哈尔滨工业大学(深圳)

《数据库》实验报告

实验五

查询处理算法的模拟实现

学　院: 计算机科学与技术

姓　名: 孙铎

学　号: 200110503

专　业: 计算机科学与技术

日　期: 2023-01-01

一、 实验目的

阐述本次实验的目的。

- 理解索引、散列的作用；
- 掌握关系选择、投影、连接、集合的交、并、差等操作的实现算法；
- 加深对算法 I/O 复杂性的理解；
- 理解两阶段多路归并排序算法的思想、实现与应用；
- 理解、实现并运用简单的索引。

二、 实验环境

阐述本次实验的环境。

- Windows 10 操作系统
- gcc version 8.1.0 (x86_64-win32-seh-rev0, Built by MinGW-W64 project)
- CodeBlocks

三、 实验内容

阐述本次实验的具体内容。

- 基于 ExtMem 程序库，模拟实现数据库的一些查询处理算法；
- 实现关系选择、连接操作算法；
- 实现简单的索引，并实现基于索引的关系选择算法；
- 实现集合并、交、差操作算法。

四、 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。

(0) 对 extmem.c 的修改以及 utils.c 工具函数的封装

- 修改 extmem.c：
  - 修改 freeBlockInBuffer 函数：freeBlockInBuffer 函数原本只将内存块标志位设置为可用，并没有真正清空内存块数据，这里将其修改为，**调用 memset 将内存块全部置 0** 并设置**标志位**为可用；
  - 修改 writeBlockToDisk 函数：将 writeBlockToDisk 函数中将内存块标志位设置为可用的代码修改为**对 freeBlockInBuffer 函数的调用**，这样实现的效果是一致的。
- 封装 utils.c 中的工具函数：
  - record2XY 函数：
    - 输入：
      - blk：内存块指针；
      - recordNum：要**读出**的块中的记录；
      - X：接收该条记录的**前 4 个字节**的传出参数；
      - Y：接收该条记录的**后 4 个字节**的传出参数。
    - 输出：无
    - 函数实现：
      1. 声明字符数组 str，长度为 5；
      2. 通过 recordNum 进行下标运算，读取 blk 对应记录的**前 4 个字节**到 str 中，并调用 **atoi 函数**将 str 转换成 int 型整数赋值给传出参数 X；
      3. 通过 recordNum 进行下标运算，读取 blk 对应记录的**后 4 个字节**到 str 中，并调用 **atoi 函数**将 str 转换成 int 型整数赋值给传出参数 Y。
  - XY2record 函数：
    - 输入：
      - blk：内存块指针；
      - recordNum：要**写入**的块中的记录；

- X：向记录**前 4 个字节**写入的数值；
- Y：向记录**后 4 个字节**写入的数值。

◆ 输出：无

◆ 函数实现：

1. 声明字符数组 fourBytes，长度为 4；
2. 判断 X **是否等于-1**，如果不等于-1，则**调用 sprintf 将 X 转换为字符串**存储至 fourBytes 中，并通过 recordNum 进行下标运算，将 fourBytes 中的 4 个字节写入 blk 对应记录的**前 4 个字节**，如果等于-1，则将**数值 0** 写入 blk 对应记录的**前 4 个字节**；
3. 判断 Y **是否等于-1**，如果不等于-1，则**调用 sprintf 将 Y 转换为字符串**存储至 fourBytes 中，并通过 recordNum 进行下标运算，将 fourBytes 中的 4 个字节写入 blk 对应记录的**后 4 个字节**，如果等于-1，则将**数值 0** 写入 blk 对应记录的**后 4 个字节**。

■ nextAddr 函数

◆ 输入：

- blk：内存块指针

◆ 输出：blk 块的后继块地址值

◆ 函数实现：

1. 声明字符数组 str，长度为 5；
2. 通过下标运算，读取 blk **最后 8 个字节的前 4 个字节**到 str 中，**调用 atoi 函数**将 str 转换成 int 型整值，并返回该值（约定将**后继块地址**存储在 blk **最后 8 个字节的前 4 个字节**中）。

■ shiftRecord 函数

◆ 输入：

- buf：**内存缓冲区**指针
- blk：**内存块**指针的指针

- recordCnt：存储**当前读到的记录数**的整型指针
- maxRecordCnt：**最大记录数**

◆ 输出：

- -1 代表之前**已经读完了所有记录**，不能再继续将 recordCnt 后移到下一条记录；
- 0 代表成功将 recordCnt **后移至下一条记录**，且下一条记录在同一块中，**不需要读入新块**；
- 大于 0 代表成功将 recordCnt **后移至下一条记录**，且下一条记录在下一块中，需要**读入新块**，返回新块地址值。

◆ 函数实现：

1. 解引用 recordCnt 并**自增 1**；
2. 如果解引用 recordCnt 的值已经**达到 maxRecordCnt**，则返回-1；
3. 如果根据解引用 recordCnt 的值判断出**下一条应当在下一块中**，则调用 nextAddr 函数，传入 blk 解引用，得到**后继块地址**，再调用 freeBlockInBuffer 函数释放原 blk 指向的内存块，调用 readBlockFromDisk 函数**将新块读入 blk** 中，并返回后继地址；
4. 如果没有达到最后一条记录，也不需要读取下一块，返回 0 即可。

■ writeToOutBlk 函数

◆ 输入：

- buf：**内存缓冲区**指针；
- outBlk：**内存块**指针的指针；
- recordCnt：存储**当前输出的记录数**的整型指针；
- outAddr：存储 outBlk **要输出到的磁盘块号**的整型指针；
- X：要写到 outBlk 上的记录的**前 4 个字节值**；
- Y：要写到 outBlk 上的记录的**后 4 个字节值**。

