实验五 查询处理算法的模拟实现

计算机科学与技术学院

200110625 柯炽炜

2023.1.7

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。

实验环境

CLion 和 CodeBlocks

实验内容

本次实验主要是通过基于ExtMem程序库，使用C语言模拟实现以下操作：

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=128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 = 128）
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R或S分别建立索引文件，利用索引文件选出S.C=128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 = 128 ）
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，实现其中一种集合操作算法：并、交、差中的一种。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

基于线性搜索的关系选择算法

- 问题分析：对当前选择的关系所存放的所有磁盘块中的内容进行遍历，找到符合条件的元组，输出并写入磁盘即可
- 源代码

  主体函数：

```
/**
 * 基于线性搜索的关系选择，
 * 缓冲区内：用6个块存放数据，2个块存放结果
 * @param num 需要选择的数值
```

```
 * @return null
 */
int linear_relationship_select(int num) {

    Buffer buf;
    // 缓冲区数据块
    unsigned char *data_blk[6];
    // 缓冲区结果块
    unsigned char *result_blk[2];
    // 结果计数
    int result_num[2] = {0};

    int answer_num = 0;
    // buffer初始化
    if (!initBuffer(520, 64, &buf)) {
        perror("Buffer Initialization Failed\n");
        return -1;
    }

    // 记录当前正在读的数据块数
    int count = 17;

    // 数据块初始化
    data_blk[0] = readBlockFromDisk(S_DISK_START_INDEX, &buf);
//    printf("读入数据块%d\n", S_DISK_START_INDEX);
    int cur_disk = S_DISK_START_INDEX;
    int next_disk = getNextDiskNum(data_blk[0]);
    for (int i = 1; i < 6; i++) {
        // 写入数据
        data_blk[i] = readBlockFromDisk(next_disk, &buf);
//        printf("读入数据块%d\n", next_disk);
        cur_disk = next_disk;
        // 计算下一磁盘块地址
        next_disk = getNextDiskNum(data_blk[i]);
    }

    // 结果块初始化
    result_blk[0] = getNewBlockInBuffer(&buf);
    result_blk[1] = getNewBlockInBuffer(&buf);

    // 完成遍历的标记变量
    int finish = 0;

    // 结果磁盘块起始下标
    int start_index = 100;

    while (count < 49) {
        for (int i = 0; i < 6; i++) {
            // 刷新result_matrix
            int result_matrix[8][2] = {0};
            printf("当前读取的数据块为%d \t", count);
            getSpecificDataFromBlk(data_blk[i], result_matrix, num);
            count += 1;
            if (result_num[0] < 7) {
                // 数据写回
```

```
                writeResultToBlock(result_matrix, &result_num[0],
result_blk[0], start_index + 1);
            } else if (result_num[1] < 7) {
                // 数据写回
                writeResultToBlock(result_matrix, &result_num[1],
result_blk[1], start_index + 2);
            }
                // 结果块都已写满
            else {
                writeBlockToDisk(result_blk[0], start_index, &buf);
                writeBlockToDisk(result_blk[1], start_index + 1, &buf);
                start_index += 2;
            }
            // 遍历结束
            if (count == 49) {
                finish = 1;
                break;
            }
        }
        if (finish) {
            // 写入数据
            writeBlockToDisk(result_blk[0], start_index, &buf);
            writeBlockToDisk(result_blk[1], start_index + 1, &buf);
            printf("将选择结果写入磁盘块 %d 与 %d\n", start_index, start_index +
1);
            break;
        }
        updateDataBlks(data_blk, &buf, 6);
    }

    // 输出IO读取次数
    printf("IO num: %d\n", buf.numIO);

    return 0;
}
```

