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: 熊天晨

学　号: 180110506

专　业: 计算机科学与技术

日　期: 2021-05-02

一、 实验目的

1. 理解索引的作用；
2. 掌握关系选择、投影、连接、集合的交、并、差等操作的实现算法；
3. 加深对算法 I/O 复杂性的理解。

二、 实验环境

1. Windows10 操作系统
2. CodeBlocks 20.03

三、 实验内容

关系 R 具有两个属性 A 和 B，其中 A 和 B 的属性值均为 int 型（4 个字节），A 的值域为[20, 60]，B 的值域为[1000, 2000]。

关系 S 具有两个属性 C 和 D，其中 C 和 D 的属性值均为 int 型（4 个字节）。C 的值域为[40, 80]，D 的值域为[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=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。

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。

(1) 实现基于线性搜索的关系选择算法

问题分析：首先单独在缓冲区中找一个空闲磁盘块，专用于存储满足 S.C=50 的元组；将 32 块关系 S 的磁盘块依次载入缓冲区中，每次选择该块中 S.C=50 的元组，记录在上述专用磁盘块中。

实验结果：结果记录在 1111.blk 里：

```
请输入选项：1
--------------------------------
1.基于线性搜索的选择算法 S.C=50
--------------------------------
读入数据块17
(X=50, Y=2766)
50,2766
(X=50, Y=2225)
50,2225
(X=50, Y=2741)
50,2741
读入数据块18
(X=50, Y=2537)
50,2537
读入数据块19
读入数据块20
读入数据块21
(X=50, Y=2883)
50,2883
读入数据块22
读入数据块23
(X=50, Y=1885)
50,1885
读入数据块24
(X=50, Y=1503)
50,1503
读入数据块25
读入数据块26
读入数据块27
读入数据块28
读入数据块29
读入数据块30
读入数据块31
读入数据块32
读入数据块33
读入数据块34
读入数据块35
读入数据块36
读入数据块37
读入数据块38
读入数据块39
(X=50, Y=1016)
50,1016
(X=50, Y=2913)
50,2913
读入数据块40
读入数据块41
读入数据块42
(X=50, Y=1770)
50,1770
读入数据块43
读入数据块44
(X=50, Y=1647)
50,1647
(X=50, Y=2588)
50,2588
读入数据块45
读入数据块46
读入数据块47
读入数据块48
注：结果写入磁盘：1111

满足选择条件的元组一共12个。

IO读写一共33次。
```

(2) 实现两阶段多路归并排序算法（TPMMS）

问题分析：

第一步：划分子集并对子集排序：

将 R 划分为 4 个子集合，每个子集合有 4 块，每次将一个子集合载入缓冲区，将这 28 个元组进行内排序，再存到磁盘上（保险起见，我存到了 101.blk-116.blk），循环 4 次。

类似地，将 S 划分为 4 个子集合，每个子集有 8 块，每次将一个子集合载入缓冲区，将这 56 个元组进行内排序，再存到磁盘块 117.blk-148.blk，循环 4 次。

第二步：4 个已排序子集合的数据利用内存缓冲区进行总排序：

缓冲区总共有 **8 块**，其中 **4 块（子集块）**分别用于载入从子集合中拿出的一块，这 4 块依次读取一个元组形成待比较集合，**1 块（比较块）**专用于存放待比较集合，**1 块（输出块）**用于存放待比较集合中的最小值。

整体伪代码如下：

已知：$S_{problem}$为待排序元素集合，$R_{problem}$-待排序集合中的元素个数，R_{block}-磁盘块或内存块能存储的元素个数，B_{memory}-可用内存块的个数，R(S)为求集合 S 的元素个数的函数，M_i为内存的第 i 块，P_{output}为输出块内存中当前元素的指针。

1. 将待排序集合 $S_{problem}$划分为 m 个子集合 S_1, S_2,...,S_m, 其中 $S_{problem}=\cup_{i=1,...,m}S_i$，且 $R_{problem}=\Sigma_{i=1,...,m}R(S_i)$, $R(S_i)<=B_{memory}*R_{block}$, i=1,...,m(注:每个 S_i的元素个数小于内存所能装载的元素个数).
2. for i=1 to m
3. {将 S_i装入内存，并采用一种内排序算法进行排序，排序后再存回相应的外存中}

/*步骤 2 和 3 完成子集合的排序。接下来要进行归并，M_1,...,M_m用于分别装载 S_1,...,S_m的一块*/;