- 输出：
  - -1 代表**写磁盘块失败**；
  - 0 代表**正常**。
- 函数实现：
  1. 如果根据 recordCnt 解引用判断出**当前 outBlk 已经写满**，则调用 XY2record 将 **outAddr 解引用再加 1** 写入 outBlk 最后 8 个字节，并调用 writeBlockToDisk 函数将 outBlk **写入到地址为 outAddr 解引用的磁盘块**上（如果调用 writeBlockToDisk 函数出错，则返回-1），并将 **outAddr 解引用自增 1**，再调用 getNewBlockInBuffer 函数为 outBlk 申请新的空闲内存块，并输出写入的信息；
  2. 无论当前 outBlk 是否写满，都调用 XY2record 函数**将 X 和 Y 的值写入 outBlk 块**对应的 **recordCnt 解引用再模 7** 的记录上，然后将 **recordCnt 解引用自增 1**；
  3. 返回 0。

(1) 实现基于线性搜索的关系选择算法

问题分析：

- 要将关系所在的磁盘块依次**读入内存**，并要依次判断各条记录**是否符合选择条件**，并**输出符合选择条件的记录**到磁盘块上。因此，需要实现的重点操作是：
  - 遍历读入的磁盘块中各条记录，**正确解析其数据值**，并判断是否符合选择条件；
  - 将符合选择条件的记录**暂存至新的内存块**，待内存块满时**输出至磁盘块**，同时也要设置好其**后继地址**。
- 核心代码与算法流程：

a) 定义变量：（只列出了部分值得说明的）
- 1 个**内存缓冲区 buf**；
- 1 个**内存块指针 blk** 用于存放**关系 S 的数据块**；

- ◆ 1 个**内存块指针 resBlk** 用于暂存满足选择条件**待输出的记录**；
- ◆ 1 个**记数变量 rowCount** 记录满足选择条件的**记录条数**。

b) 调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区；

c) 从**起始磁盘块 17** 开始，通过解析其后继地址，**遍历关系 S 的所有块**，并在每次遍历时执行如下操作：

i. 调用 **readBlockFromDisk 函数读取**遍历到的磁盘块至内存块，其指针存放至 blk 中；

ii. 遍历块中的**前 7 条记录**，调用 record2XY 函数将每条记录的 **S.C 和 S.D** 分别存入**变量 X 和变量 Y** 中；

iii. 如果 **X 等于选择条件 128**，则输出 X 和 Y 的值，令 **rowCount 加 1**，并判断 resBlk 中之前是否已经**写满了 7 条记录**，如果已经写满，则调用 XY2record 函数，将**后继块地址**写入 resBlk 的最后 8 个字节，并调用 writeBlockToDisk 函数**将 resBlk 输出至对应磁盘位置**，再调用 getNewBlockInBuffer 函数**重新获取空闲块**给 resBlk，无论 resBlk 是否已经写满，最后都要再调用 XY2record 函数，**将当前的 X 和 Y 值写入** resBlk 的下一条空记录中；

iv. 调用 nextAddr 函数读取**后继块地址**，调用 freeBlockInBuffer 函数释放 blk，继续遍历关系 S 的下一块。

d) 将**最后一个不超过 7 条选择结果记录**的 resBlk 写入磁盘。因为之前每次都是写满一块后，**下一次要写入新的块时才将上一块写入磁盘**，所以遍历结束后，只要存在满足选择条件的记录，resBlk 中就一定有**剩余待输出的选择结果记录**；

e) 输出一些结果信息，详见下方的**实验结果**。

f) 调用 freeBuffer 函数**释放内存缓冲区 buf**

实验结果：

```
--------------------------------
基于线性搜索的选择算法 S.C = 128:
--------------------------------
(X=128, Y=684)
(X=128, Y=431)
(X=128, Y=615)
(X=128, Y=429)
(X=128, Y=584)
(X=128, Y=592)
(X=128, Y=457)
(X=128, Y=720)
(X=128, Y=447)
(X=128, Y=871)
注：结果写入磁盘：100
注：结果写入磁盘：101

满足选择条件的元组一共10个

IO读写一共34次


Process returned 0 (0x0)   execution time : 0.109 s
Press any key to continue.
```

(2) 实现两阶段多路归并排序算法（TPMMS）

问题分析：

- 首先需要将磁盘块**分组读入内存**，进行**内排序**，输出**中间结果**至磁盘，再将磁盘上的内排序结果按照**归并排序的方式读入内存处理**，并将归并结果**输出至磁盘**，最后**擦除内排序中间结果**。因此，需要实现的重点操作是：
  - 内排序：
    - 将磁盘块**分组读入内存**；
    - 对每组磁盘块使用内排序算法**排序所有记录**；
    - 将内排序中间结果分块输出至磁盘。
  - 归并排序：
    - 为每组磁盘块安排好内存块，**每组分别读入一部分到内存**；
    - **横向比较每组内存块的记录**，按排序方式每次选择一条记录输出；
    - 当一组内存块的记录全部处理完时，需要**换入该组的下一块**，当该组全部块都处理完时，需要对其进行标记。
- 核心代码与算法流程：
  a) 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小

