

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: 沈文心

学 号: 180110723

专 业: 计算机科学与技术

日 期: 2021-5-5

一、 实验目的

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。熟悉数据库的各个查询算法的原理，并编程模拟实现查询算法。

二、 实验环境

1. C 语言
2. Extmem 程序库
3. Windows, VSCode

三、 实验内容

借助所提供的 Extmem 库，模拟实现一下四种数据库查询算法：

1. 实现基于线性搜索的关系选择算法
2. 实现两阶段多路归并排序算法
3. 实现基于索引的关系选择算法
4. 基于排序的连接操作算法
5. 基于排序的两趟扫描算法

四、 实验过程

(1) 实现基于线性搜索的关系选择算法

问题分析：

在数据是无序的情况下，选择某个 key 值符合条件的元组，需要将磁盘

中的该表中的所有数据读入内存，依次分析其 **key** 值是否满足条件。如果符合条件，将其存入 **buffer** 的写块 **blk_w** 中，当 **blk_w** 装满时，将其写到外存中，并清空 **blk_w** 的旧数据。

实验结果：

```
-----  
基于线性搜索的选择算法  
-----
```

```
读入数据块17
```

```
(C = 50,D = 2766)
```

```
(C = 50,D = 2225)
```

```
(C = 50,D = 2741)
```

```
读入数据块18
```

```
(C = 50,D = 2537)
```

```
读入数据块19
```

```
读入数据块20
```

```
读入数据块21
```

```
(C = 50,D = 2883)
```

```
读入数据块22
```

```
读入数据块23
```

```
(C = 50,D = 1885)
```

```
读入数据块24
```

```
(C = 50,D = 1503)
```

```
结果写入磁盘 :100
```

```
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
```

```
(C = 50,D = 1016)
```

```
(C = 50,D = 2913)
```

```
读入数据块40
```

```
读入数据块41
```

```
读入数据块42
```

```
(C = 50,D = 1770)
```

```
读入数据块43
```

```
读入数据块44
```

```
(C = 50,D = 1647)
```

```
(C = 50,D = 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
```

```
结果写入磁盘 :101
```

```
满足条件的元组一共 12 个
```

```
IO's is 34
```

注：实验一的结果存在 100.blk/101.blk 中,如图所示：

```
block:100(50 2766)(50 2225)(50 2741)(50 2537)(50 2883)(50 1885)(50 1503)
block:101(50 1016)(50 2913)(50 1770)(50 1647)(50 2588)(0 0)(0 0)
```

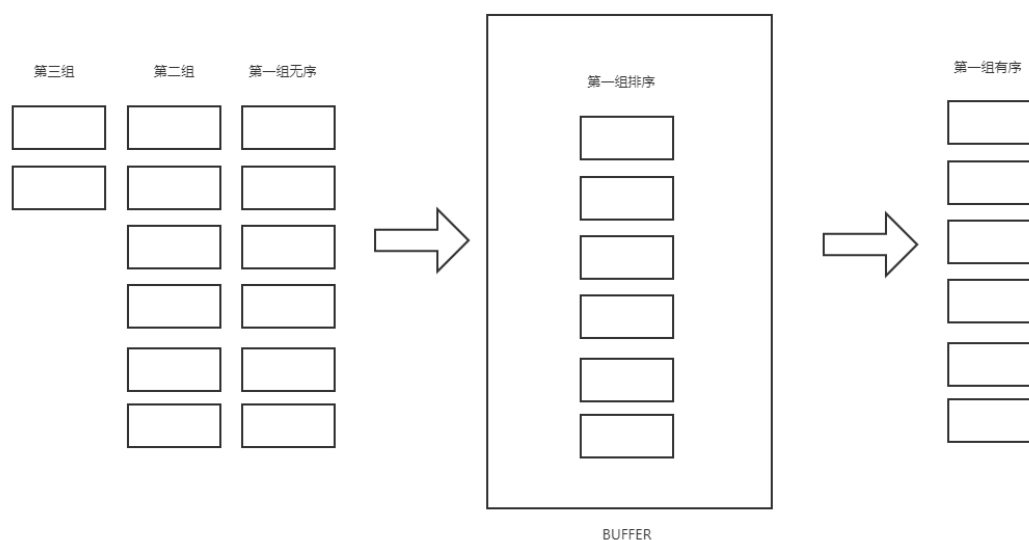
(2) 实现两阶段多路归并排序算法 (TPMMS)