4. for i=1 to m
5. { 调用 read block 函数，读 S_i的第一块存入 M_i中，同时将其第一个元素存入 $M_{compare}$的第 i_{th}个位置; }
6. 设置 P_{output}为输出内存块的起始位置;
7. 求 $M_{compare}$中 m 个元素的最小值及其位置 i。
8. If (找到最小值及其位置 i) then
9. { 将第 i_{th}个位置的元素存入 M_{output}中的 P_{output}位置，P_{output}指针按次序指向下一位置；
10. If (P_{output}指向结束位置) then
11. { 调用 Write Block 按次序将 M_{output}写回磁盘；置 P_{output}为输出内存块的起始位置；继续进行; }
12. 获取 M_i的下一个元素.
13. If (M_i有下一个元素)
14. { 将 M_i下一个元素存入 $M_{compare}$的第 i_{th}个位置。转步骤 7 继续执行。 }
15. Else { 调用 read block 按次序读 S_i的下一块并存入 M_i;
16. If (S_i有下一块)
{ 将其第一个元素存入 $M_{compare}$的第 i_{th}个位置。转步骤 7 继续执行。 }
17. ELSE { 返回一个特殊值如 Finished，以示 S_i子集合处理完毕，M_i为空，且使 $M_{compare}$中的第 i_{th}位置为该特殊值，表明该元素不参与 $M_{compare}$的比较操作。转步骤 7 继续执行。 }
18. }
19. } /*若 $M_{compare}$的所有元素都是特殊值 Finished，即没有最小值，则算法结束*/

其中编码时需要注意的是（曾遇到的 bug）：

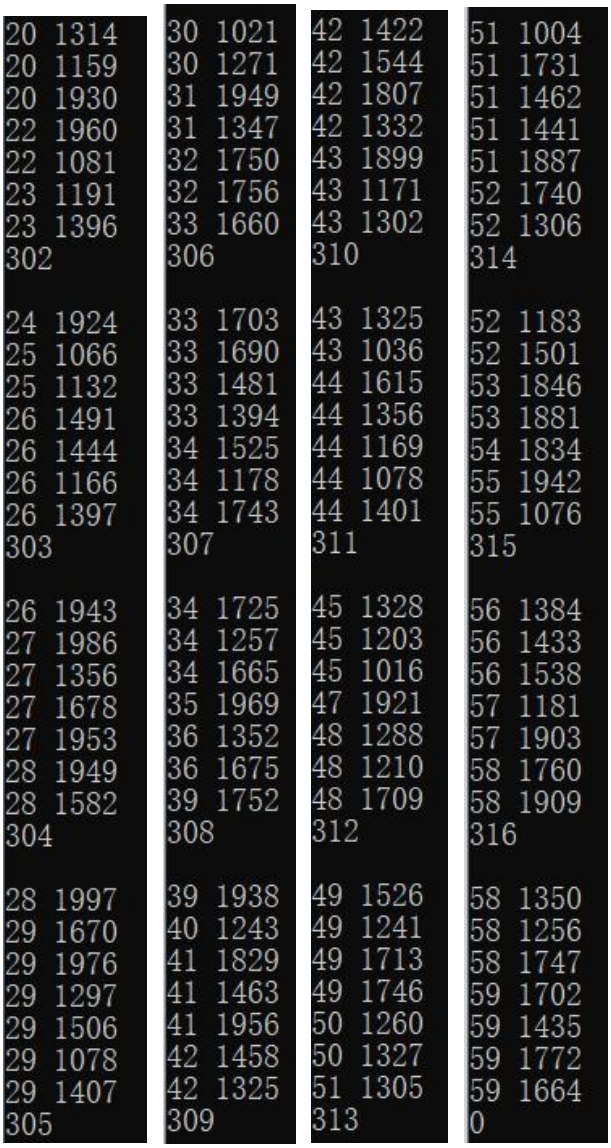
(1) 每组中依次取出一块到缓冲区中，若缓冲区里的块内元组已全输出至磁盘，则该组换下一块，直至该组没有未载入的块；

(2) 在缓冲区的 4 块子集块（**所在组必须未输出完**）分别拿出一个元组到比较块中；

(3) 在比较块中找最小 R.A/S.C 的元组写入输出块，输出块集齐 7 个元组则输出至磁盘中。

实验结果：

关系 R：按 R.A 升序排列，第一步组内排序后的中间结果存在 101.blk-116.blk 中，第二步归并排序后的结果存在 301.blk-316.blk，下面是写回到磁盘的信息的打印结果：