520B，块大小 64B 的内存缓冲区；

b) 调用封装好的**函数 internalSort** 分别对关系 R 和关系 S 分组进行**内排序**，internalSort 函数定义如下：

i. 声明大小为 8 的内存块指针数组 blks，用于存放**关系的一组数据块**；

ii. 从关系存放的起始块开始，通过 nextAddr 函数获取后继块地址，**循环读入关系的每个数据块**；

iii. 每读入 8 个数据块，进行一次**冒泡排序（升序）**，排序 8 个数据块中的所有记录，利用冒泡排序的规则，遍历所有记录，调用 record2XY 函数**将相邻两条记录字段 C 和字段 D 的值分别读入** X1、X2 和 Y1、Y2，并比较，如果 X1Y1 比 X2Y2 大，则再调用 XY2record 函数**交换相邻两条的记录值**，重新写入对应磁盘块的记录上；

iv. 冒泡排序结束后，调用 writeBlockToDisk 函数，**循环输出该组的每一块至磁盘**，并输出中间结果存储到的磁盘块号，除了关系的最后一个数据块，其他数据块都要调用 XY2record 函数将**后继地址**写入块的最后 8 个字节上。

c) 调用封装好的**函数 externalSort** 分别对关系 R 和关系 S 分组进行**归并排序**，externalSort 函数定义如下：

i. 定义变量：（只列出了部分值得说明的）

- blks：数组，存储**各组读入内存的数据块指针，每组分配 1 个内存块**；
- recordCnts：数组，存储各**组读到了第几条记录**；
- blk：**1 个**暂存归并排序**输出记录的数据块**；
- recordCnt：**循环计数变量**，存储下一次应当将结果写到 **blk 上第几条记录**，从 0 开始，最大值为 6，到达 6 后循环回到 0。

ii. 将**每组的第 1 个块**先读入内存，其指针分别存储在 blks 数组中；

iii. 调用 getNewBlockInBuffer 函数为 blk 申请可用内存块；

iv. 循环从**每组**对应的内存块中**分别调用 record2XY 函数读入一条记录**，其中对应的**块指针**存储在 blks 数组中，对应块上的**记录号**存储在 recordCnts 数组中，如果某一组已经**处理完所有记录**，会将其 recordCnts 数组对应元素**标记为-1**（在后面的步骤中有体现），先比较 A 或 C 字段，A 或 C 字段相等再比较 B 或 D 字段，将**各组最小的记录值**存储到变量 Xpre 和 Ypre 中，并将**选择到最小记录的组号**存储在变量 select 中（**升序**排序，所以选择最小的先输出）；

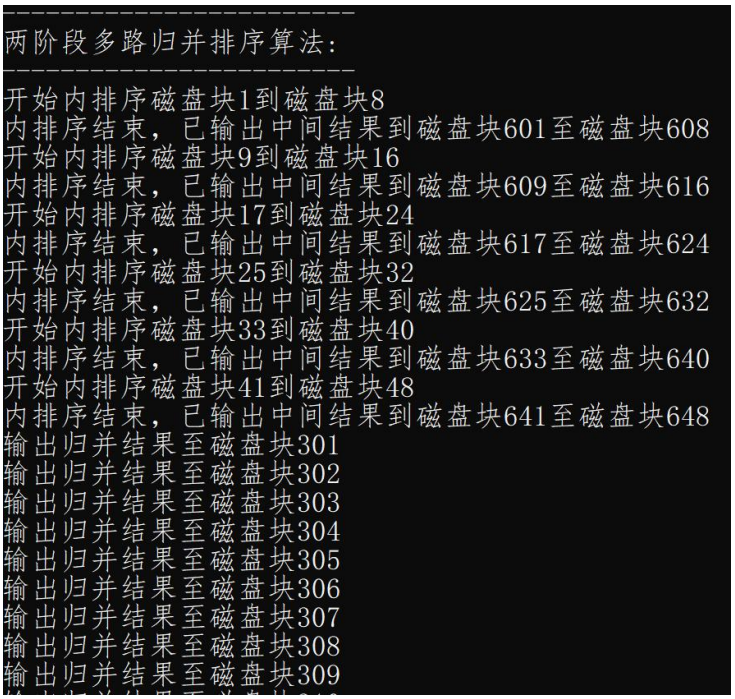
v. **将最小记录调用 XY2record 函数写入 blk** 的 recordCnt 条记录处，并令 recordCnt 加 1，如果 blk 已经**写满了 7 条**，则调用 writeBlockToDisk 函数将 blk **输出至磁盘对应位置**，并再次调用 getNewBlockInBuffer 函数为 blk 申请可用内存块，同时，如果不是最后一块，则还要调用 XY2record 函数向块中写入**后继地址**（因为已知关系 R 和关系 S 的记录总数为 7 的倍数，所以这里简化处理，只在写满 7 条时输出，也能保证输出关系的所有记录没有剩余）；

vi. 令 recordCnts[select]加 1，即本次被选择的组要**后移一条记录**，如果该组已经处理完**最后一条记录**，则将 recordCnts[select]**标记为-1**，如果不是最后一条记录，但通过 recordCnts[select]判断出来已经**处理完了一整个块**，则调用 nextAddr 函数读出**后继地址**，并调用 freeBlockInBuffer 函数释放 blks[select]，然后调用 readBlockFromDisk 函数读入后继块到 blks[select]中，回到步骤 iv. 继续参与循环处理，直到所有组的所有记录均处理完。