问题分析:

按照算法, 我将该算法拆分为两阶段。分别是 step1, step2

```
int tpms(int start,int end){
    Buffer buf;
    if (!initBuffer(520, 64, &buf)){
        perror("Buffer Initialization Failed!\n");
        return -1;
    }
    int rid_s = TASK2_BLK_RESULT_STEP1, rid_e = TASK2_BLK_RESULT_STEP1; //step1 的结果存储的id的起始地址
    tpms_step1(start,end,rid_s,&rid_e,&buf);
    int rid_s_2 = TASK2_BLK_RESULT_STEP2, rid_e_2 = TASK2_BLK_RESULT_STEP2;
    tpms_step2(rid_s,rid_e,&buf,rid_s_2,&rid_e_2);
}
```

第一阶段: (以 R 数据库为例)

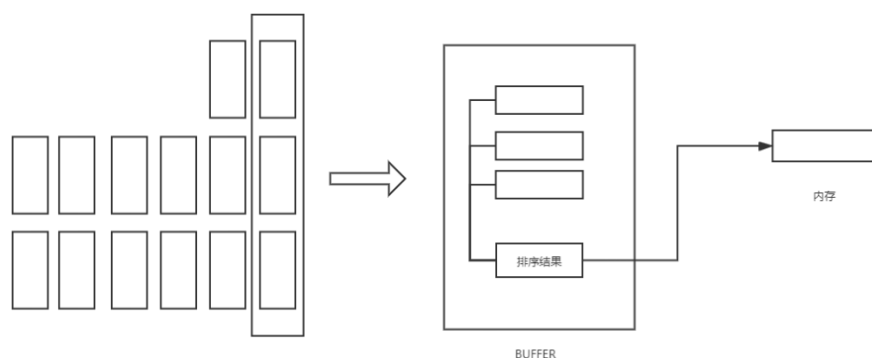


第一阶段将内存的 blk, 按照 7 个分为一组, 在 Buffer 中将其排序, 然后存入外存中。

排序方法选择的是归并排序。

先将组中的 7 个 Blk 分别排序, 然后使用归并排序。

第二阶段: (以 R 数据库为例)



将之前排好序的数据按照分组，从小到大依次多路读入，然后再 Buffer 中将其排序。直到左右数据都被排序结束。

实验结果：

对于阶段一、R 表，将阶段一的结果存在 200.blk 到 215.blk 中，共 16 个，结果如图：

```

block:200(20 1314)(20 1159)(22 1960)(22 1081)(23 1191)(24 1924)(25 1066)
block:201(26 1491)(26 1444)(27 1986)(27 1678)(27 1356)(28 1949)(29 1670)
block:202(29 1976)(29 1297)(29 1506)(30 1021)(31 1949)(34 1525)(34 1178)
block:203(35 1969)(36 1352)(40 1243)(41 1829)(42 1458)(43 1899)(43 1171)
block:204(43 1302)(44 1615)(45 1203)(45 1328)(45 1016)(48 1288)(48 1210)
block:205(49 1526)(49 1241)(49 1713)(50 1260)(50 1327)(51 1305)(52 1740)
block:206(52 1306)(55 1942)(56 1384)(57 1181)(58 1760)(59 1702)(59 1435)
block:207(20 1930)(23 1396)(25 1132)(26 1166)(26 1943)(26 1397)(27 1953)
block:208(28 1582)(28 1997)(29 1078)(30 1271)(32 1750)(33 1660)(33 1690)
block:209(33 1703)(33 1481)(34 1743)(34 1725)(36 1675)(39 1752)(39 1938)
block:210(41 1463)(42 1422)(42 1325)(42 1544)(42 1332)(42 1807)(43 1325)
block:211(43 1036)(44 1356)(44 1169)(44 1078)(47 1921)(48 1709)(51 1004)
block:212(51 1731)(51 1462)(51 1441)(52 1183)(52 1501)(53 1846)(53 1881)
block:213(54 1834)(55 1076)(56 1433)(56 1538)(58 1909)(58 1350)(59 1772)
block:214(29 1407)(31 1347)(32 1756)(33 1394)(34 1257)(34 1665)(41 1956)
block:215(44 1401)(49 1746)(51 1887)(57 1903)(58 1747)(58 1256)(59 1664)

