|  |
| --- |
| 哈尔滨工业大学(深圳) |
| **《数据库》实验报告** |
|  |
| **实验四**  **查询处理算法的模拟实现**  学 院: 计算机科学与技术   |  |  | | --- | --- | | 姓 名: | 郑漫莎 | | 学 号: | 180110711 | | 专 业: | 计算机科学与技术 | | 日 期: | 2021-05-05 | |

# 实验目的

*阐述本次实验的目的。*

理解索引的作用，掌握关系选择、连接、集合的交、并差等操作的实现算法，理解算法的I/O复杂性。掌握利用缓冲区的排序方法--两阶段多路归并排序算法。

# 实验环境

*阐述本次实验的环境。*

Windows7操作系统、VScode

# 实验内容

*阐述本次实验的具体内容。*

关系R具有A和B两个属性，其中A和B的属性值均为int型（4字节），A的值域为[20,60]，B的值域[1000,2000]。

关系S具有C和D两个属性，其中C和D的属性值均为int型（4字节），C的值域为[40,80]，D的值域[1000,3000]。

1. 实现基于线性搜索的关系选择算法：基于ExtMem程序库，使用C语言实现线性搜索算法，选出S.C为输入值的元组，记录IO读写次数，并将选择结果存放在磁盘上。
2. 实现两阶段多路归并排序算法(TPMMS)：利用内存缓冲区将关系R和S分别排序，并将排序后的结果存放在磁盘上。
3. 实现基于索引的关系选择算法：利用2中的排序结果为关系S建立索引文件，利用索引文件选出S.C为输入值的元组，并将选择结果存放在磁盘上。记录IO读写次数，与1的结果进行对比。
4. 实现基于排序的连接操作算法：利用2中的排序结果，对关系S和R计算S.C连接R.A，并统计连接次数，并将连接结果存放在磁盘上。
5. 实现基于排序的两趟扫描算法：利用2中的排序结果，实现并、交、差，并将结果存放在磁盘上，并统计并、交、差操作后的元组个数。

# 实验过程

*对实验中的5个题目分别进行分析，并对核心代码和算法流程进行讲解，用自然语言描述解决问题的方案 。并给出程序正确运行的结果截图。*

1. **实现基于线性搜索的关系选择算法**

问题分析：顺序读取表S所代表的磁盘块，对于每个块进行查找，并将第一个元素相等的结果写入缓存区。当从缓存区获取的块满时，写入磁盘，重新获取缓存区的写入块。最后将剩余结果写入磁盘。

算法流程：

1. 从输入端获取需要搜索的值equal\_num。
2. 从缓冲区中申请一个空块re\_blk， 用于写入结果
3. 从关系S的第一个磁盘块到最后一个磁盘块，在当前读入磁盘块中查找所有S.C=equal\_num的元组，将元组写入缓存区re\_blk，获得下一块的地址，释放读入的磁盘块。
   1. 当re\_blk满时，写入磁盘，重新申请缓冲区的块。
4. 将re\_blk中的内容写入磁盘中。
5. 输出结果。

核心代码：

1. 对于每个blk，有7个元组，通过getXY()获得该blk的第i个元组，编号从0-6。
2. 如果元组为需要的元组，则输出结果，计数加一。
3. 将结果用writeToBlk()写入缓存区re\_blk，blk\_count为缓存区中已有的元组数量，当缓存区块满，写入write\_blk的磁盘块中，重新申请缓存区块。
4. 获得当前磁盘块的下一磁盘块作为返回值，释放当前磁盘块。

for(int i = 0; i < 7; i++)

    {

        getXY(&X, &Y, blk, i);

        // equal

        if(X == equal\_num)

        {

            printf("(X=%d, Y=%d)\n", X, Y);

            \*count\_num = \*count\_num + 1;

            writeToBlk(buf, X, Y, re\_blk, &re\_blk\_count, write\_blk);