工具函数:

```
/**
 * 找到磁盘的后继数据块地址
 * @param blk 当前磁盘块
 * @return 后续磁盘块地址
 */
int getNextDiskNum(unsigned char *blk) {
    // 七个元组
    int tuple_num = 7;

    char str[5];
    for (int i = 0; i < 4; i++) {
        str[i] = *(blk + tuple_num * 8 + i);
    }
    int addr = atoi(str);
//    printf("next address: %d\n", addr);
    return addr;
}
```

```
/**
 * 获取blk中等于num的数据
 * @param blk 磁盘块
 * @param num 特定值
 * @param result_matrix 结果数组
 */
void getSpecificDataFromBlk(unsigned char *blk, int (*result_matrix)[2], int
num) {
    // 从blk读取出的数据
    char str_1[4];
    char str_2[4];
    // 输出的结果
    char str[8] = {"-1"};
    // 字符型转整形数据
    int x = -1, y = -1;

    // 结果数组
    int result_num = 0;

    // BLK_LENGTH - 1 是因为需要存放一个后继磁盘块地址
    for (int i = 0; i < BLK_LENGTH - 1; i++) {
        for (int j = 0; j < DATA_LENGTH; j++) {
            str_1[j] = *(blk + i * 8 + j);
        }
        x = atoi(str_1);
        for (int j = 0; j < DATA_LENGTH; j++) {
            str_2[j] = *(blk + i * 8 + DATA_LENGTH + j);
        }
        y = atoi(str_2);

        // 写入结果
        if (x == 128) {
//            printf("找到结果(%d, %d)\n", x, y);
            result_matrix[result_num][0] = x;
            result_matrix[result_num][1] = y;
            result_num += 1;
        }
    }

    // 没有该块符合条件的数据
    if (result_num == 0) {
        printf("当前块没有符合条件的数据\n");
    } else {
        printf("当前块找到的数据有: ");
        for (int i = 0; i < result_num; i++) {
            printf("(%d, %d)\t", result_matrix[i][0], result_matrix[i][1]);
        }
        printf("\n");
    }
}
```

- 实验结果：（console窗口输出的结果）

```
```

```
**********************************************************************************
************

任务一:
当前读取的数据块为17      当前块没有符合条件的数据
当前读取的数据块为18      当前块没有符合条件的数据
当前读取的数据块为19      当前块没有符合条件的数据
当前读取的数据块为20      当前块没有符合条件的数据
当前读取的数据块为21      当前块没有符合条件的数据
当前读取的数据块为22      当前块没有符合条件的数据
当前读取的数据块为23      当前块找到的数据有: (128, 684)   (128, 431)
当前读取的数据块为24      当前块没有符合条件的数据
当前读取的数据块为25      当前块找到的数据有: (128, 615)   (128, 429)
当前读取的数据块为26      当前块没有符合条件的数据
当前读取的数据块为27      当前块没有符合条件的数据
当前读取的数据块为28      当前块没有符合条件的数据
当前读取的数据块为29      当前块没有符合条件的数据
当前读取的数据块为30      当前块找到的数据有: (128, 584)
当前读取的数据块为31      当前块没有符合条件的数据
当前读取的数据块为32      当前块没有符合条件的数据
当前读取的数据块为33      当前块没有符合条件的数据
当前读取的数据块为34      当前块找到的数据有: (128, 592)   (128, 457)
当前读取的数据块为35      当前块找到的数据有: (128, 720)
当前读取的数据块为36      当前块没有符合条件的数据
当前读取的数据块为37      当前块没有符合条件的数据
当前读取的数据块为38      当前块没有符合条件的数据
当前读取的数据块为39      当前块没有符合条件的数据
当前读取的数据块为40      当前块没有符合条件的数据
当前读取的数据块为41      当前块没有符合条件的数据
当前读取的数据块为42      当前块没有符合条件的数据
当前读取的数据块为43      当前块没有符合条件的数据
当前读取的数据块为44      当前块没有符合条件的数据
当前读取的数据块为45      当前块找到的数据有: (128, 447)
当前读取的数据块为46      当前块找到的数据有: (128, 871)
当前读取的数据块为47      当前块没有符合条件的数据
当前读取的数据块为48      当前块没有符合条件的数据
将选择结果写入磁盘块 100 与 101
IO num: 34
**********************************************************************************
************
```

实现两阶段多路归并算法:

- 问题分析:
  - 首先实现内排序，缓冲区的大小允许可以完成8块数据块的单次内排序，只需要对当前关系所占取的内存块进行分组并依次实现内排序即可
  - 再进行外排序
    - 对每个分组块分配一个指针:

```
// 声明指向每个组的指针
unsigned char *group[group_num];
memset(group, 0, sizeof(group));
```

- 使用数组记录每个分组当前读取下标

```
// 记录当前组读取到的位置
int cur_group_index[group_num];
memset(cur_group_index, 0, sizeof(cur_group_index));
```

- 再根据当前对应的下标依次比较并选出较小的值写入结果块

```
// 计算当前指针所指的元组值
for (int j = 0; j < 4; j++) {
    str_1[j] = *(group[i] + offset * 8 + j);
    str_2[j] = *(group[i] + offset * 8 + 4 + j);
}
tuple[i][0] = atoi(str_1);
tuple[i][1] = atoi(str_2);
// 找到最小的值
if (tuple[i][compare_col] <= min_tuple[compare_col]) {
    min_tuple[0] = tuple[i][0];
    min_tuple[1] = tuple[i][1];
    // 更新所选择的组号
    select_group = i;
}
```

- 源代码

```
/**
 * 创建 end - start个指针分别指向每一个子表
 * 因为每个子表已经排好序，所以找到最小值的那个子表移入结果块然后进行排序即可
 * @param start
 * @param end
 * @param compare_col 比较的属性
 */
void TPMMS(int start, int end, int compare_col) {
    // 声明缓冲区并初始化
    Buffer buf;
    initBuffer(520, 64, &buf);
    // 对原始数据进行内排序
    sort_blk_data(start, end, 0, 200);
    // 组数
    int group_num = (end + 1 - start) / 8;

    // 声明指向每个组的指针
    unsigned char *group[group_num];
    memset(group, 0, sizeof(group));

    // 声明结果块
    int result_num[8 - group_num];
    memset(result_num, 0, sizeof(result_num));
    unsigned char *result_blk[8 - group_num];
    for (int i = 0; i < (8 - group_num); i++) {
        result_blk[i] = getNewBlockInBuffer(&buf);
    }

    // 声明写回磁盘块的下标
```

```
    int write_disk_index = 300 + start;

    // 记录当前组读取到的位置
    int cur_group_index[group_num];
    memset(cur_group_index, 0, sizeof(cur_group_index));

    // 所有组都已经完成的标记变量
    int all_finish = 0;

    // 每一个组单独的标记变量
    int finish[group_num];
    memset(finish, 0, sizeof(finish));

    int tuple[group_num][2];
    memset(tuple, 0, sizeof(tuple));

    while (1) {
        char str_1[4];
        char str_2[4];
        // 判断所有组是否已经完全遍历
        int finish_num = 0;
        while (finish_num < group_num && (finish[finish_num] == 1)) {
            finish_num++;
        }
        if (finish_num == group_num) {
            break;
        }
        // 当前循环所选择的组号
        int select_group = -1;
        // 选择最小的元组
        int min_tuple[2] = {1000, 1000};
        for (int i = 0; i < group_num; i++) {
            if (cur_group_index[i] == 63) {
                finish[i] = 1;
                continue;
            }
            // 初始化
            if (group[i] == NULL) {
                int cur_blk_num = 200 + start + i * 8 + cur_group_index[i] /
8;
                group[i] = readBlockFromDisk(cur_blk_num, &buf);
            }
            // 当前下标对应的块内偏移量
            int offset = cur_group_index[i] % 8;
            if (offset == 7) {
                // 计算下一磁盘块地址
                for (int j = 0; j < 4; j++) {
                    str_1[j] = *(group[i] + offset * 8 + j);
                }
                // 释放原有磁盘块
                freeBlockInBuffer(group[i], &buf);
                // 读入新的磁盘块
                int next_group = 200 + atoi(str_1);
                group[i] = readBlockFromDisk(next_group, &buf);
                cur_group_index[i] += 1;