```

对于阶段二、R 表：

其操作过程输出如图：

```

读入块:200
读入块:207
读入块:214
(20,1314)(20,1159)(20,1930)(22,1960)(22,1081)(23,1191)(23,1396) 结果写入磁盘:250
读入块:201
(24,1924)(25,1066)(25,1132)(26,1491)(26,1444)(26,1166)(26,1943) 结果写入磁盘:251
读入块:208
(26,1397)(27,1986)(27,1678)(27,1356)(27,1953)(28,1949)(28,1582) 结果写入磁盘:252
读入块:202
(28,1997)(29,1670)(29,1976)(29,1297)(29,1506)(29,1078)(29,1407) 结果写入磁盘:253
(30,1021)(30,1271)(31,1949)(31,1347)(32,1750)(32,1756)(33,1660) 结果写入磁盘:254
读入块:209
读入块:203
(33,1690)(33,1703)(33,1481)(33,1394)(34,1525)(34,1178)(34,1743) 结果写入磁盘:255
(34,1725)(34,1257)(34,1665)(35,1969)(36,1352)(36,1675)(39,1752) 结果写入磁盘:256
读入块:210
读入块:215
(39,1938)(40,1243)(41,1829)(41,1463)(41,1956)(42,1458)(42,1422) 结果写入磁盘:257
读入块:204
(42,1325)(42,1544)(42,1332)(42,1807)(43,1899)(43,1171)(43,1302) 结果写入磁盘:258
读入块:211
(43,1325)(43,1036)(44,1615)(44,1356)(44,1169)(44,1078)(44,1401) 结果写入磁盘:259
读入块:205
(45,1203)(45,1328)(45,1016)(47,1921)(48,1288)(48,1210)(48,1709) 结果写入磁盘:260
(49,1526)(49,1241)(49,1713)(49,1746)(50,1260)(50,1327)(51,1305) 结果写入磁盘:261
读入块:212
读入块:206
(51,1004)(51,1731)(51,1462)(51,1441)(51,1887)(52,1740)(52,1306) 结果写入磁盘:262
读入块:213
(52,1183)(52,1501)(53,1846)(53,1881)(54,1834)(55,1942)(55,1076) 结果写入磁盘:263
(56,1384)(56,1433)(56,1538)(57,1181)(57,1903)(58,1760)(58,1909) 结果写入磁盘:264
(58,1350)(58,1747)(58,1256)(59,1702)(59,1435)(59,1772)(59,1664) 结果写入磁盘:265

