点，为以后写入文件做准备。
3. 删除所有结点的操作被写到一个私有函数clearSkipList中，在调用reset函数和将内存中内容写入SSTable中被调用。
4. SSTable
5. 单个SSTable设计如下：

开头为结点数目，之后为每个结点的数据，依次为key值、value字长、value值，而后为索引区，每个结点的索引依次为key值、offset，最后为该文件内所有value值的总长度。

1. 从文件中获取结点的函数如下

uint64\_t getTable(SSTnode \*Node,uint64\_t level,uint64\_t table);

SSTnode中保存key.value.next，level为层数，table为文件次序，先从文件头获得结点数目，而后依次读取结点数据。

1. 向文件中写入结点的函数如下

**void** writeTable(SSTnode \*Node,uint64\_t level,uint64\_t table,uint64\_t Num);

参数含义同上，Num为要写入的结点个数。

1. **获得**文件中key值范围的函数如下，主要为compaction以及get函数服务

**void** getRange(uint64\_t level,uint64\_t table, uint64\_t &min,uint64\_t &max);

判断某key值是否位于某SSTable中函数如下，其中调用了getRange函数，目的与其相似

**bool** inRange(uint64\_t level,uint64\_t table, uint64\_t min,uint64\_t max);

在文件中搜寻key值函数如下，主要为put函数调用

**bool** search(uint64\_t level,uint64\_t table,uint64\_t key,std::string &Result);

1. 将内存中文件写入SSTable函数原型如下

**void** tranToSST();

主要是将链表中底层的Node结点转化为writeTable函数要求的SSTnode结点，写入level0，同时判断是否compaction

1. Compaction函数设计如下

**void** compaction(uint64\_t lev);

该函数较为复杂，将分阶段介绍。

1. 第一个阶段扫描要进行compaction的level，得到最小值min和最大值max，如果level为0，则扫描目标为table0,table1,table2，若level大于0，则扫描所有超过数量限制的文件。
2. 第二个阶段扫描level+1层，依次获得各个table的范围并与min和max比较，在min与max之间则加入要compaction的队伍。
3. 第三个阶段建立一个SSTnode的指针数组，数组大小为参与compaction的table数，每个table的数据形成一个链表，其表头由上述指针数组里的指针指向，所有被提取的table全部删除。
4. 第四个阶段为归并排序，将原指针数组中所有数据按key值大小汇成一条链表，依次循环比较各个当前链表头的key值，选出最小的加入新链表中，如果出现key值相同的情况，考虑到“越晚插入优先级越高”、“除level0外其他层中不会出现重复结点”和“level0层中table数越大说明插入越晚”三个基本原则，不难得到优先级最高的那个，将优先级低而key值相同的结点直接释放掉。
5. 第五个阶段新生成table，先将之前删除的高层table补上，若此时还有归并阶段生产的结点剩余，则写到高层的最后。
6. 第六个阶段，判断高层是否需要compaction
7. 设计

程序通过了给出的三个测试，测试环境为clion，写出CMakeLists文件后进行编译运行。