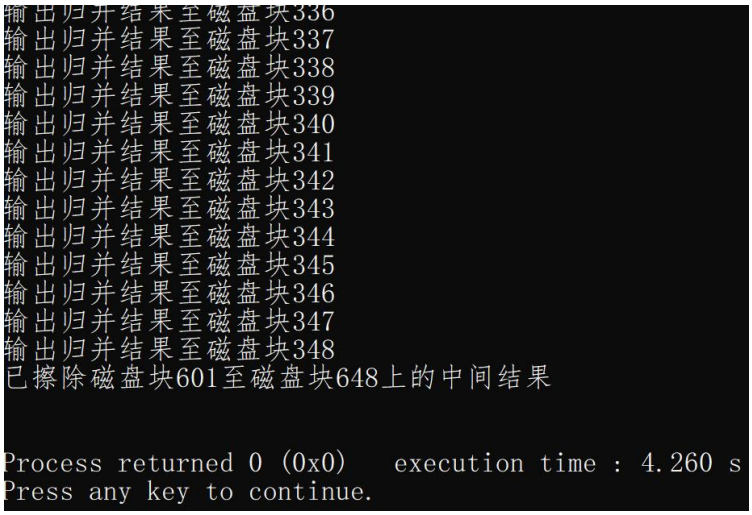
d) 循环调用 dropBlockOnDisk 函数，**擦除内排序中间结果**；

e) 调用 freeBuffer 函数**释放内存缓冲区 buf**。

实验结果：



……



(3) 实现基于索引的关系选择算法

问题分析：

- 首先要**设计索引块结构**，为排好序的关系 S **建立索引**，然后**遍历各索引块**，根据**索引项指针**快速定位满足条件的记录应该出现在哪些块中，然后遍历这些块中的记录，**选择满足筛选条件的记录**并输出。因此，需要实现的重点操作是：
  - 遍历排好序的关系 S，为**每一块建立一条索引项存放在索引块**中，并输出索引块；

- 遍历索引块，对比**索引项上的值和筛选条件**，通过**索引项上的指针**快速定位满足条件记录所在块，选择记录并输出。

- 核心代码与算法流程：

a) 调用 buildIndex 函数，**为 S.C 建立索引**，返回**索引块的起始地址**，其中 buildIndex 函数的定义如下：

i. 定义变量：（只列出了部分值得说明的）

- dataBlk：存储读入内存的关系 S 的 **1 个数据块**的指针；
- idxBlk：存储要输出至磁盘的 **1 个索引块**指针。

ii. 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区（注：这里之所以单独定义一个内存缓冲区，是因为**后面还要统计基于索引的选择操作的 I/O 次数**，如果使用同一个缓冲区建立索引，则 **I/O 次数会叠加**，所以**建立索引时单独使用一个缓冲区**）；

iii. 调用 getNewBlockInBuffer 函数为 idxBlk 申请可用内存块；

iv. 通过起始地址、nextAddr 函数和 readBlockFromDisk 函数，读入磁盘块到 dataBlk 中，**循环遍历排好序的关系 S 的所有数据块**，每次遍历一个数据块时，调用 record2XY 函数读取**数据块第一条记录**，用于创建**索引项**，如果索引块中已经写满了 7 个 8 字节索引项，则**先输出该索引块**，调用 XY2record 写入**后继地址**，然后调用 writeBlockToDisk 函数输出至磁盘，再调用 getNewBlockInBuffer 函数重新为 idxBlk 申请空闲内存块，无论索引块是否写满 7 项，都要再调用 XY2record 函数将该条记录 **C 字段值和块地址写入索引块**的 8 个字节上，最后调用 freeBlockInBuffer 函数释放 dataBlk，待下一次循环重新读入磁盘块到 dataBlk 中（注：要保证**不同索引项的索引值不重复**，这可以通过记录上一索引项的值，并与当前值进行对比来实现）；

v. 调用 writeBlockToDisk 函数**输出最后一个不超过 7 项的索引块**至磁盘；

vi. 调用 freeBuffer 函数**释放内存缓冲区 buf**。

b) 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区；

c) 定义变量：（只列出了部分值得说明的）

- dataBlk：存放读入内存的关系 S **数据块**指针；
- idxBlk：存放读入内存的 S.C **索引块**指针；
- resBlk：暂存待输出至磁盘的筛选记录。

d) 通过索引块起始地址、nextAddr 函数和 readBlockFromDisk 函数，**读入索引块到 indexBlk** 中，**顺序遍历各索引块**，每次遍历执行如下操作：

i. 通过 record2XY 函数读入索引块中的**各条索引项**，如果当前索引项的值**大于目标值 128**，或者当前索引项是**空的**（这说明已经找完了最后一个索引项），说明**从上一个索引项指针的数据块到当前索引项指向的数据块**（如果已经找完了最后一个索引项，则是直接到最后一个数据块）中可能出现与目标值 128 匹配的记录，则需要**回退索引项，遍历对应数据块记录**，标记 flag 为 1，退出当前遍历索引项的循环；

ii. 结束索引项循环后，判断 flag，如果 flag 为 1，则调用 record2XY 函数**读入上一条索引项**（如果上一条索引项**不在当前索引块**，则还需要**读入上一索引块**，如果**不存在上一索引块**，说明关系中**没有符合条件的记录**，直接退出循环即可），根据索引项指向的数据块，遍历有可能存在符合筛选条件记录的所有数据块，读入数据块时输出信息，遍历数据块每条记录，如果记录符合筛选条件，则将记录**写入结果暂存块 resBlk**，在正式写入前判断结果块是否已满（达到 7 条记录），如果已满，则写入后继地址，先将 resBlk 输出至磁盘，再写入，如果当前遍历到的记录已经大于选择条件值，由有序性可知，之后一定不存在满足条件的记录，直接退出循环；