关系 S：按 S.C 升序排列，第一步组内排序后的中间结果存在 117.blk-148.blk 中，第二步归并排序后的结果存在 317.blk-348.blk，下面是写回到磁盘的信息的部分打印结果：

```
40 2385
40 1705
40 1243
40 1339
41 2427
41 2909
41 1829
318

41 2609
41 1269
42 2152
42 1544
42 1422
43 2908
43 2223
319

43 1236
43 1171
43 2333
43 2880
43 2857
44 1672
44 1606
320

44 1401
44 1003
45 1288
45 1203
45 2861
45 2038
45 2416
321
```

```
46 2285
46 1691
46 1664
46 2642
46 2410
46 1998
46 2564
322

46 2232
47 1429
47 1053
47 2254
47 1921
47 2410
47 1994
323

47 1429
48 2553
48 1810
48 1506
48 1509
48 2132
48 1709
324

49 2531
49 1605
49 1100
49 1084
49 2724
50 2766
50 2225
325
```

```
50 2741
50 2537
50 2883
50 1885
50 1503
50 1016
50 2913
326

50 1770
50 1647
50 2588
51 2252
51 2486
51 1501
51 1305
327

51 1887
51 2214
52 1876
52 2607
52 1134
52 1839
52 1242
328

52 1306
52 1783
53 1012
53 2809
53 1476
53 1635
53 2140
329
```

```
53 1394
54 2053
54 2366
54 1204
54 1930
54 1834
54 1906
330

55 1133
55 2886
55 2512
55 1536
56 1023
56 1544
56 1713
331

56 1796
56 1949
56 2486
56 1590
57 1312
57 1583
57 2735
332

58 1303
58 1256
58 1747
59 2154
59 2527
59 1680
59 1973
333
```

```
59 1518
59 2195
59 1990
60 2645
60 1254
60 1743
60 2524
334

60 2975
60 1083
60 1398
60 1447
60 2196
60 1920
60 1428
335

60 2716
61 1957
61 2181
61 2184
61 1853
61 1630
61 2958
336

61 2556
61 1366
62 2906
62 1775
62 1668
62 2574
62 2676
337
```

```
62 1276
62 1617
62 1564
63 2495
63 1519
63 2713
63 1983
338

63 1098
64 2552
64 1758
64 2376
64 1106
64 2478
64 1779
339

64 2053
64 1171
65 1363
65 1427
65 2469
65 1278
65 2014
340

65 2535
65 2681
66 2712
66 1057
66 1406
66 2808
67 1529
341
```

```
67 1243
67 2818
67 1765
67 1035
68 1412
68 2100
68 1962
342

68 1218
69 2441
69 2917
69 2596
69 2721
70 2216
70 2042
343

70 1852
70 1962
71 2573
71 2053
71 1509
71 1856
71 2149
344

72 1081
72 1015
72 1082
72 1325
72 1102
73 1283
73 2814
345
```

```
74 1204
74 2340
74 1203
74 1819
74 1584
74 1127
75 2961
346

75 1680
76 1384
76 1111
76 1811
76 2369
77 1626
77 2979
347
```

```
77 1218
77 1775
77 2680
78 1113
78 2666
78 1308
78 1833
348

78 1496
79 1612
79 2975
79 2576
79 2542
79 1271
79 1725
0
```

```
关系R排序后输出到文件301.blk到316.blk
关系S排序后输出到文件317.blk到348.blk

请选择以下一项：
```

说明：R/S 排序结果的最后一个磁盘块的后继地址置为 0。

(3) 实现基于索引的关系选择算法

问题分析：

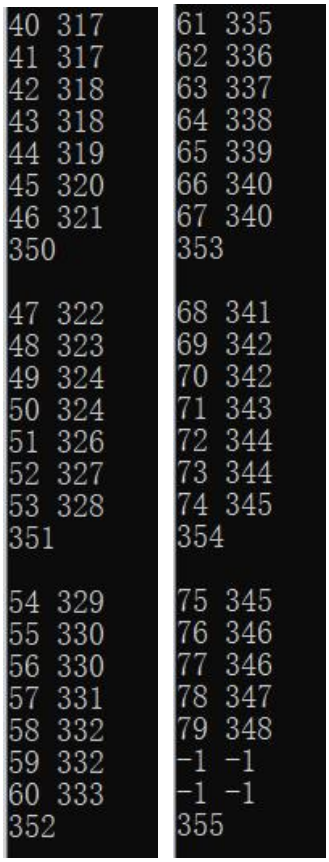
（1）基于已排序的 317.blk-348.blk，建立聚簇索引，索引字段为不同的 S.C 值，索引项的指针指向出现特定 S.C 值的**第一个块。**索引存储在 349.blk-354.blk。