```

将结果存在 250.blk 到 265.blk 中，如图，其已经完全有序：

```

block:250(20 1314)(20 1159)(20 1930)(22 1960)(22 1081)(23 1191)(23 1396)
block:251(24 1924)(25 1066)(25 1132)(26 1491)(26 1444)(26 1166)(26 1943)
block:252(26 1397)(27 1986)(27 1678)(27 1356)(27 1953)(28 1949)(28 1582)
block:253(28 1997)(29 1670)(29 1976)(29 1297)(29 1506)(29 1078)(29 1407)
block:254(30 1021)(30 1271)(31 1949)(31 1347)(32 1750)(32 1756)(33 1660)
block:255(33 1690)(33 1703)(33 1481)(33 1394)(34 1525)(34 1178)(34 1743)
block:256(34 1725)(34 1257)(34 1665)(35 1969)(36 1352)(36 1675)(39 1752)
block:257(39 1938)(40 1243)(41 1829)(41 1463)(41 1956)(42 1458)(42 1422)
block:258(42 1325)(42 1544)(42 1332)(42 1807)(43 1899)(43 1171)(43 1302)
block:259(43 1325)(43 1036)(44 1615)(44 1356)(44 1169)(44 1078)(44 1401)
block:260(45 1203)(45 1328)(45 1016)(47 1921)(48 1288)(48 1210)(48 1709)
block:261(49 1526)(49 1241)(49 1713)(49 1746)(50 1260)(50 1327)(51 1305)
block:262(51 1004)(51 1731)(51 1462)(51 1441)(51 1887)(52 1740)(52 1306)
block:263(52 1183)(52 1501)(53 1846)(53 1881)(54 1834)(55 1942)(55 1076)
block:264(56 1384)(56 1433)(56 1538)(57 1181)(57 1903)(58 1760)(58 1909)
block:265(58 1350)(58 1747)(58 1256)(59 1702)(59 1435)(59 1772)(59 1664)

```

对于阶段一、S 表，结果存在 300.blk 到 331.blk 中，结果如图：

```
block:300(42 2152)(43 2908)(44 1672)(44 1606)(46 2285)(48 2553)(49 2531)
block:301(50 2225)(50 2766)(50 2741)(50 2537)(50 2883)(50 1885)(52 1876)
block:302(52 2607)(52 1134)(53 1012)(53 2809)(54 2053)(54 2366)(55 1133)
block:303(56 1023)(56 1544)(56 1713)(57 1312)(59 2154)(59 2527)(60 2645)
block:304(60 1254)(60 1743)(60 2524)(61 1957)(62 2906)(63 2495)(64 2552)
block:305(64 1758)(65 1427)(65 1363)(65 2469)(69 2441)(71 2573)(71 2053)
block:306(72 1081)(73 1283)(74 2340)(74 1204)(77 1626)(78 1113)(79 1612)
block:307(40 2385)(40 1705)(41 2427)(41 2909)(43 2223)(43 1236)(45 1288)
block:308(46 1691)(46 1664)(47 1111)(48 1810)(48 1506)(49 1605)(50 1503)
block:309(51 2252)(51 2486)(52 1839)(52 1306)(52 1242)(52 1783)(53 1476)
block:310(55 2886)(55 2512)(55 1536)(57 1583)(58 1303)(59 1973)(59 1680)
block:311(60 2975)(61 2181)(62 1775)(62 1668)(63 1519)(63 2713)(64 1106)
block:312(64 2376)(65 1278)(65 2014)(66 2712)(67 1529)(72 1015)(72 1082)
block:313(74 1203)(74 1819)(77 2979)(77 1218)(77 1775)(79 2975)(79 2576)
block:314(40 1243)(41 1829)(41 2609)(42 1422)(42 1544)(43 1171)(43 2333)
block:315(44 1401)(45 1203)(45 2861)(46 2642)(47 1053)(48 1509)(48 2132)
block:316(53 1635)(54 1204)(54 1930)(56 1796)(57 2735)(59 1518)(59 2195)
block:317(60 1083)(60 1447)(60 1398)(61 2184)(61 1630)(61 1853)(62 2676)
block:318(62 2574)(63 1983)(64 2478)(65 2535)(65 2681)(66 1057)(66 1406)
block:319(67 1243)(68 1412)(69 2917)(70 2216)(70 2042)(71 1509)(71 1856)
block:320(71 2149)(72 1325)(75 2961)(76 1384)(76 1111)(77 2680)(78 2666)
block:321(41 1269)(44 1003)(45 2038)(46 2410)(46 1998)(46 2564)(47 2254)
block:322(47 1921)(47 2410)(48 1709)(49 1100)(50 2913)(50 1016)(50 1770)
block:323(50 1647)(50 2588)(51 1501)(51 1887)(51 1305)(54 1834)(54 1906)
block:324(56 1949)(58 1747)(58 1256)(60 2196)(60 1920)(61 2958)(61 2556)
block:325(62 1617)(62 1276)(62 1564)(64 1779)(64 2053)(66 2808)(67 2818)
block:326(68 2100)(68 1962)(68 1218)(69 2596)(69 2721)(70 1852)(70 1962)
block:327(74 1584)(74 1127)(76 1811)(78 1308)(79 2542)(79 1725)(79 1271)
block:328(40 1339)(43 2880)(43 2857)(45 2416)(46 2232)(47 1994)(47 1429)
block:329(49 1084)(49 2724)(51 2214)(53 2140)(53 1394)(56 2486)(56 1590)
block:330(59 1990)(60 1428)(60 2716)(61 1366)(63 1098)(64 1171)(67 1765)
block:331(67 1035)(72 1102)(73 2814)(75 1680)(76 2369)(78 1833)(78 1496)
```