        }

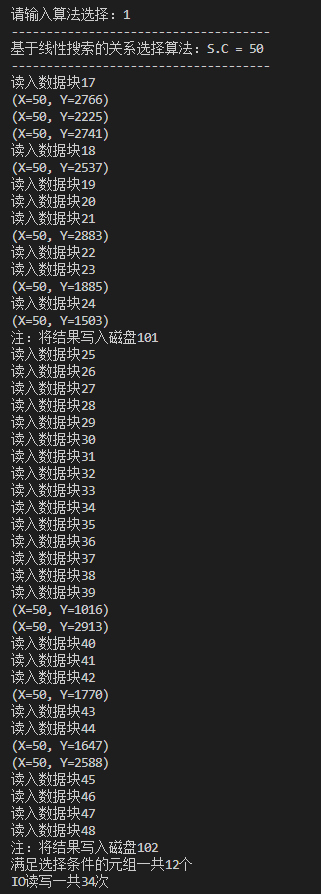
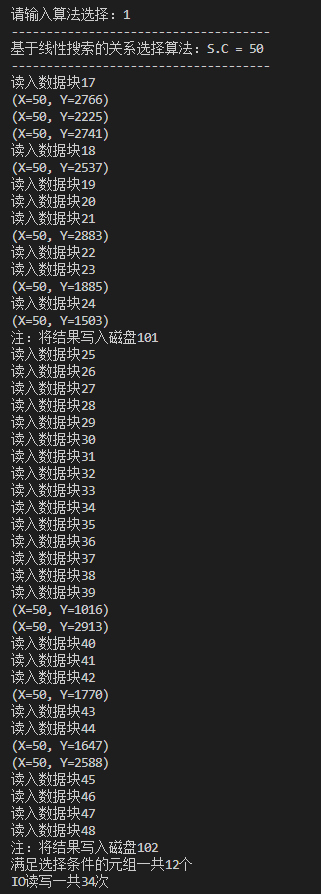
    }

    \*blk\_count = re\_blk\_count;

     getAddr(&addr, blk);

    freeBlockInBuffer(blk, buf);

实验结果：



1. **实现两阶段多路归并排序算法（TPMMS）**

问题分析：对于需要排序的表分为几个子集进行内排序，将内排好的子集读入首块到缓存区中，进行归并排序，直至所有子集的所有块排序完成，结果写入磁盘中。创建索引文件，利于后续基于索引的排序算法。

算法流程：

1. 内排序。
   1. 读入数据块直到缓存区满，这些数据块为一个子集。
   2. 将缓存区内的元组按照选择最小值进行排序，并将排序结果写入磁盘中，记录该子集存入磁盘的位置。
   3. 重复a,b过程，知道表所有数据块都进行读取且内排序。
2. 归并：
   1. 初始化读取所有子集的首块数据块。
   2. 当缓存区内有子集时，比较子集之间第一个元组大小，选择最小的元组，将元组写入缓冲区块作为结果。
   3. 有最小元组的数据块块的第一个元组更新。当该数据块元组后面有元组时，第一个元组为后面的元组。当该元组为当前数据块最后一个时，读入下一个数据块。当该子集所有数据块都已读入，没有下一个数据块时，删除该子集，释放占用区域。
   4. 重复b，c操作，直到有且仅有一个子集。将该子集的所有元组写入缓存区块。
   5. 将缓存区内剩余结果写入磁盘块，输出结果。

核心代码：

1. 子集基本信息：
   1. Blk\_start：子集开始磁盘块
   2. Blk\_end：子集结束磁盘块
   3. Blk：当前子集的磁盘块在缓冲区的指针
   4. Count：当前磁盘块需要比较的元组位置
   5. Next：下一个子集信息

typedef struct group\_list

{

    unsigned int blk\_start;

    unsigned int blk\_end;

    unsigned char\* blk;

    int count;

    struct group\_list\* next;

} GroupList;

1. 查找最小元组(X,Y)，以及其所在子集的基本信息
2. 最小元组写入缓存区块，获得子集用getNext()更新为下一个比较的元组，或没有元组删除子集。
3. 当只有一个子集时，将该子集的所有元组写入缓存区块。

    while(head != NULL)

    {

        small = findSmall(buf, head, &X, &Y);

        head = small->next;

        // small couple next or delete

        getNext(small, buf);

        writeToBlk(buf, X, Y, blk, &blk\_count, write\_blk);

        // only one blk compare

        if(head->next == head)

        {

            unsigned int addr;

            writeBlkLastToBlkBuf(buf, head->blk, head->count, blk, &blk\_count, write\_blk);

            getAddr(&addr, head->blk);

            while(addr <= head->blk\_end)

            {

                head->blk = readBlockFromDisk(addr, buf);

                head->count = 0;

                writeBlkLastToBlkBuf(buf, head->blk, head->count, blk, &blk\_count, write\_blk);