（2）根据索引找到出现 S.C=50 的第一个块，再根据出现 S.C=51 的第一个块确定块的范围，读取这些块并选出其中 S.C=50 的元组输出即可。

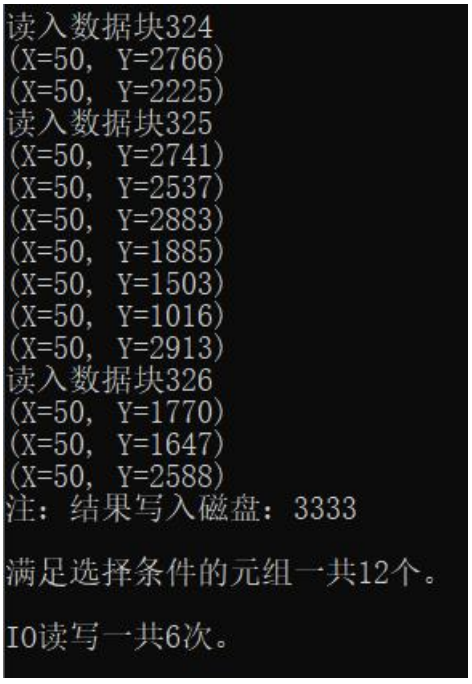
实验结果：

（1）存储在 349.blk-354.blk 的索引内容：

索引项的指针指向出现特定 S.C 值的**第一个块，**即如下图表示，S.C=40 的项第一次出现的位置在编号为 317 的 blk 里。则可以找到 S.C=50 和 S.C=51 第一次出现的位置（已经排序好了），是从 324 块到 326 块，故只需在 324~326 块之间搜索。



（2）输出连接结果与 I/O 次数，使用索引的关系选择算法仅需 6 次 I/O 操作，与第一题的 I/O 次数（33 次）相比少一些，输出结果在 3333.blk 磁盘上。



(4) 实现基于排序的连接操作算法（Sort-Merge-Join）

问题分析：

缓冲区 8 块的分配如下：

1 块专门放 R 关系的块（301.blk-316.blk 依次放入，**16 次大循环**）；

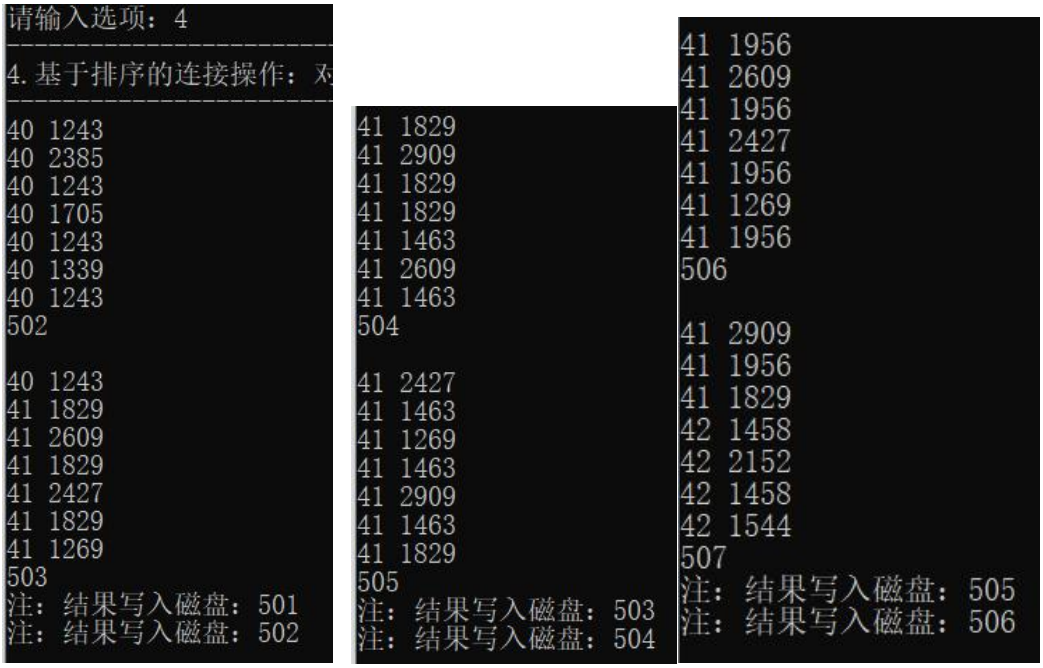
4 块专门放 4 块 S 关系的块（用 S 关系的 TPMMS 第一步的中间结果，分别从 117.blk-124.blk, 125.blk-132.blk, 133.blk-140.blk, 141.blk-148.blk 依次选择一块放入，直至**放入完**或**已判断不用读**）；