对于阶段二、R表，结果存在 350.blk 到 381.blk 中，结果如图：

```
block:350(40 2385)(40 1705)(40 1243)(40 1339)(41 2427)(41 2909)(41 1829)
block:351(41 2609)(41 1269)(42 2152)(42 1422)(42 1544)(43 2908)(43 2223)
block:352(43 1236)(43 1171)(43 2333)(43 2880)(43 2857)(44 1672)(44 1606)
block:353(44 1401)(44 1003)(45 1288)(45 1203)(45 2861)(45 2038)(45 2416)
block:354(46 2285)(46 1691)(46 1664)(46 2642)(46 2410)(46 1998)(46 2564)
block:355(46 2232)(47 1111)(47 1053)(47 2254)(47 1921)(47 2410)(47 1994)
block:356(47 1429)(48 2553)(48 1810)(48 1506)(48 1509)(48 2132)(48 1709)
block:357(49 2531)(49 1605)(49 1100)(49 1084)(49 2724)(50 2225)(50 2766)
block:358(50 2741)(50 2537)(50 2883)(50 1885)(50 1503)(50 2913)(50 1016)
block:359(50 1770)(50 1647)(50 2588)(51 2252)(51 2486)(51 1501)(51 1887)
block:360(51 1305)(51 2214)(52 1876)(52 2607)(52 1134)(52 1839)(52 1306)
block:361(52 1242)(52 1783)(53 1012)(53 2809)(53 1476)(53 1635)(53 2140)
block:362(53 1394)(54 2053)(54 2366)(54 1204)(54 1930)(54 1834)(54 1906)
block:363(55 1133)(55 2886)(55 2512)(55 1536)(56 1023)(56 1544)(56 1713)
block:364(56 1796)(56 1949)(56 2486)(56 1590)(57 1312)(57 1583)(57 2735)
block:365(58 1303)(58 1747)(58 1256)(59 2154)(59 2527)(59 1973)(59 1680)
block:366(59 1518)(59 2195)(59 1990)(60 2645)(60 1254)(60 1743)(60 2524)
block:367(60 2975)(60 1083)(60 1447)(60 1398)(60 2196)(60 1920)(60 1428)
block:368(60 2716)(61 1957)(61 2181)(61 2184)(61 1630)(61 1853)(61 2958)
block:369(61 2556)(61 1366)(62 2906)(62 1775)(62 1668)(62 2676)(62 2574)
block:370(62 1617)(62 1276)(62 1564)(63 2495)(63 1519)(63 2713)(63 1983)
block:371(63 1098)(64 2552)(64 1758)(64 1106)(64 2376)(64 2478)(64 1779)
block:372(64 2053)(64 1171)(65 1427)(65 1363)(65 2469)(65 1278)(65 2014)
block:373(65 2535)(65 2681)(66 2712)(66 1057)(66 1406)(66 2808)(67 1529)
block:374(67 1243)(67 2818)(67 1765)(67 1035)(68 1412)(68 2100)(68 1962)
block:375(68 1218)(69 2441)(69 2917)(69 2596)(69 2721)(70 2216)(70 2042)
block:376(70 1852)(70 1962)(71 2573)(71 2053)(71 1509)(71 1856)(71 2149)
block:377(72 1081)(72 1015)(72 1082)(72 1325)(72 1102)(73 1283)(73 2814)
block:378(74 2340)(74 1204)(74 1203)(74 1819)(74 1584)(74 1127)(75 2961)
block:379(75 1680)(76 1384)(76 1111)(76 1811)(76 2369)(77 1626)(77 2979)
block:380(77 1218)(77 1775)(77 2680)(78 1113)(78 2666)(78 1308)(78 1833)
block:381(78 1496)(79 1612)(79 2975)(79 2576)(79 2542)(79 1725)(79 1271)
```