```

```
                offset = cur_group_index[i] % 8;
            }
            // 计算当前指针所指的元组值
            for (int j = 0; j < 4; j++) {
                str_1[j] = *(group[i] + offset * 8 + j);
                str_2[j] = *(group[i] + offset * 8 + 4 + j);
            }
            tuple[i][0] = atoi(str_1);
            tuple[i][1] = atoi(str_2);
            // 找到最小的值
            if (tuple[i][compare_col] <= min_tuple[compare_col]) {
                min_tuple[0] = tuple[i][0];
                min_tuple[1] = tuple[i][1];
                // 更新所选择的组号
                select_group = i;
            }
        }
        // 所有的组都已经判断完毕
        if (select_group == -1) {
            // 所有结果块写回磁盘块
            for (int i = 0; i < 8 - group_num; i++) {
                char str_addr[4];
                // 所有磁盘块已经写完
                if (write_disk_index > (300 + end)) {
                    break;
                }
                // 写入下一个磁盘块地址
                itoa(write_disk_index + 1, str_addr, 10);
                for (int j = 0; j < 4; j++) {
                    *(result_blk[i] + result_num[i] * 8 + j) = str_addr[j];
                }
                writeBlockToDisk(result_blk[i], write_disk_index, &buf);
                printf("结果块%d 写入磁盘下标 %d\n", i, write_disk_index);
                write_disk_index += 1;
            }
        } else {
            // 写入结果块
            itoa(min_tuple[0], str_1, 10);
            itoa(min_tuple[1], str_2, 10);
            // 写入所有磁盘块的数据
            for (int i = 0; i < 8 - group_num; i++) {
                if (result_num[i] < 7) {
                    for (int j = 0; j < 4; j++) {
                        *(result_blk[i] + result_num[i] * 8 + j) = str_1[j];
                        *(result_blk[i] + result_num[i] * 8 + 4 + j) =
str_2[j];
                    }
//                    printf("(%d, %d)\t", atoi(str_1), atoi(str_2));
//                    printf("写入结果块%d\t", i);
                    result_num[i] += 1;
                    break;
                }
            }
            // 更新对应的下标
            cur_group_index[select_group] += 1;
```

```
//            printf("group %d cur_index: %d\n", select_group,
cur_group_index[select_group]);
            // 所有结果块写满写回磁盘块
            if (result_num[8 - group_num - 1] == 7) {
                for (int i = 0; i < 8 - group_num; i++) {
                    // 写入下一磁盘块地址
                    char str_addr[4];
                    itoa(write_disk_index + 1, str_addr, 10);
//                    printf("结果块%d 写入地址%d\n", i, write_disk_index + 1);
                    for (int j = 0; j < 4; j++) {
                        *(result_blk[i] + 7 * 8 + j) = str_addr[j];
                    }
                    // 写回磁盘
                    writeBlockToDisk(result_blk[i], write_disk_index, &buf);
                    printf("结果块%d 写入磁盘下标 %d\n", i, write_disk_index);
                    write_disk_index += 1;
                    // 申请新的磁盘块
                    result_blk[i] = getNewBlockInBuffer(&buf);
                    result_num[i] = 0;
                }
            }
        }
    }
}
```

