|  |
| --- |
| 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: | 段裕 | | 学 号: | 180110704 | | 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。*

1. 理解索引的适用环境，掌握索引生成原理与工作原理；
2. 加深对算法I/O限制的理解；
3. 掌握关系选择、连接，集合的交、并、差等操作的实现算法。

\*其中，I/O限制是本实验的重点难点与趣味所在！

# 实验环境

*阐述本次实验的环境。*

编程语言：C语言

编程辅助IDE：Code Blocks

高级语言编译器：gcc version 8.1.0 (x86\_64-posix-seh-rev0, Built by MinGW-W64 project)

# 实验内容

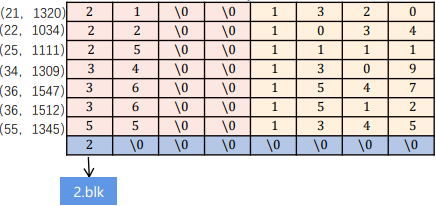
*阐述本次实验的具体内容。*

1. 项目背景

实验给出了一个API，包含一个内存结构体Buffer，以及与该数据结构配套的一系列操作（没错，是不是想到了当年的数据结构实验课，仅仅是实现一个链表我就被指针什么的折磨疯掉）。

实验假设内存和外存都是按块(BLK)存放的，每一个块64字节。内存最多放8个块，有效数据部分512字节，内存是520字节，多出的8个字节作为这8个有效数据块是否被占用的标志。实验的外存也就是本人电脑的磁盘，并假设外存无限大，另外外存BLK也是64字节的大小，可以装任意多个BLK。

那么接下来BLK里放的又是什么呢，实验给出了48个BLK，每个BLK可以视作一个8×8的二维字节数组，如图（取自指导书）：



每一个BLK都有8个行(LINE)，每一个LINE包含2(ATTRNUM)个整数，一个整数采用4(ELESIZE)个字节进行存储，每一个LINE又可以视为表的一行或者说一个元组，同时表包含2个字段。另外，一个BLK包含7个有效数据LINE，一个指针LINE。

最后，关系在哪里呢，刚刚讨论到，实验给出了48个BLK，每个BLK包含7个有效数据行，R分16个BLK存放，一共112个元组，S分32个BLK存放，一共224个元组。

这里，有同学在群里说到，这些块都是顺序存放的，那这个指针是没有用的，其实这个说法十分欠考虑。上面提到，内存、外存按块存取，关系中的数据也是分块存放，那么遍历一个关系R也就是遍历存放R的16个BLK，而为了访问具体的LINE，还必须遍历BLK的所有数据行，此时不可以认为BLK必然是满的，比如你找到20条符合条件的结果，此时需要3个块存放，但显然最后一个块不是满的，只有6条有效数据，而这些结果很可能在后面的问题中需要，**故而一开始就应该假定BLK不一定满，此时应该将指针作为判断块有效数据行的结束标志。**而在我的代码实现中，判断一行是指针是，**指针行的后一半必为空**，而使用这个条件作为指针行的判断就限制BLK的数量不应超过4位数，这一限制在本实验中是完全满足的。

另外，提一下实验代码框架的几个重要函数：

A、getNewBlockInBuffer()

尝试从Buffer(内存imem: 520B)申请一个BLK(64B)的空间

1. readBlockFromDisk

尝试从外存读一个BLK到内存

1. writeBlockToDisk

将指定的内存BLK写入外存的磁盘文件

1. freeBlockInBuffer

将该数据块标记为可用

注意到freeBlockInBuffer只是把数据块的有效位翻转，其中的数据还会残留，这些残留的数据可能影响接下来的操作，另外，整个项目仅能申请一次内存，于是在每一个算法执行结束后，内存可能还会有残存数据，这些数据可能影响下一个算法的执行，这样潜在的因素可能导致令人困惑费解的BUG。于是我编写了内存清空函数，它将清空整个imem，将其恢复为初始状态

1. clear\_mem()