            }

            free(head);

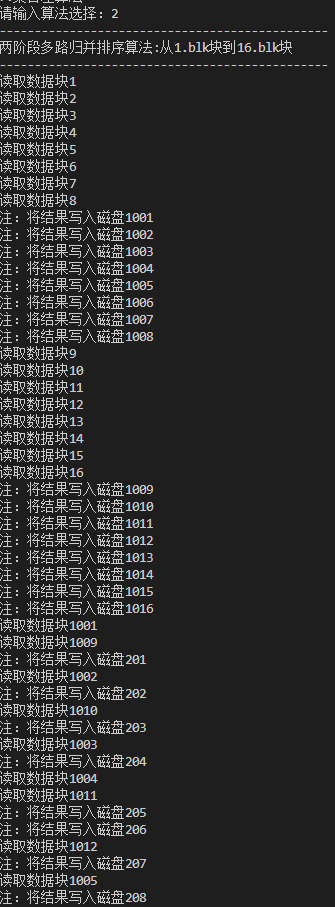
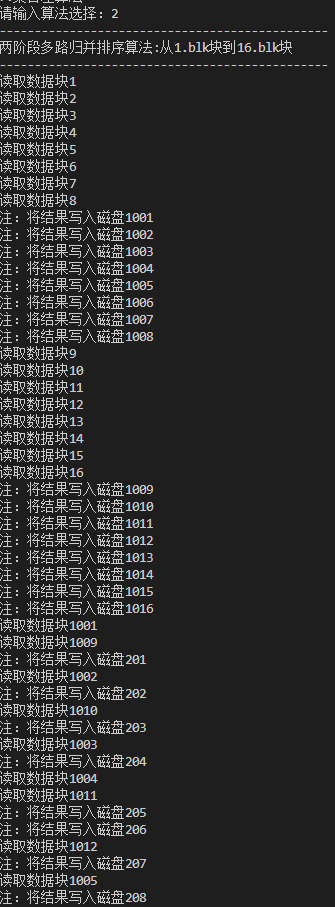
            head = NULL;

            break;

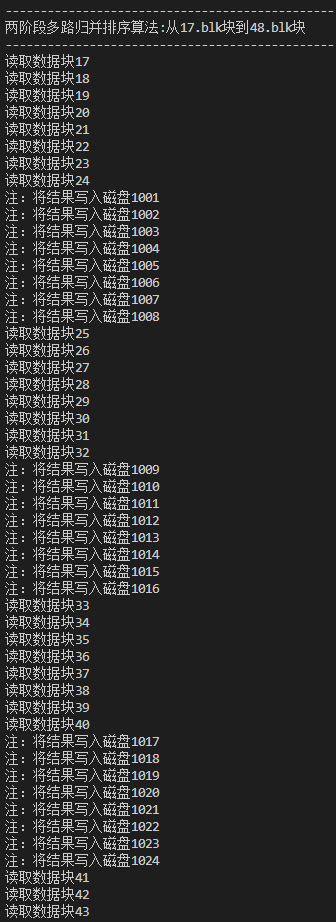
        }

    }

实验结果：









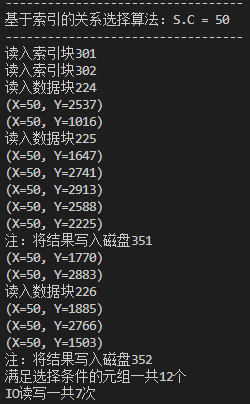
1. **实现基于索引的关系选择算法**

问题分析：根据2创建的索引文件，读取索引文件，获得所需数据所在磁盘数据的位置。读取数据块，输出结果。

实验流程：

1. 创建索引文件：
   1. 对于需要创建索引文件的数据块逐个读入，将块内最后一个元组的第一个元素作为索引字段，值为该数据块对应位置，存入缓存区块。
   2. 将缓存区块剩余内容写入磁盘。
2. 基于索引的关系选择算法:
   1. 从输入端获取需要选择的数据equal\_num
   2. 读取索引块，直到索引块的索引字段大于equal\_num，其值为数据块指针。
   3. 读取数据块指针，逐个读入数据块，查找结果，写入缓冲区块，直到第一个元素大于equal\_num
   4. 将缓冲区块剩余结果写入磁盘，输出结果。