1 块 compare 专用来对从以上 5 块取出的 5 个元组进行比对连接：在比对时，若发现**读入的某元组的 S.C 已经大于 R.A，**则该元组所在的组后面都可以不读了。

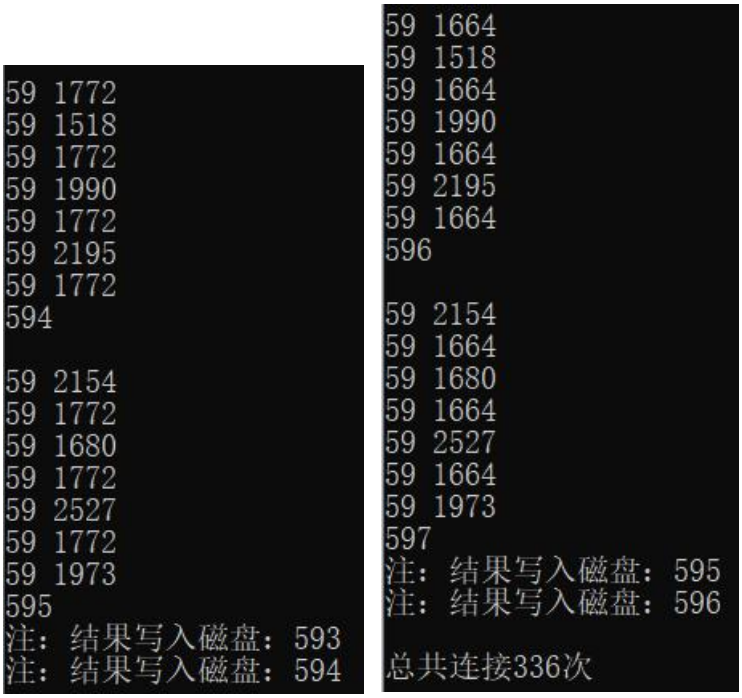
2 块 output 专用来存放**满足连接条件**的元组**对**（2 块可存放 7 个元组**对**）。

如 TPMMS 的第二步一样归并即可。

实验结果：连接结果存放在 501.blk-596.blk，下面是打印存放信息的部分截图（提交的代码中打印语句已注释掉）：



（中间省略一部分打印结果）



(5) 实现基于散列的两趟扫描算法，实现交集合操作算法

问题分析：

类似第 4 题，**缓冲区用了 7 块：**

1 块专门放 R 关系的块（301.blk-316.blk 依次放入，**16 次大循环**）；

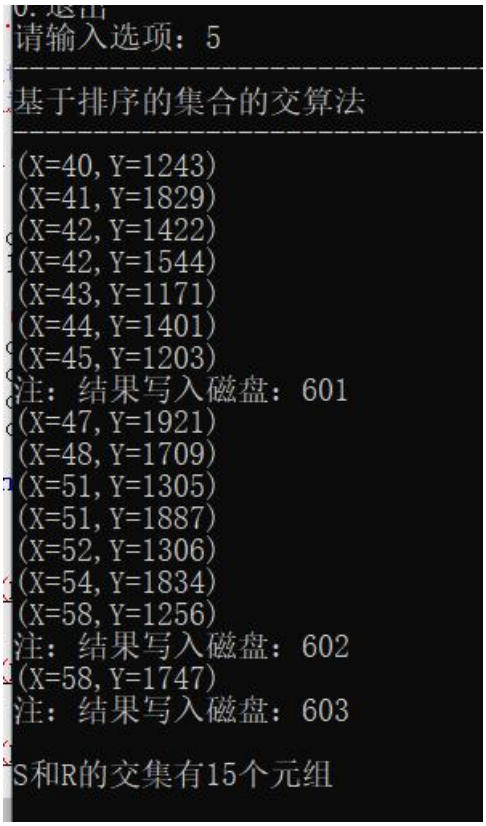
4 块专门放 4 块 S 关系的块（用 S 关系的 TPMMS 第一步的中间结果，分别