(3) 实现基于索引的关系选择算法

问题分析:

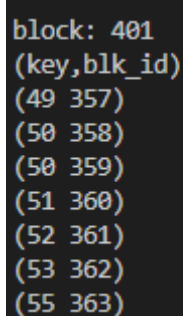
该问题也可分为两个部分。第一个部分是建立索引，第二个部分是根据索引进行选择。

第一部分：建立索引

在本次实验中，我选择的方案是对每一个磁盘块的第一个记录建立索引，而并非记录每一个记录中第一条所在的磁盘块，其目的是减少索引文件的体积，提高建立索引的效率。建立索引的思路是将每一个 blk 读出来，然后为其建立索引。这样就可以根据索引找到每一个 blk 的初始 key 值。

第二部分：根据索引选择 value 值

以 S 表为例，部分索引如图：



```
block: 401
(key,blk_id)
(49 357)
(50 358)
(50 359)
(51 360)
(52 361)
(53 362)
(55 363)
```

我根据索引选择磁盘块的思路是：首先找到第一个 key 值大于等于 mykey = 50 的索引块，然后往前回退一个索引块。

即，满足 $key \geq 50$ 的索引块为 (50, 358), 回退一个索引块就得到我所需要的索引块(49,357)，将 blk_id = 357 的 blk 读入，然后进行 key 值匹配，直到 blk 中的 key 值大于 mykey，则选择结束。

实验结果:


```

-----create index begin-----
结果写入磁盘:400
结果写入磁盘:401
结果写入磁盘:402
结果写入磁盘:403
结果写入磁盘:404
-----index created-----
-----
基于索引的选择算法排序
-----
读入索引块400
没有满足条件的元组
读入索引块401
读入数据块357
(C = 50,D = 2225)
(C = 50,D = 2766)
读入数据块358
(C = 50,D = 2741)
(C = 50,D = 2537)
(C = 50,D = 2883)
(C = 50,D = 1885)
(C = 50,D = 1503)
结果写入磁盘:450
(C = 50,D = 2913)
(C = 50,D = 1016)
读入数据块359
(C = 50,D = 1770)
(C = 50,D = 1647)
(C = 50,D = 2588)
结果写入磁盘:451
IO's is 7

```

其中索引表存在 400.blk~404.blk 中，如图所示：

```

block: 400      (40 350)(41 351)(43 352)(44 353)(46 354)(46 355)(47 356)
block: 401      (49 357)(50 358)(50 359)(51 360)(52 361)(53 362)(55 363)
block: 402      (56 364)(58 365)(59 366)(60 367)(60 368)(61 369)(62 370)
block: 403      (63 371)(64 372)(65 373)(67 374)(68 375)(70 376)(72 377)
block: 404      (74 378)(75 379)(77 380)(78 381)(0 0)(0 0)(0 0)