2、实验要求

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=50的元组，记录IO读写次数，并将选择结果存放在磁盘上。（模拟实现 select S.C, S.D from S where S.C = 50）

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。（**不可定义长度大于10的整型或字符型数组**）

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=50的元组，并将选择结果存放在磁盘上。**记录IO读写次数，与（1）中的结果对比**。（模拟实现 select S.C, S.D from S where S.C = 50 ）

④ 实现基于排序的连接操作算法（Sort-Merge-Join）：对关系S和R计算S.C连接R.A ，并**统计连接次数，将连接结果存放在磁盘上**。 （模拟实现 select S.C, S.D, R.A, R.B from S inner join R on S.C = R.A）

⑤ 实现基于排序或散列的两趟扫描算法，实现并（S R） 、交（S R） 、差（S - R）**其中一种集合操作算法**。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

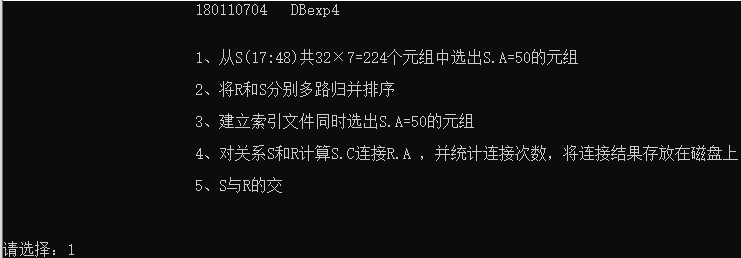
*对实验中的5个题目分别进行分析，并对核心代码和算法流程进行讲解，用自然语言描述解决问题的方案 。并给出程序正确运行的结果截图。*

1. **实现基于线性搜索的关系选择算法**

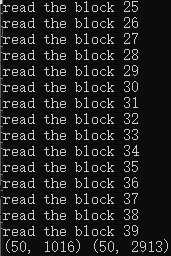
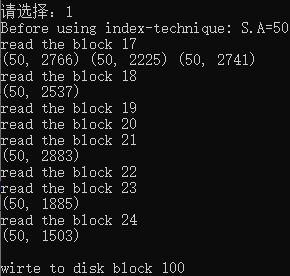
**问题分析：**S元组排放无序，遍历所有的BLK，对于每一个BLK，遍历每一个元组，对于每一个元组，提取第一个元素C，将C与键值KEY=50进行比较，若相等则将这一LINE复制到输出缓冲区，更新缓冲区有效LINE数，移动输出缓冲OUT\_BLK指针，当输出缓冲区满时，将输出缓冲区写入外存 (writeBlockToDisk)，清空OUT\_BLK指向内存块，重新申请一块可用的内存块作为输出缓冲区。

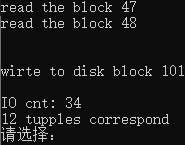
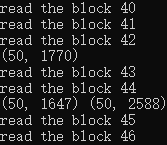
上面将函数输出行为描述的较为详细，而在遍历BLK时也有一些补充细节，根据传入的块号范围，读入一个块到内存（readBlockFromDisk），然后就是上面说到的遍历LINE、键值比较、输出处理，这一块到这里处理完毕，需要将其标记为可用（freeBlockInBuffer），同时清空这一内存块，这样就算完成了一个BLK的遍历，同时为下一个BLK做好了准备工作。

**实验结果：**



上图是系统运行时打印的提示信息，下图将展示问题1的结果：





上图结果表明读取了17到48一共32个块，结果有12条，写入文件100.blk和文件101.blk中，IO次数34，可以简单分析，遍历32个块32次读写，2个输出IO有2次，一共34次。

1. **实现两阶段多路归并排序算法（TPMMS）**

**问题分析：**这个问题是5个问题里最能体现IO复杂性与内存限制的问题，问题描述很清晰，外存需要排序的块较多，内存一次装不下，但要给这多块数据进行排序。思路也较为明了，既然内存装不下，就把数据分组，使得每一组都可以放进内存，每一组在内存里进行排序，这是第一阶段，这一阶段的任务是将待排序的数据块分组，将每一组装入内存，对每一组进行排序，使得每一组都是有序的。第二阶段便是经典的归并排序，每组每次进一个块，每次将一条最小的数据输出，这样一直到所有组输出完毕。