从 117.blk-124.blk, 125.blk-132.blk, 133.blk-140.blk, 141.blk-148.blk 这 4 组中依次选择一块放入，直至 4 组都**放入完或已判断不用读**）；

1 块 compare 专用来对从以上 5 块取出的 5 个元组进行比对，若 **X 和 Y 的值都相等**，则把这个元组写入 output 块中。在比对时，若发现**读入的某元组的 S.C 已经大于 R.A**，则该元组所在的组后面都可以不读了。

1 块 output 专用来存放 R 和 S 交集的元组。

实验结果：



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。

(6) 实现基于散列的两趟扫描算法，实现并集合操作算法

类似交算法，缓存区用了 7 块：

1 块专门放 R 关系的块（301.blk-316.blk 依次放入，**16 次大循环**）；

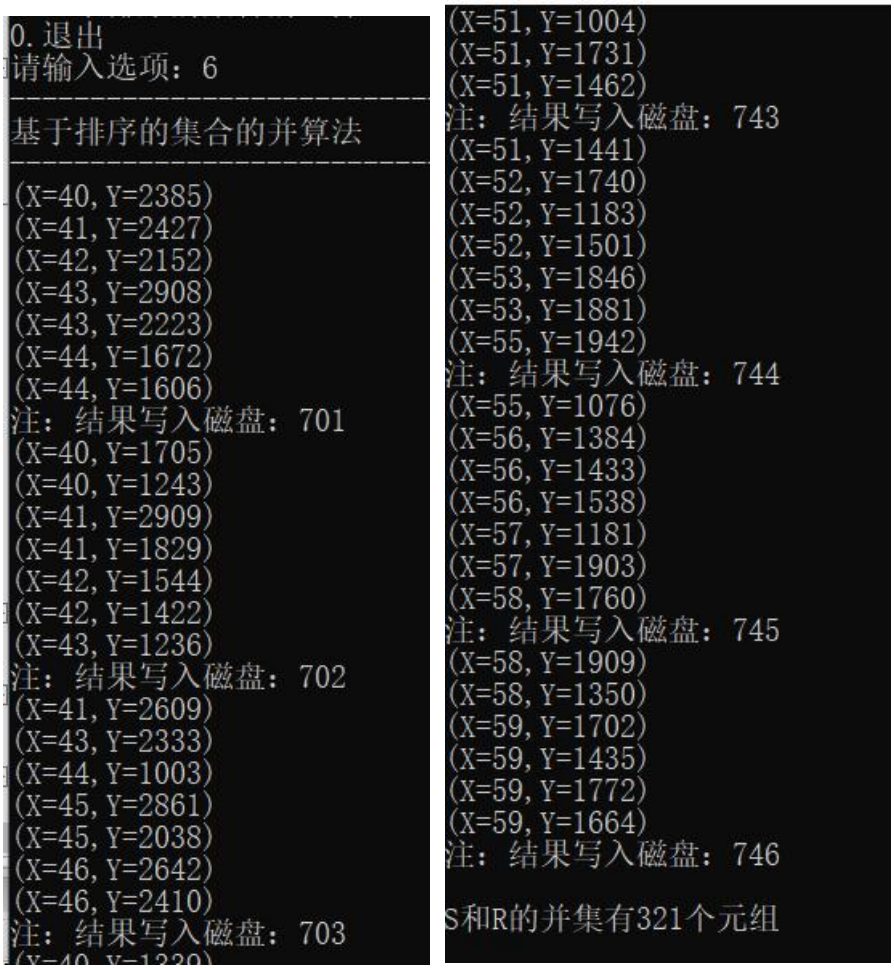
4 块专门放 4 块 S 关系的块（用 S 关系的 TPMMS 第一步的中间结果，分别从 117.blk-124.blk, 125.blk-132.blk, 133.blk-140.blk, 141.blk-148.blk 这 4 组中依次选择一块放入，直至 4 组都放入完或已判断不用读）；