```

选择结果存在 450.blk~451.blk 中，如图所示：

```

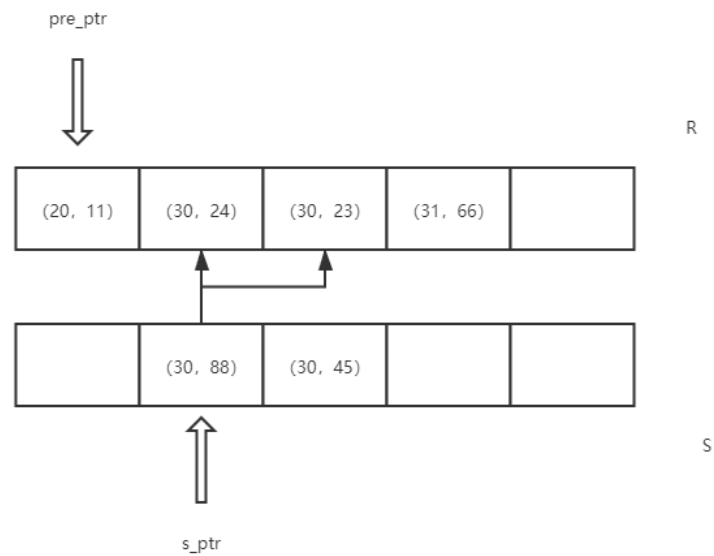
block: 450      (50 2225)(50 2766)(50 2741)(50 2537)(50 2883)(50 1885)(50 1503)
block: 451      (50 2913)(50 1016)(50 1770)(50 1647)(50 2588)(0 0)(0 0)

```

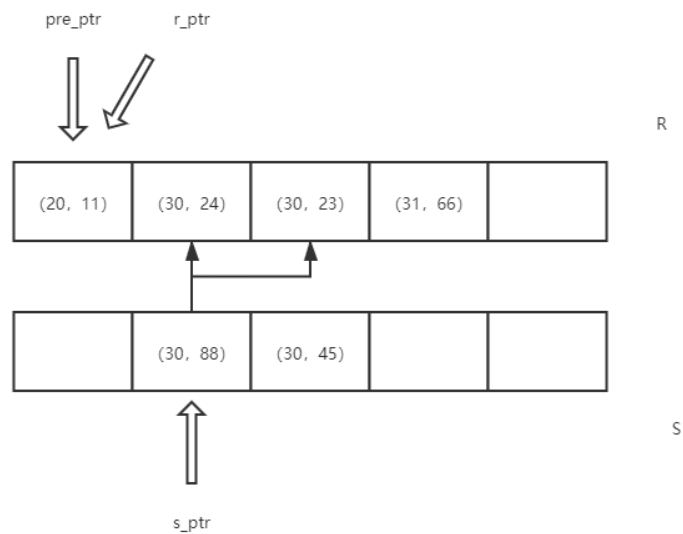
(4) 实现基于排序的连接操作算法（Sort-Merge-Join）

问题分析：

用指针来辅助其的移动。其中 `pre_ptr` 表示上次匹配成功开始的 R 表中的位置。`S_ptr` 表示 S 表中待匹配数据的位置。如图：



每次都让 r_ptr 从 pre_ptr 开始的位置往后遍历。



若 $r_ptr.X = s_ptr.X$ (即 $R.A = S.C, X$ 表示表中第一个属性):