以上，是一个理论上的思路，还不够清稀，还是较笼统，在具体实现上还有许多细节，此时需要考虑具体数据结构，以及输入阶段、遍历阶段、输出处理等过程。接下来将较为详细的进行分析：

1. 第一阶段：分组，内排序

以R为例，R占16个BLK，内存只有8个BLK，一般来说分组时组内BLK的数量少于内存BLK数。但这里为了实现方便，不妨使内存满载，8个BLK一组，分为2组，分别是1-8和9-16这两组。以第一组为例，将其全部读进Buffer，此时得到8个指向各个BLK的指针。

到这里，由于数据结构的复杂性，接下来的实现就稍微复杂，但始终记住，编程往往遵循最简单的思想。假设我们抛开数据结构，最简单的方法就是一维数组的冒泡排序，两个嵌套的for循环就行，如果把这个过程抽象出来，就是：第一步，初始化指针，第二步访问指针数据，第三步比较和交换，最后移动指针。

一般而言，第三步是排序算法的重点，而如果具体到这里，移动指针则便成为了重点，这里待排序的是8个BLK，每个BLK有7条数据，我们把它看成1个一维数组，还是以上四步，移动指针的时候需要判断是否一个块结束，此时将指针移动到下一块，这是内排序的核心思想，不考虑指针移动细节，还是一维数组的冒泡排序。

1. 第二阶段：各个小组归并

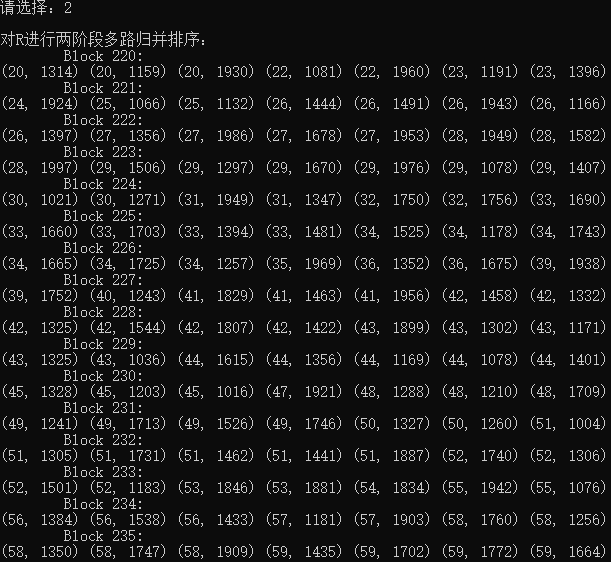
这里就体现了I/O复杂性。以R的1-8与9-16为例，假设内排序的结果写到了100-108和109-116里。

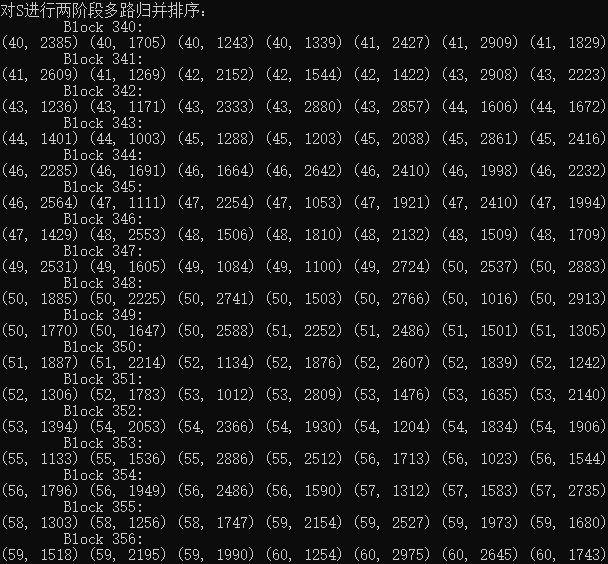
初始化：读入100.blk，读入109.blk，得到一组(2个)BLK指针，申请一块OUT\_BLK，得到输出缓冲块指针。