1 块 compare 专用来对从以上 5 块取出的 5 个元组进行比对，若**循环完 S 的 4 个组**，都没有发现 S 元组与 compare 块里 R 元组 **X 和 Y 的值都相等**，则把这个 R 元组写入 output 块中；若发现了，就不写入。（在比对时，若发现读入的某元组的 S.C 已经大于 R.A，则该元组所在的组后面都可以不读了。）

循环结束后，再将 S 的所有元组输出，得到 S 和 R 的并集。

1 块 output 专用来存放 S 和 R 的并集。

结果存储在 701.blk-747.blk，运行结果如下（部分截图）：



(7) 实现基于散列的两趟扫描算法，实现差 R-S 集合操作算法

类似交算法，缓存区用了 7 块：

1 块专门放 R 关系的块（301.blk-316.blk 依次放入，**16 次大循环**）；

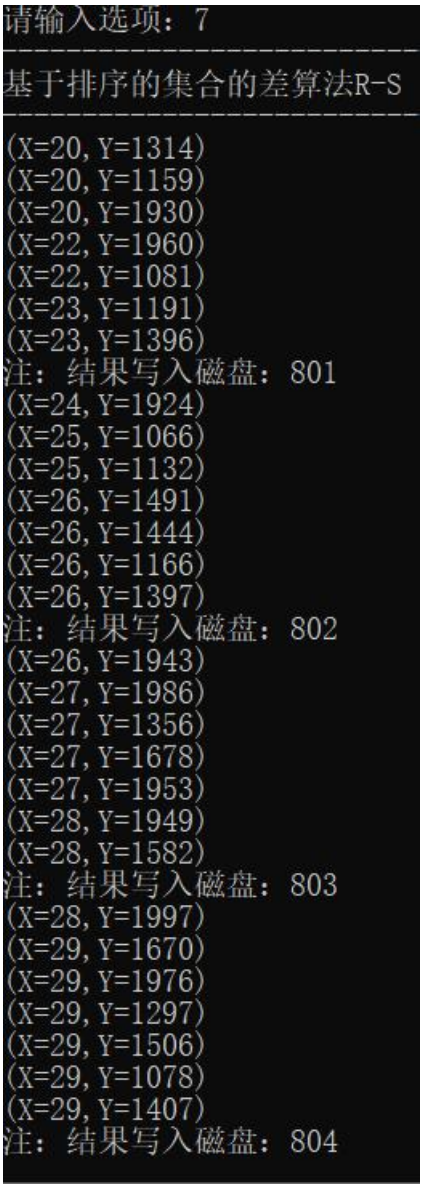
4 块专门放 4 块 S 关系的块（用 S 关系的 TPMMS 第一步的中间结果，分别从 117.blk-124.blk, 125.blk-132.blk, 133.blk-140.blk, 141.blk-148.blk 这 4 组中依次选择一块放入，直至 4 组都**放入完或已判断不用读**）；

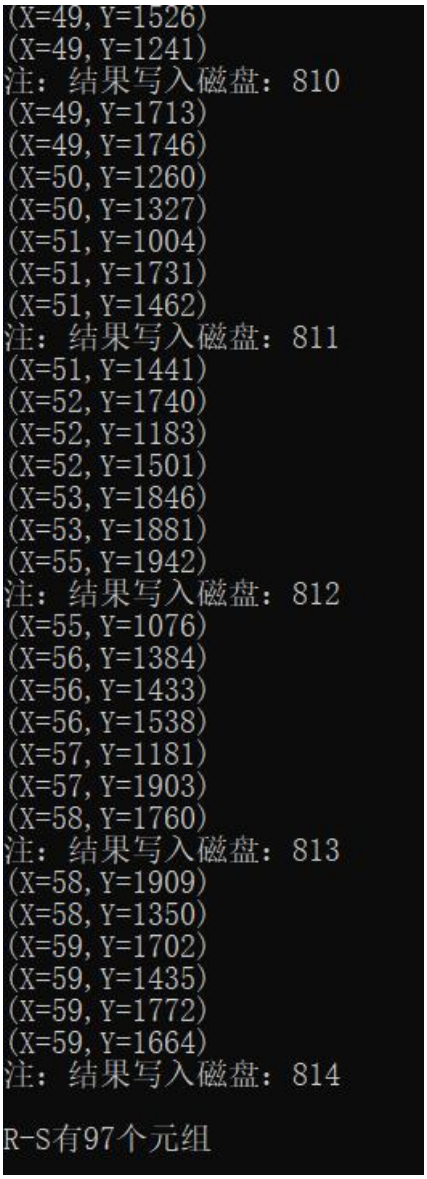
1 块 compare 专用来对从以上 5 块取出的 5 个元组进行比对，若**循环完 S 的 4 个组**，都没有发现 S 元组与 compare 块里 R 元组 **X 和 Y 的值都相等**，则把这个