实验结果：和1相比，大大减少了IO读写次数



1. **实现基于排序的连接操作算法（Sort-Merge-Join）**

问题分析：根据2的R和S排序结果，将S中第一个元素和R中第一个元素相等的元组都连接，写入磁盘块。

实验流程：

1. 初始化数据，R和S数据块指针blk\_R,blk\_S为R和S已排序块的第一个块，R的回转元组指针R\_begin为R的第一个块。
2. 当R或S不为空：
   1. 获得RX和SX
   2. 当RX<SX，R指向下一个元组，回转指针与R相同
   3. 当RX==SX，将S和R元组写入缓冲区块。R指向下一个元组位置。当R为空时，S指向下一个元组位置，如果下一个S和当前S值相同，则R返回R\_begin位置
   4. 当RX>SX时，S指向下一个元组位置，如果下一个S和当前S值相同，则R返回R\_begin位置

实验结果：



1. **实现基于散列的两趟扫描算法，实现交、并、差其中一种集合操作算法**

问题分析：R和S并集

实验流程：

1. 初始化
2. RX<SX，写入R元组，获得下一个R，更新R\_begin
3. RX>SX，写入S元组，获得下一个S
4. RX==SX==equal\_num，写入所有值为equal\_num的R里的元组。循环S里所有值为equal\_num的元组，每个循环如下。
   1. 更新R为R\_begin
   2. 判断R所指元组Y值是否与SY相等，不等则R为下一个元组，相等则跳出循环。
   3. 直到RX不为equal\_num，仍未找到SY相等的元组，则S写入缓存区块，S更新为下一个。否则S更新为下一个，不写入缓冲区块。
   4. 循环结束后更新R\_begin
5. 重复2,3,4直到R或S为空，写入不为空的表剩余所有元组
6. 写入磁盘，输出结果。

实验结果：



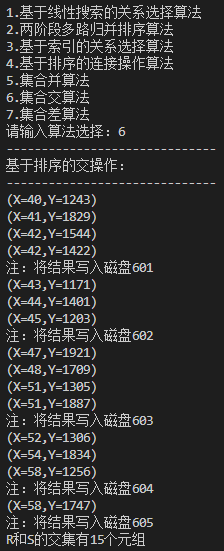
# 附加题

*对剩余的两种集合操作进行问题分析，并给出程序正确运行的结果截图。*

问题分析：R和S交集

实验流程：

1. 初始化
2. RX==SX&&RY==SY：保存R和S元组，S指向下一个元组，当如果下一个S和当前S值相同，则R返回R\_begin位置，否则更新R\_begin.
3. 仅RX==SX，获得下一个R元组
4. RX<SX，获得下一个R元组，更新R\_begin
5. RX>SX，S指向下一个元组，当如果下一个S和当前S值相同，则R返回R\_begin位置，否则更新R\_begin.
6. 重复上述2,3,4,5，直到R或S为空，写入不为空的表剩余所有元组
7. 写入磁盘，输出结果。



问题分析：R和S差集

实验流程：

1. 初始化
2. RX<SX，R更新为下一个元组，更新R\_begin
3. RX>SX，S写入磁盘块，S更新为下一个元组
4. RX == SX，循环S里所有值为equal\_num的元组，每个循环如下。
   1. 更新R为R\_begin
   2. 判断R所指元组Y值是否与SY相等，不等则R为下一个元组，相等则跳出循环。
   3. 直到RX不为equal\_num，仍未找到SY相等的元组，则S写入缓存区块，S更新为下一个。否则S更新为下一个，不写入缓冲区块。
   4. 循环结束后更新R\_begin
5. 重复2,3,4直到R或S为空，写入不为空的表剩余所有元组
6. 写入磁盘，输出结果。



# 总结

*总结本次实验的遇到并解决的问题、收获及反思。*

问题：环境有点奇奇怪怪。Debug版和release版的Codeblock运行代码会在基于索引的排序中卡死，原因在于Extmem.c中readBlockFromDisk的fgetc()卡死。但是，Codeblock中进行debug又是正确运行的。

解决：换成VSCode，通过编写makefile进行实验。初步怀疑是编译器的问题。

收获：代码速度越来越快了吧。还有不同环境下进行实验的时候，文件格式和编译的方式都有所了解。最关键的还是对于数据库的IO实现有了更深的认识。