数据结构大作业报告

吴紫航 171860659

1. 实验目的

实现一个简单的数据库，有单点增删查改，区间增删查改，混合操作，集合操作四类功能

1. 实验设计
2. 过程思路

框架先把每个command对应的文件初始化对象，然后进行四种测试，每个测试开始的时候先读入数据，然后根据command循环读入，然后循环解析，解析用到helper函数，根据解析的内容进入相应的函数体里面实现相应的命令

1. 数据库类型设计

Command和record的结构，框架代码已经给出，这里我的数据库的结构的类是mydatabase，这个类中有一个head和tail指针，表示record记录的链表首尾，链表上每个结点表示数据库的一行记录，tablename表示表名，num表示记录个数即链表长度，（head不存记录，仅作为表头），colnum表示有多少个属性，不包括名字和id

Colname数组储存属性名字。

（3）具体功能函数的实现算法

Helper：

根据传入command的每个分块的名字，用条件判断即可，主要是关键字的识别，复杂度o（1）

INSERT：

在helper中找到参数id，name，每个value的值，然后存入record中，参数传递这个record即可，在插入过程中根据id的先后顺序，遍历链表，找到对应位置插入，因为原始数据id有序，所有直接在表尾插入，复杂度o（1）

DELETE id：

Id作为参数传入，从头开始向后找id，如果id<当前结点，则停止，否则一直寻找。如果没找到不做处理，如果找到了则删除，复杂度o（n）

DELETE id1，id2：

Id1和id2作为参数传入，从头开始向后找id1，如果id1<当前结点，则停止，否则一直寻找。如果没找到不做处理，如果找到了则从id1开始删除，如果当前小于id2则循环删除，最多遍历一次表，算法复杂度o(n)

SET/ADD id/id1，id2 key value：

传入id/id1，id2，key，value作为参数，寻找思路和delete中思路相同，找到则改值/加值，否则无操作，复杂度o（n）

QUERY id:

从链表头开始找，找到则输出到文件退出函数，复杂度O（n）

QUERY name = str

从表头开始找符合相同名字的，一直到表尾为止，找到的输出文件，复杂度O（n）

QUERY key ？ value:

?可以是> >= < <= = ！=

则传入key，符号symb，值value，按链表依次遍历满足条件的

QUERY id1 id2 key ? value:

和前者思路类似，但是要从第id1个表项开始遍历，直到id2个表项或者到表尾

查询前k大，第k大，前k小，第k小的数据的方法：

如果依然暴搜，复杂度达到O(KN),太不划算。

以求前k大为例解释我的算法：

先建立一个长度是k的最小堆，那么堆顶是k个数中最小的，再遍历剩下的n-k个数字，每次把当前数字和堆顶数字比大小，如果当前数字大，则替换堆顶，调整最小堆（堆的结构在common.cpp,common.h）里面增加了。这样遍历一遍就可以了，每次循环，都要把堆顶和当前的指针项比较大小，决定是否替换，这样堆始终是所有数据中的前k大的数据，因为始终把比较小的记录替换出去了，算法复杂度Nlogk，接近N，比较快捷！

第K大的数就是堆顶，即最大的K个数中的最小的那个，前K大的数只要把堆因此拿出堆顶，再逆向输出到文件即可；

同理找第K小的数，只要维护一个最大堆，始终把比较大的数字从堆顶替换出去即可

这样混合操作的复杂度都是Nlogk了

SUM id1 id2 key：

从第id1个结点开始，一直累加到id2个结点即可

复杂度O（n）

UNION table1 table2：

先在helper中识别哪个表在前，两个表都需要进行去重的操作，因此用name 字符数组记录table的名字，不断加入，但遍历到有重复的不加入，去重复杂度O（n^2）

然后直接输出第一个表的所有名字，然后对于第二个表，每个名字在第一个表中搜索，如果不在则输出到文件，输出复杂度O(N+N^2)

整个过程复杂度O(N^2)

INTER table1 table2：

先在helper中识别哪个表在前，两个表都需要进行去重的操作，因此用name 字符数组记录table的名字，不断加入，但遍历到有重复的不加入，去重复杂度O（n^2）

然后对于第一个表，每个名字在第二个表中搜索，如果在则输出到文件，输出复杂度O(N^2)

整个过程复杂度O(N^2)

文件输出函数，和record输入时一样对<<进行重载即可

1. 实验备注

实验环境是vs2017，默认的栈设置会不够，在设置中调大栈堆框架就能得到正确的结果，实验代码量1200行左右

用时，

第一阶段3.1s左右

第二阶段0.06s左右

第三阶段0.2s左右

第四阶段0.7s左右

包括load时间，一共6s左右