R 元组写入 output 块中。在比对时，若发现**读入的某元组的 S.C 已经大于 R.A，**则该元组所在的组后面都可以不读了。

1 块 output 专用来存放在 S 中找不到的 R 元组。

结果存储在 801.blk-815.blk，运行结果如下：





六、 总结

总结本次实验的遇到并解决的问题、收获及反思。

第 3 题花了我最多时间，主要问题在：读写操作是一次一块也就是 7 个元组，但是我们建立索引块的时候，要存入的数据并不是 7 的整数。前面用 if(count==7)

判断到这一个块满了之后，就可以一次性将这一块的 7 个元组存好了，但是后面，最后一个索引块并不满足刚好 7 个能存满，所以这时候要专门加一个逻辑来判断我们需要的数据存完是在什么时候，后面的就直接存入默认值-1：

```
488                 if(count==7){//this blk is full,but the final blk isn't sure to be full
489                     endpos=i;//endpos mark the final place of the full blk
490                     index.addr = addr+1;
491
492                     for(int m=0; m<7; m++){
493                         printf("%d %d\n", index.X[m], index.Y[m]);
494                     }
495                     printf("%d\n\n", index.addr);
496
497                     write_data_blk(save, index);
498                     writeBlockToDisk(save, addr, buf);
499                     init_data_blk(&index);
500                     addr++;
501                     count = 0;
502                 }
503             }
504         }
```

```
506     for(int count=0,i=endpos+1;i<81;i++){ //上面count=7是能够写满一个块的，这里用来处理剩下的不足写满一个块的索引（不能用count=7判断的）
507         //printf("i=%d\n",i);
508         //printf("indexnum[%d]=%d,indexnum[%d]=%d\n",i,indexnum[i],i+1,indexnum[i+1]);
509         if(indexnum[i]!=999 && indexnum[i+1]!=999){
510             index.X[count] = i;
511             index.Y[count] = indexnum[i];
512             //printf("%d %d\n", index.X[count], index.Y[count]);
513             count++;
514
515             //for(int m=0; m<7; m++){
516                 //printf("%d %d\n", index.X[m], index.Y[m]);
517             //}
518             //printf("%d\n\n", index.addr);
519         }
520         if(indexnum[i]!=999 && indexnum[i+1]==999){
521             index.X[count] = i;
522             index.Y[count] = indexnum[i];
523
524             index.addr = addr+1;
525
526             for(int m=0; m<7; m++){
527                 printf("%d %d\n", index.X[m], index.Y[m]);
528             }
529             printf("%d\n\n", index.addr);
530
531             write_data_blk(save, index);
532             writeBlockToDisk(save, addr, buf);
533             init_data_blk(&index);
534             addr++;
535             //count = 0;
536         }
537     }
```

同时，写到第 3 题的时候还出现了一个问题：打印中间结果的时候出现 X=2588 的情况，调了很久，后来发现是我 data 和 out 的区域重合了，数据混乱了，后来专门开辟两个不同的区域就好了：

```
读入数据块326
(X=50, Y=1770)
(X=50, Y=1647)
(X=50, Y=2588)
(X=51, Y=2252)
(X=51, Y=2486)
(X=51, Y=1501)
(X=51, Y=1305)

(X=50, Y=1770)
**********(X=50, Y=1770)
(X=1770, Y=1647)
(X=50, Y=2588)
**********(X=2588, Y=2588)
(X=51, Y=2252)
(X=51, Y=2486)
(X=51, Y=1501)
(X=51, Y=1305)
注：结果写入磁盘：3333

满足选择条件的元组一共11个。
```

然后是第 4 题，问题跟第 3 题一样，不知道哪里内存重合了，调试了特别久还是找不出来，逻辑和结果都是对的（336），但是：在运行功能 4 之前，不能运行 3。运行 3 之后再运行 4，4 的结果就会变成不是 336，但是 3 的答案一直是对的（12）。判断是功能 4 跟功能 3 有内存上的重叠部分。

这次实验涉及了聚簇索引的建立、线性搜索算法和两阶段多路归并算法，加深了理论课的印象。