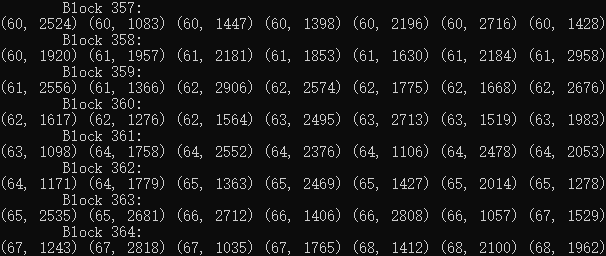
归并阶段：内存中此时有2个输入块，1个输出块，2个输入指针均指向各自的第一行数据，比较出最小值，将最小值所在LINE写入OUT\_BLK，输出指针移动，输出记录数更新，最小值所在组的指针移动向后移动一行。如果检测到输出缓冲区满，则输出缓冲区写入外存，输出缓冲区清空，输出记录清零，重新读取一块输出缓冲区。如果某输入块指针移动到尾行，则清空该内存块，读取所属小组的下一块装进内存。如果发现当前输入块指针指向的是当前小组的最后一块的最后一个数据行，那么当该指针指向的数据被写入到输出缓冲区时，指针不移动，也不清空这一块，而是采取一个技巧，**将块指针指向的数据修改为数据上限加一**，此时指针指向的数据还会参与比较，但一定不会输出到输出缓冲区。以这样一种策略，当所有小组的指针都移动到小组尾数据行时，根据比较出的最小值，如果这个值比设定的数据上限还大，那么表明归并排序结束，退出循环。注意，退出循环后，输出缓冲区往往会有残余数据，应该一并写入外存。

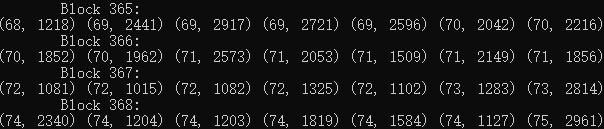
总结，这一问题的难点在于第二阶段，因为它操作的粒度是数据行级的，不是BLK级的，每一轮只处理一条数据，而不是每轮遍历一个数据块，遍历完后就可以清空内存块，因此操作粒度更加细节，指针移动的控制也更为复杂。

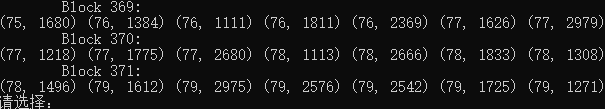
**实验结果：**











如上图，选择2，打印R和S归并排序的结果，注意系统展示时只是将已排序的块数据进行打印。可以看到，关系R排序好的16个BLK存放在220.blk到235.blk，关系S排序好的32个BLK存放在340.blk到371.blk。而第一阶段的排序结果，对于R，存放在200.blk到215.blk，对于S，存放在300.blk到331.blk，特点是每8个一组，组内数据有序。

1. **实现基于索引的关系选择算法**

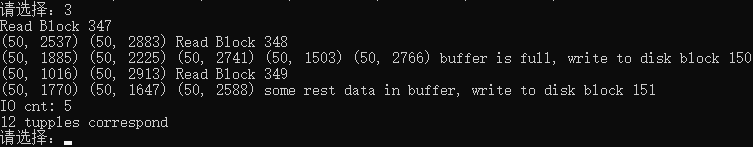
**问题分析：**上面谈到，S的已排序版存放在340.BLK到371.BLK这32个BLK中。设想对每一个BLK做一个索引，索引项包含该BLK首行数据的第一个字段，例如340 40 （块号 键值）。

建立这样的索引项后，设立头尾指针，遍历索引文件的索引项，将键值与KEY(50)比较，每当键值较小时头指针移动，尾指针跟随头指针，每当键值与KEY相等时，尾指针前移，直到有一个键值大于KEY时，停止遍历，那么头尾指针指向索引项对应的数据块就是需要读入内存进行查找。例如：