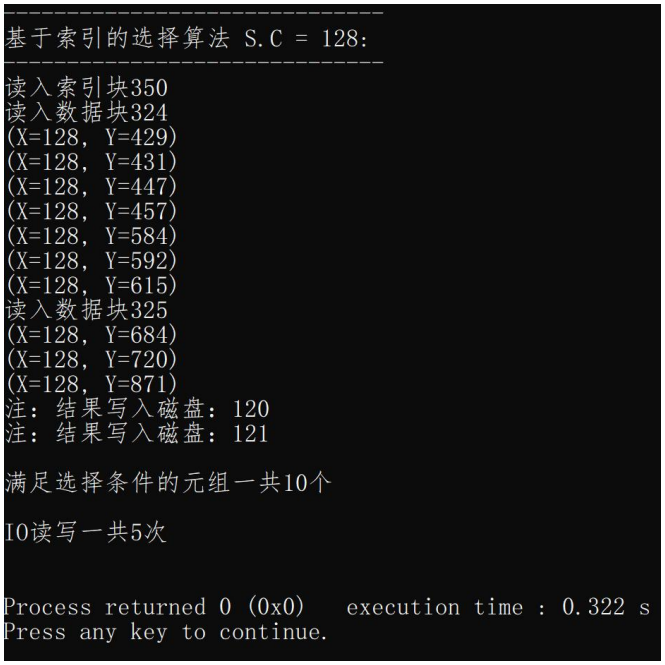
iii. 如果 flag 为 0，则输出信息“没有满足条件的元组”，并继续遍历下一索引块。

e) 将 resBlk 中不超过 7 条记录的最后一块写入磁盘

f) 输出一些结果信息，详见下方的**实验结果**。

g) 调用 freeBuffer 函数**释放内存缓冲区 buf**。

实验结果：



- 可以看出，相比于线性选择，基于索引的选择，**结果不变，但 IO 读写次数明显减少**

(4) 实现基于排序的连接操作算法（Sort-Merge-Join）

问题分析：

- 要分别**读取排好序的关系 R 和关系 S 至内存块**，以互相**协调**的方式**分别遍历**，当遍历到的记录**满足连接条件** R.A=S.C 时，进行输出。因此，需要实现的重点操作是：
  - **协调**好关系 R 和关系 S 记录的**遍历顺序**，防止遗漏本该连接的记录；
  - 当遇到两条符合连接条件的记录时，**处理好记录的连接字段连续相等的情况**，即两个关系中符合连接条件的记录紧挨着后面的记录字段值是相等的，依然符合连接条件，这种连续也有可**能跨越了一块**，

需要处理好；

- 在**不满足连接条件**时，正确**后移关系的记录**。

- 核心代码与算法流程：

a) 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区；

b) 定义变量：（只列出了部分值得说明的）

- blk：暂存**输出结果**的 1 个内存块；
- blks：数组，存储关系 R 和关系 S **读入内存的数据块**；
- recordCnts：数组，记录关系 R 和关系 S **当前正处理的记录号**。

c) 先将关系 R 和关系 S 的**第 1 个数据块**读入 blks，将 **recordCnts 均置为 0**（注：读入的是已经排好序的关系 R 和关系 S）；

d) 调用 getNewBlockInBuffer 函数为 blk 申请空闲内存块；

e) 调用 record2XY 函数分别**读入关系 R 和关系 S 的当前记录**到变量 A、B、C、D 中；

f) 比较 A 和 C 的值，按**三种不同的结果**分别执行下列三种操作：

- 如果 **A < C**，则调用 shiftRecord 函数（定义在 utils.c 中）让**关系 R 后移一条记录**（因为数据块中的记录是**升序**的，只有较小的一方后移变大，才有可能相等从而连接），如果关系 R 已经**读取完了最后一条记录**，则**结束**整个循环；
- 如果 **A > C**，则调用 shiftRecord 函数让**关系 S 后移一条记录**，如果关系 S 已经**读取完了最后一条记录**，则**结束**整个循环；
- 如果 **A = C**，则先**记录下关系 R 当前块和当前记录**，然后从**关系 S 的当前记录**开始，向后遍历**每一个字段 C 与当前记录相等的记录**，对于关系 S 中每一条这样的记录，将其**与关系 R 的当前记录和关系 R 后面与字段 A 当前记录相等的所有记录**进行**连接**，调用 XY2record 函数向 blk 中输出连接结果（每条连接结果占 16 个字节，规定**每块中最多存 3 条连接结果**，最后 8 个字节依然存放后继地址），其中，**记录后移**的操作都由 **shiftRecord 函数**实

现，关系 R 从当前记录到后面所有字段 A 相等记录的循环遍历，通过**之前记录下的关系 R 当前块和当前记录**实现（因为这个循环遍历可能是**跨块**的，所以也记录了当前块），遍历过程中，如果**关系 S 遍历到了最后一条记录**，则说明不再有连接结果，直接**结束整个循环**（因为遇到 A = C 时，规定让**关系 S 在外层循环**），如果**关系 R 遍历到了最后一条记录**，可以标记好关系 R 已经读完，但先**不能结束循环**，依然要**等关系 S 的所有记录外层循环结束**，才能结束整个循环（遇到 A = C 时，规定让关系 R 在外层循环）。

g) 重新执行**步骤 e)**，循环读入下一条记录，直到已经**读完了其中一个关系的最后一条记录**，或在**步骤 f)中就已经遍历完了**所有可能的连接情况，才**结束**整个循环；