- 运行结果：

```
任务二：
对关系R进行TPMMS内排序
********************************************************************************
************
第 1 轮内排序：
原始数据为：
(131, 479)  (123, 477)  (107, 411)  (139, 476)  (106, 461)  (119, 406)
(105, 497)
(137, 473)  (137, 451)  (134, 459)  (138, 411)  (100, 421)  (109, 472)
(135, 441)
(133, 455)  (130, 436)  (121, 438)  (124, 468)  (117, 438)  (123, 422)
(131, 454)
(109, 404)  (139, 461)  (116, 421)  (122, 474)  (138, 497)  (114, 410)
(118, 414)
(102, 465)  (117, 475)  (137, 420)  (108, 436)  (120, 418)  (101, 410)
(101, 406)
(105, 428)  (109, 409)  (101, 470)  (124, 412)  (104, 450)  (119, 428)
(107, 477)
(115, 401)  (129, 455)  (133, 467)  (105, 476)  (137, 428)  (114, 425)
(102, 413)
(120, 827)  (122, 546)  (125, 886)  (127, 767)  (128, 447)  (130, 656)
(140, 610)
经过排序的数据为：
(100, 421)  (101, 410)  (101, 406)  (101, 470)  (102, 465)  (102, 413)
(104, 450)
(105, 497)  (105, 428)  (105, 476)  (106, 461)  (107, 411)  (107, 477)
(108, 436)
```

```
(109, 472)  (109, 404)  (109, 409)  (114, 410)  (114, 425)  (115, 401)
(116, 421)
(117, 438)  (117, 475)  (118, 414)  (119, 406)  (119, 428)  (120, 418)
(120, 827)
(121, 438)  (122, 474)  (122, 546)  (123, 477)  (123, 422)  (124, 468)
(124, 412)
(125, 886)  (127, 767)  (128, 447)  (129, 455)  (130, 436)  (130, 656)
(131, 479)
(131, 454)  (133, 455)  (133, 467)  (134, 459)  (135, 441)  (137, 473)
(137, 451)
(137, 420)  (137, 428)  (138, 411)  (138, 497)  (139, 476)  (139, 461)
(140, 610)
写入数据块201
写入数据块202
写入数据块203
写入数据块204
写入数据块205
写入数据块206
写入数据块207
写入数据块208
**********************************************************************
**********
第 2 轮内排序:
原始数据为:
(137, 471)  (120, 499)  (133, 441)  (126, 423)  (129, 488)  (116, 470)
(124, 410)
(100, 439)  (101, 405)  (138, 468)  (123, 452)  (117, 442)  (104, 440)
(133, 428)
(132, 483)  (116, 452)  (110, 405)  (138, 475)  (128, 453)  (124, 426)
(116, 414)
(102, 492)  (129, 475)  (129, 402)  (131, 492)  (101, 409)  (114, 414)
(107, 434)
(121, 464)  (116, 424)  (130, 411)  (110, 413)  (128, 459)  (129, 430)
(117, 403)
(100, 400)  (117, 426)  (110, 491)  (138, 495)  (138, 407)  (130, 495)
(137, 480)
(126, 485)  (134, 486)  (108, 482)  (119, 431)  (111, 445)  (112, 467)