则 join 成功,记录结果。如果 $pre_ptr.X$ 跟 $r_ptr.X$ 不同, 则更新 $pre_ptr = r_ptr$

若 $r_ptr.X < s_ptr.X$, 则 r_ptr++

若 $r_ptr.X > s_ptr.X$, 则 s_ptr++ , 此时表示 S 中开始匹配一个新的 tuple, 则重置 $r_ptr = pre_ptr$

实验结果：

```
-----
基于排序的连接算法
-----
结果写入磁盘:500
结果写入磁盘:501
结果写入磁盘:502
结果写入磁盘:503
结果写入磁盘:504
```

中间输出省略...

```
结果写入磁盘:589
结果写入磁盘:590
结果写入磁盘:591
结果写入磁盘:592
结果写入磁盘:593
结果写入磁盘:594
结果写入磁盘:595
总共链接 336 次
```

将结果存在 500.blk~595.blk 中，部分结果如图：

```
block: 500      (40 2385)(40 1243)(40 1705)(40 1243)(40 1243)(40 1243)(40 1339)
block: 501      (40 1243)(41 2427)(41 1829)(41 2427)(41 1463)(41 2427)(41 1956)
block: 502      (41 2909)(41 1829)(41 2909)(41 1463)(41 2909)(41 1956)(41 1829)
block: 503      (41 1829)(41 1829)(41 1463)(41 1829)(41 1956)(41 2609)(41 1829)
block: 504      (41 2609)(41 1463)(41 2609)(41 1956)(41 1269)(41 1829)(41 1269)
block: 505      (41 1463)(41 1269)(41 1956)(42 2152)(42 1458)(42 2152)(42 1422)
block: 506      (42 2152)(42 1325)(42 2152)(42 1544)(42 2152)(42 1332)(42 2152)

...

block: 591      (59 1772)(59 1973)(59 1664)(59 1680)(59 1702)(59 1680)(59 1435)
block: 592      (59 1680)(59 1772)(59 1680)(59 1664)(59 1518)(59 1702)(59 1518)
block: 593      (59 1435)(59 1518)(59 1772)(59 1518)(59 1664)(59 2195)(59 1702)
block: 594      (59 2195)(59 1435)(59 2195)(59 1772)(59 2195)(59 1664)(59 1990)
block: 595      (59 1702)(59 1990)(59 1435)(59 1990)(59 1772)(59 1990)(59 1664)
```

(5) 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：

选择完成交集集合操作。

思路跟 task4 很像，主要改变是之前的条件满足判断由 (r_ptr.X = s_ptr.X) 改为 (r_ptr.X = s_ptr.X && r_ptr.Y = s_ptr.Y)。

需要注意的是，当 (r_ptr.X = s_ptr.X && pre_ptr != r_ptr) 的时候 pre_ptr =

r_ptr.X 这一个指针的更新方式不变。

实验结果：

其结果存在 600.blk~602.blk 中

```
-----  
基于排序的交算法  
-----  
(X = 40,Y = 1243)  
(X = 41,Y = 1829)  
(X = 42,Y = 1422)  
(X = 42,Y = 1544)  
(X = 43,Y = 1171)  
(X = 44,Y = 1401)  
(X = 45,Y = 1203)  
结果写入磁盘:600  
(X = 47,Y = 1921)  
(X = 48,Y = 1709)  
(X = 51,Y = 1887)  
(X = 51,Y = 1305)  
(X = 52,Y = 1306)  
(X = 54,Y = 1834)  
(X = 58,Y = 1747)  
结果写入磁盘:601  
(X = 58,Y = 1256)  
结果写入磁盘:602  
S和R的交集一共有15个元组
```

五、 总结

这一次实验相较于之前的实验有挑战性了一些。其中最难调的实验是归并排序的实验。这个 task 真的很花耐心。在这一次是实验中，对索引、归并排序等都有了更深的认识，添加了索引之后 IO 次数减少，效率大大提升。本次实验花时较多的一个主要原因也是因为 C 语言的熟练度有所下降，并且 blk 文件没办法直接打开，有读出来看，给找 bug 带来了一定的困难。