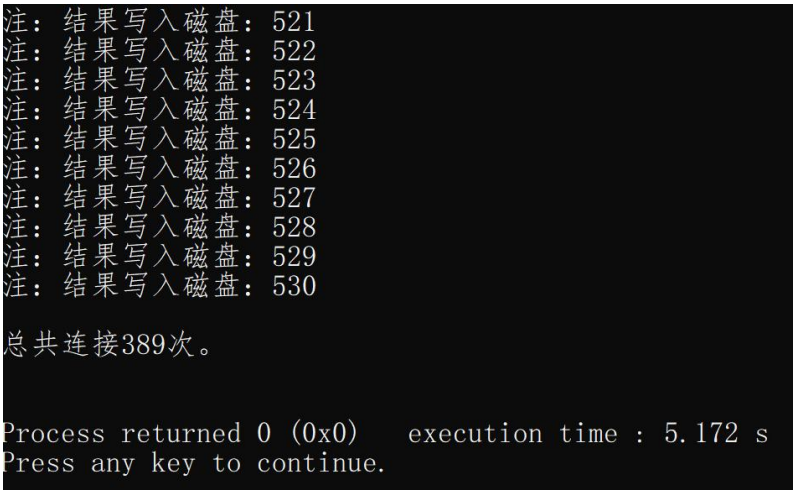
h) 输出 blk 中剩余的连接记录至磁盘；

i) 输出一些结果信息，详见下方的**实验结果**。

j) 调用 freeBuffer 函数**释放内存缓冲区 buf**。

实验结果：

...



(5) 实现基于排序的两趟扫描算法，实现交集合操作算法

问题分析：

- 要分别**读取排好序的关系 R 和关系 S 至内存块**，以互相**协调**的方式**分别遍历**，当遍历到的记录**满足交条件** R.A=S.C 且 R.B=S.D 时，进行输出。因此，需要实现的重点操作是：
  - **协调**好关系 R 和关系 S 记录的**遍历顺序**，在**满足交条件**时输出该条记录；
  - 在**不满足交条件**时，正确**后移关系的记录**。
- 核心代码与算法流程：
  a) 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区；
  b) 定义变量：（只列出了部分值得说明的）
     - blk：暂存**输出结果**的 1 个内存块；
     - blks：数组，存储关系 R 和关系 S **读入内存的数据块**；
     - recordCnts：数组，记录关系 R 和关系 S **当前正处理的记录号**。
  c) 先将关系 R 和关系 S 的**第 1 个数据块**读入 blks，将 **recordCnts 均置为 0**（注：读入的是已经排好序的关系 R 和关系 S）；
  d) 调用 getNewBlockInBuffer 函数为 blk 申请空闲内存块；
  e) 调用 record2XY 函数分别**读入关系 R 和关系 S 的当前记录**到变量 A、B、C、D 中；

f) 比较 A、B、C、D 的值，按**三种不同的结果**分别执行下列三种操作：

- A = C 且 B = D：调用 writeToOutBlk 函数将 A 和 B 输出至 blk 对应记录处，输出信息，并调用 shiftRecord 函数将**关系 R 和关系 S 分别后移一条记录**，如果已经读完了关系 R 或关系 S 的最后一条记录，则结束整个循环；
- A < C，或 A = C 且 B < D：调用 shiftRecord 函数将**关系 R 后移一条记录**，如果已经读完了关系 R 的最后一条记录，则结束整个循环；
- A > C，或 A = C 且 B > D：调用 shiftRecord 函数将**关系 S 后移一条记录**，如果已经读完了关系 S 的最后一条记录，则结束整个循环。

g) 重新执行**步骤 e)**，循环读入下一条记录，直到已经**读完了其中一个关系的最后一条记录**，或在**步骤 f)中就已经读完了其中一个关系的所有记录**，才**结束**整个循环；

h) 输出 blk 中剩余的交记录至磁盘；

i) 输出一些结果信息，详见下方的**实验结果**。

j) 调用 freeBuffer 函数**释放内存缓冲区 buf**。

实验结果：

```
------------------------
基于排序的集合的交算法:
------------------------
(X=120, Y=418)
(X=120, Y=827)
(X=122, Y=546)
(X=123, Y=477)
(X=125, Y=886)
(X=127, Y=767)
(X=128, Y=447)
注：结果写入磁盘：140
(X=129, Y=430)
(X=130, Y=436)
(X=130, Y=656)
(X=134, Y=437)
(X=139, Y=461)
(X=140, Y=610)
注：结果写入磁盘：141

S和R的交集有13个元素。


Process returned 0 (0x0)   execution time : 0.302
Press any key to continue.
```

五、 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。

(1) 实现基于排序的两趟扫描算法，实现并集合操作算法

问题分析：

- 要分别**读取排好序的关系 R 和关系 S 至内存块**，以互相**协调**的方式**分别遍历**，当遍历到的**记录完全相等时**，即 R.A=S.C 且 R.B=S.D 时，要去重输出。因此，需要实现的重点操作是：
  - **协调**好关系 R 和关系 S 记录的**遍历顺序**，在**记录完全相等**时只输出其中一条记录；
  - 当**记录不完全相等**时，正确**输出并后移关系的记录**。
- 核心代码与算法流程：
  a) 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区；
  b) 定义变量：（只列出了部分值得说明的）
     - blk：暂存**输出结果**的 1 个内存块；
     - blks：数组，存储关系 R 和关系 S **读入内存的数据块**；
     - recordCnts：数组，记录关系 R 和关系 S **当前正处理的记录号**。
  c) 先将关系 R 和关系 S 的**第 1 个数据块**读入 blks，将 **recordCnts 均置为 0**（注：读入的是已经排好序的关系 R 和关系 S）；
  d) 调用 getNewBlockInBuffer 函数为 blk 申请空闲内存块；
  e) 调用 record2XY 函数分别**读入关系 R 和关系 S 的当前记录**到变量 A、B、C、D 中；
  f) 比较 A、B、C、D 的值，按**三种不同的结果**分别执行下列三种操作：
     - **A = C 且 B = D**：调用 writeToOutBlk 函数将 A 和 B 输出至 blk 对应记录处，输出信息，并调用 shiftRecord 函数将**关系 R 和关系 S 分别后移一条记录**，如果已经读完了关系 R 或关系 S 的最后一条记录，则分别用 noMoreR 或 noMoreS 变量进行标记；