。。。 （47，1111）（49，1112）（50，1113）（53，1114）。。。

那么需要读的是1112块和1113块。

**实验结果：**



读了3个块，写了2次，一共只有5次IO操作。结果写入150.blk和151.blk

1. **实现基于排序的连接操作算法（Sort-Merge-Join）**

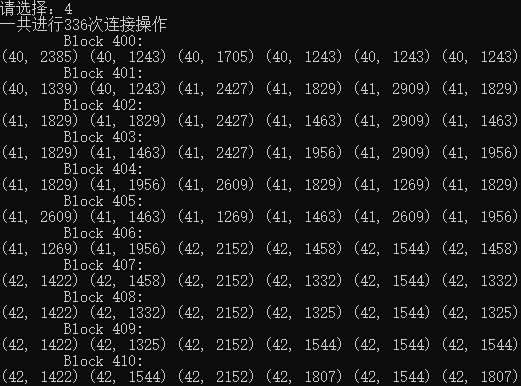
**问题分析：**利用已排序的块实现连接操作。这里根据r.A和s.C进行等值连接。按照类似索引中思路，需要选出可能进行连接的数据块。例如

A（47，1234）（48，2111）。。。。（70，3221）

B（25，1234）（27，2111）。。。。（45，3221）

这两个块必然不可能有可以连接的数据，因为A的头比B的尾还大，因此，但凡出现A.head > B.tail 或者 B.head > A.tail的情况，就可以直接跳过。在实验给的数据集上，可以使连接操作快很多。

**实验结果：**



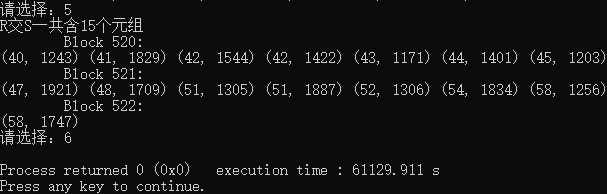


可以看到一共有336次连接操作，一共分112个块存放，注意到此时每一个块只存了6个元组，因为连接后，每一行数据包含4个数据，相当于原来的两行数据，因此为了保持数据的完整性，每块存放3行（对于连接后的新表）数据。

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

**问题分析：**我挑一个简单的交集来做，准确地说，和上面的连接貌似基本没区别，把连接条件变成r.A=s.C和r.B=s.D就行。

**实验结果：**



可见有15个元组是R和S共有的，选择6，退出程序。如果仔细观察，EXCUTION TIME为什么这么长，因为写报告的时候程序一直没关，前后差不多一天。

# 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。*

其实也是可以做的，但我觉得没必要，毕竟实验所要求掌握的我已经掌握了。

# 总结

*总结本次实验的遇到并解决的问题、收获及反思。*

遇到的问题实际上上面已经说到了，我想到的解决思路也已经说的较为详细，下面我可能更多的谈一谈感受。

一个是C语言这个高级语言本身，C语言指针是其一大特色，同时也只有C语言区分无符号数和符号数，C语言可能更贴合实际机器。在操作指针时，思路逻辑应当十分清晰，指针应当指向哪里，操作的是哪一块数据，指针如何进行移动，都应当十分精准。

在此次编程中，程序员（本人）更能体会到内存是一个字节数组，程序需要巧妙的利用这个字节数组定义数据结构，定义操作，通过指针，一些操作会变得十分的简洁明了。而要利用好这个字节数组，我发现，基址偏移思想是一个很重要的思想，基指针不要移动，利用偏移进行访问，此时程序可能定义较多的变量，但这些中间变量使得程序编写更加高效，思路更加清晰。

编程往往遵循最简洁的思想。不要试图把一些本身就复杂的事情越想越复杂，不要给自己挖坑。例如，内排序时，使用冒泡排序法，抽象出排序的几个阶段，此时可以将多个BLK视作一个大的一维数组等等。

最后，这个实验课给我的收获不亚于当年的操作系统的文件系统，希望能一直办下去！