(117, 412)
(124, 499)  (130, 417)  (118, 478)  (116, 420)  (132, 422)  (110, 450)
(134, 437)
经过排序的数据为:
(100, 439)  (100, 400)  (101, 405)  (101, 409)  (102, 492)  (104, 440)
(107, 434)
(108, 482)  (110, 405)  (110, 413)  (110, 491)  (110, 450)  (111, 445)
(112, 467)
(114, 414)  (116, 470)  (116, 452)  (116, 414)  (116, 424)  (116, 420)
(117, 442)
(117, 403)  (117, 426)  (117, 412)  (118, 478)  (119, 431)  (120, 499)
(121, 464)
(123, 452)  (124, 410)  (124, 426)  (124, 499)  (126, 423)  (126, 485)
(128, 453)
(128, 459)  (129, 488)  (129, 475)  (129, 402)  (129, 430)  (130, 411)
(130, 495)
(130, 417)  (131, 492)  (132, 483)  (132, 422)  (133, 441)  (133, 428)
(134, 486)
```

```
(134, 437)  (137, 471)  (137, 480)  (138, 468)  (138, 475)  (138, 495)
(138, 407)
写入数据块209
写入数据块210
写入数据块211
写入数据块212
写入数据块213
写入数据块214
写入数据块215
写入数据块216
结果块0 写入磁盘下标 301
结果块1 写入磁盘下标 302
结果块2 写入磁盘下标 303
结果块3 写入磁盘下标 304
结果块4 写入磁盘下标 305
结果块5 写入磁盘下标 306
结果块0 写入磁盘下标 307
结果块1 写入磁盘下标 308
结果块2 写入磁盘下标 309
结果块3 写入磁盘下标 310
结果块4 写入磁盘下标 311
结果块5 写入磁盘下标 312
结果块0 写入磁盘下标 313
结果块1 写入磁盘下标 314
结果块2 写入磁盘下标 315
结果块3 写入磁盘下标 316
对关系R进行TPMMS内排序
********************************************************************************
************
第 1 轮内排序:
原始数据为:
(124, 605)  (146, 445)  (123, 532)  (124, 844)  (141, 754)  (135, 919)
(136, 454)
(123, 477)  (129, 486)  (130, 436)  (134, 437)  (139, 461)  (129, 430)
(120, 418)
(121, 739)  (130, 656)  (155, 544)  (127, 680)  (145, 805)  (123, 794)
(156, 600)
(151, 704)  (139, 878)  (133, 456)  (129, 591)  (124, 424)  (132, 640)
(154, 728)
(159, 630)  (137, 733)  (139, 576)  (141, 732)  (154, 810)  (148, 570)
(140, 610)
(135, 671)  (125, 587)  (137, 912)  (134, 457)  (125, 886)  (146, 721)
(157, 456)
(149, 873)  (128, 684)  (152, 717)  (155, 600)  (128, 431)  (148, 854)
(120, 736)
(130, 790)  (156, 809)  (129, 492)  (120, 581)  (125, 725)  (159, 447)
(157, 902)
经过排序的数据为:
(120, 418)  (120, 736)  (120, 581)  (121, 739)  (123, 532)  (123, 477)
(123, 794)
(124, 605)  (124, 844)  (124, 424)  (125, 587)  (125, 886)  (125, 725)
(127, 680)
(128, 684)  (128, 431)  (129, 486)  (129, 430)  (129, 591)  (129, 492)
(130, 436)
```