- 关系 R 还有剩余记录，且满足，**A < C，或 A = C 且 B < D**，或关系 S 已经没有剩余记录：调用 writeToOutBlk 函数将 A 和 B 输出至 blk 对应记录处，输出信息，并调用 shiftRecord 函数将**关系 R 后移一条记录**，如果已经读完了关系 R 的最后一条记录，则用 noMoreR 变量进行标记；
- 关系 S 还有剩余记录，且满足，**A > C，或 A = C 且 B > D**，或关系 R 已经没有剩余记录：调用 writeToOutBlk 函数将 C 和 D 输出至 blk 对应记录处，输出信息，并调用 shiftRecord 函数将**关系 S 后移一条记录**，如果已经读完了关系 S 的最后一条记录，则用 noMoreS 变量进行标记。

g) 重新执行步骤 e)，循环读入下一条记录，直到已经**读完了两个关系的所有记录**，才**结束**整个循环；

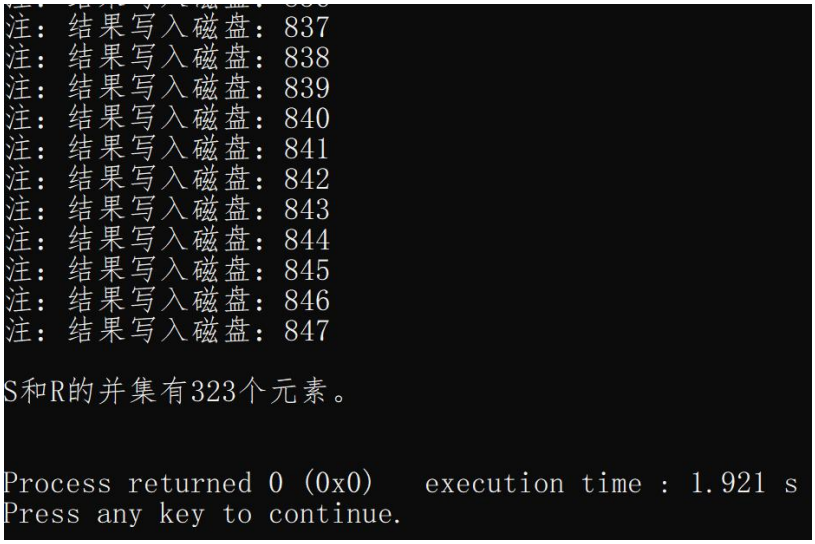
h) 输出 blk 中剩余的交记录至磁盘；

i) 输出一些结果信息，详见下方的**实验结果**。

j) 调用 freeBuffer 函数**释放内存缓冲区 buf**。

实验结果：

...



(2) 实现基于排序的两趟扫描算法，实现差集合操作算法

问题分析：

- 要分别**读取排好序的关系 R 和关系 S 至内存块**，以互相**协调**的方式**分别遍历**，当遍历到的**记录完全相等时**，即 R.A=S.C 且 R.B=S.D 时，**不能输出**，当**确定关系 S 中当前记录不会与关系 R 中未读到的记录完全相等**时，**才输出**关系 S 的记录（实现的差操作是 S - R）。因此，需要实现的重点操作是：
  - **协调**好关系 R 和关系 S 记录的**遍历顺序**，在**记录完全相等**时不输出；
  - 当**记录不完全相等**时，正确**后移关系的记录**，并在根据**排序关系**确定了**关系 S 中的记录不可能与关系 R 中未读到的记录完全相等**时，才输出关系 S 的记录。
- 核心代码与算法流程：
  k) 定义 1 个内存缓冲区 buf，调用 **initBuffer 函数初始化 buf** 为总大小 520B，块大小 64B 的内存缓冲区；
  l) 定义变量：（只列出了部分值得说明的）
    - blk：暂存**输出结果**的 1 个内存块；
    - blks：数组，存储关系 R 和关系 S **读入内存的数据块**；
    - recordCnts：数组，记录关系 R 和关系 S **当前正处理的记录号**。
  m) 先将关系 R 和关系 S 的**第 1 个数据块**读入 blks，将 **recordCnts 均置**

为 0（注：读入的是已经排好序的关系 R 和关系 S）；

n) 调用 getNewBlockInBuffer 函数为 blk 申请空闲内存块；

o) 调用 record2XY 函数分别**读入关系 R 和关系 S 的当前记录**到变量 A、B、C、D 中；

p) 比较 A、B、C、D 的值，按**三种不同的结果**分别执行下列三种操作：

- **A = C 且 B = D**：调用 shiftRecord 函数将**关系 R 和关系 S 分别后移一条记录**，如果已经读完了关系 R 或关系 S 的最后一条记录，则分别用 noMoreR 或 noMoreS 变量进行标记；
- 关系 R 还有剩余记录，且满足，**A < C，或 A = C 且 B < D**，或关系 S 已经没有剩余记录：调用 shiftRecord 函数将**关系 R 后移一条记录**，如果已经读完了关系 R 的最后一条记录，则用 noMoreR 变量进行标记；
- 关系 S 还有剩余记录，且满足，**A > C，或 A = C 且 B > D**，或关系 R 已经没有剩余记录：调用 writeToOutBlk 函数将 C 和 D 输出至 blk 对应记录处，输出信息，并调用 shiftRecord 函数将**关系 S 后移一条记录**，如果已经读完了关系 S 的最后一条记录，则用 noMoreS 变量进行标记。

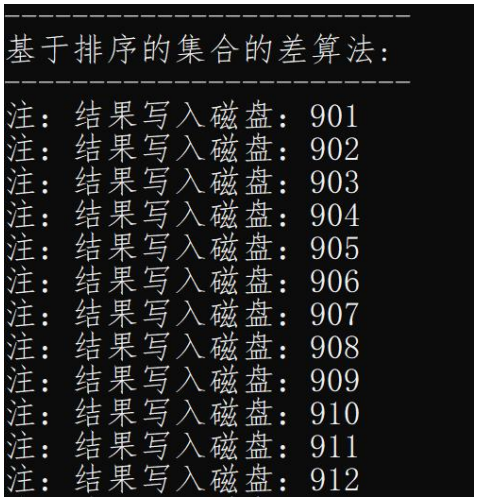
q) 重新执行步骤 e)，循环读入下一条记录，直到已经**读完了关系 S 的所有记录**，才**结束**整个循环；

r) 输出 blk 中剩余的交记录至磁盘；

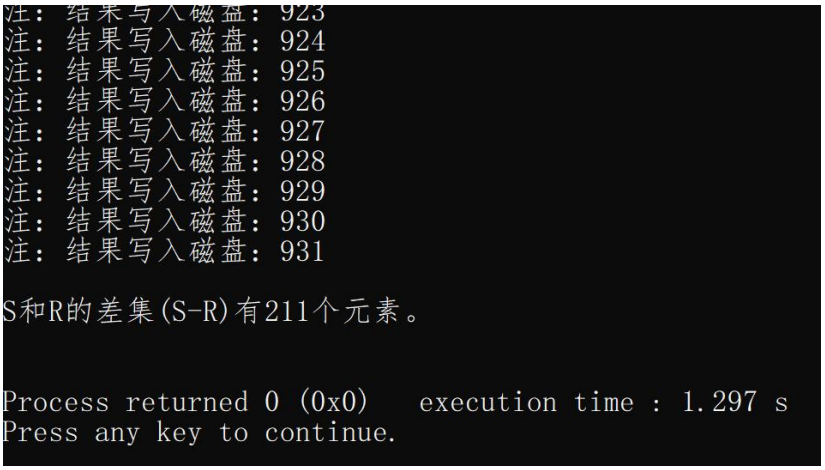
s) 输出一些结果信息，详见下方的**实验结果**。

t) 调用 freeBuffer 函数**释放内存缓冲区 buf**。

实验结果：



...



六、 总结

总结本次实验的遇到并解决的问题、收获及反思。

- 问题：
  - **边界条件**的处理问题：实现数据库查询处理算法时，在操作数据块和记录时，会经常涉及到**边界条件的处理**，例如某一数据块读到了或写到了最后，某一关系读到了最后等等，面对这些边界条件，也要以“**具体情况具体分析**”的思想去处理，可以模拟出边界条件，通过**打印调试**的办法，分析出边界条件处理的**关键点**，从而解决边界问题；

- 实现**连接操作**时**记录连续相等**的情况：在**基于排序的连接操作**中，很可能会遇到两个关系记录**可以连接**且之后的记录都**连续相等**的情况，一个好的处理办法是，先**记录下一个关系的起始记录和起始块**，**不断扫描**该关系的连续相等的记录，与分别与另一个关系的一条条连续相等记录进行连接，这样做保证不会遗漏满足连接条件的记录。

- 收获：
  - **连接操作**的高效性：真正理解了**基于排序的连接**操作的效率是高于**先笛卡尔积再筛选过滤**的，因为只需要**一次顺序遍历两个关系**，遇到记录符合连接条件时再连接即可，而不需要**两个关系嵌套循环再过滤**，这样明显可以提升效率。
  - **基于排序的集合操作**的高效性：对关系先进行排序后，再进行集合操作，可以根据**“排序关系”**这一重要条件，简化很多判断，比如在**升序**排序后进行**差操作 S - R** 时，当发现关系 S 的当前记录小于关系 R 的当前记录时，由于**关系 R 之后的记录一定是大于当前记录的**，所以关系 S 的当前记录一定不会与关系 R 中的记录相同，这时就可以直接输出，不再需要额外的判断，很高效。

- 建议：
  - 建议之后实验**改用 VSCode、Clion 等 IDE**，因为 Codeblocks 并不稳定，有**较多的 Bug**，且 **UI 界面并不美观**，这些因素虽然与实验内容无关，但它们是有可能影响同学们**实验效率**的；
  - 建议为该实验写一个**判题系统**，自动检测是否正确完成了实验任务，这样方便同学们清楚自己的**实验完成情况**，**减少盲目性**；
  - 可以再**明确一些实验要求**，例如如果想要同学们练习模拟**真实磁盘离散读写**的情况，就**明确同学们使用离散读写的方式**，只能用**后继地址**读取下一块，实验数据也离散，**杜绝连续的情况**，否则就明确告诉同学们本实验不需要模拟真实读写情况，可以连续读，或者在评分标准中写明离散与连续的评分方式，不要总是处于中间地带，不然同学们没有明确的目标，大多数都会选择方便的连续读写，起

不到模拟真实磁盘读写的实验训练效果，再比如**内存块的利用率**，本实验封装的 **api 只能一次读写一块**，实验要求又不明确是否要一次读写多块，那很多同学为了方便就只会一次读一块，在两个关系连接、集合操作时，每组只分配一个数据块，只有一个输出缓存块，就可以很方便地完成任务，而且符合 api 的调用规则，所以建议可以重新设计一下 api，能一次读写多块，并明确实验要求。