(130, 656) (130, 790) (132, 640) (133, 456) (134, 437) (134, 457)
(135, 919)
(135, 671) (136, 454) (137, 733) (137, 912) (139, 461) (139, 878)
(139, 576)
(140, 610) (141, 754) (141, 732) (145, 805) (146, 445) (146, 721)
(148, 570)
(148, 854) (149, 873) (151, 704) (152, 717) (154, 728) (154, 810)
(155, 544)
(155, 600) (156, 600) (156, 809) (157, 456) (157, 902) (159, 630)
(159, 447)
写入数据块217
写入数据块218
写入数据块219
写入数据块220
写入数据块221
写入数据块222
写入数据块223
写入数据块224
****
****
第 2 轮内排序:
原始数据为:
(152, 634) (122, 554) (134, 494) (128, 615) (120, 775) (148, 616)
(128, 429)
(142, 718) (135, 542) (121, 793) (137, 460) (137, 895) (141, 636)
(148, 635)
(120, 571) (148, 547) (146, 843) (156, 479) (137, 585) (145, 625)
(157, 498)
(123, 733) (156, 698) (156, 482) (158, 764) (144, 609) (147, 663)
(142, 450)
(156, 857) (142, 748) (152, 826) (147, 561) (136, 711) (139, 802)
(125, 496)
(140, 551) (136, 434) (153, 642) (146, 834) (158, 432) (141, 629)
(128, 584)
(139, 488) (145, 689) (141, 504) (122, 682) (151, 444) (129, 844)
(122, 760)
(150, 835) (138, 512) (120, 852) (145, 521) (134, 883) (155, 668)
(139, 815)
经过排序的数据为:
(120, 775) (120, 571) (120, 852) (121, 793) (122, 554) (122, 682)
(122, 760)
(123, 733) (125, 496) (128, 615) (128, 429) (128, 584) (129, 844)
(134, 494)
(134, 883) (135, 542) (136, 711) (136, 434) (137, 460) (137, 895)
(137, 585)
(138, 512) (139, 802) (139, 488) (139, 815) (140, 551) (141, 636)
(141, 629)
(141, 504) (142, 718) (142, 450) (142, 748) (144, 609) (145, 625)
(145, 689)
(145, 521) (146, 843) (146, 834) (147, 663) (147, 561) (148, 616)
(148, 635)
(148, 547) (150, 835) (151, 444) (152, 634) (152, 826) (153, 642)
(155, 668)
(156, 479) (156, 698) (156, 482) (156, 857) (157, 498) (158, 764)
(158, 432)

```
写入数据块225
写入数据块226
写入数据块227
写入数据块228
写入数据块229
写入数据块230
写入数据块231
写入数据块232
**********************************************************************************
***********
第 3 轮内排序:
原始数据为:
(140, 915)  (145, 570)  (159, 737)  (129, 535)  (153, 456)  (121, 596)
(125, 625)
(151, 457)  (128, 592)  (126, 774)  (128, 457)  (126, 685)  (121, 582)
(145, 540)
(152, 573)  (131, 808)  (147, 552)  (130, 766)  (128, 720)  (151, 644)
(142, 679)
(146, 463)  (153, 573)  (157, 691)  (138, 700)  (130, 556)  (150, 890)
(134, 900)
(134, 756)  (130, 474)  (159, 585)  (120, 781)  (154, 593)  (155, 916)
(135, 747)
(154, 528)  (149, 523)  (158, 602)  (149, 763)  (122, 463)  (137, 660)
(155, 464)
(142, 841)  (127, 743)  (145, 840)  (131, 892)  (138, 816)  (143, 732)
(138, 674)
(125, 808)  (138, 598)  (123, 468)  (151, 461)  (149, 573)  (153, 641)
(126, 919)
经过排序的数据为:
(120, 781)  (121, 596)  (121, 582)  (122, 463)  (123, 468)  (125, 625)
(125, 808)
(126, 774)  (126, 685)  (126, 919)  (127, 743)  (128, 592)  (128, 457)
(128, 720)
(129, 535)  (130, 766)  (130, 556)  (130, 474)  (131, 808)  (131, 892)
(134, 900)
(134, 756)  (135, 747)  (137, 660)  (138, 700)  (138, 816)  (138, 674)
(138, 598)
(140, 915)  (142, 679)  (142, 841)  (143, 732)  (145, 570)  (145, 540)
(145, 840)
(146, 463)  (147, 552)  (149, 523)  (149, 763)  (149, 573)  (150, 890)
(151, 457)
(151, 644)  (151, 461)  (152, 573)  (153, 456)  (153, 573)  (153, 641)
(154, 593)
(154, 528)  (155, 916)  (155, 464)  (157, 691)  (158, 602)  (159, 737)
(159, 585)
写入数据块233
写入数据块234
写入数据块235
写入数据块236
写入数据块237
写入数据块238
写入数据块239
写入数据块240
**********************************************************************************
***********
```

```
第 4 轮内排序:
原始数据为:
(131, 472)  (120, 827)  (136, 608)  (146, 690)  (147, 750)  (145, 546)
(151, 578)
(123, 889)  (137, 854)  (136, 810)  (123, 791)  (137, 831)  (154, 585)
(134, 552)
(131, 617)  (146, 833)  (129, 683)  (130, 516)  (154, 460)  (150, 839)
(131, 718)
(143, 465)  (144, 650)  (138, 639)  (136, 743)  (131, 799)  (153, 613)
(134, 485)
(144, 838)  (145, 456)  (148, 876)  (150, 709)  (124, 767)  (128, 447)
(125, 901)
(148, 756)  (144, 667)  (122, 756)  (124, 854)  (124, 566)  (128, 871)
(127, 767)
(123, 587)  (135, 691)  (120, 554)  (127, 816)  (132, 606)  (137, 442)
(122, 546)
(139, 884)  (121, 475)  (156, 433)  (156, 839)  (149, 839)  (130, 476)
(122, 646)
经过排序的数据为:
(120, 827)  (120, 554)  (121, 475)  (122, 756)  (122, 546)  (122, 646)
(123, 889)
(123, 791)  (123, 587)  (124, 767)  (124, 854)  (124, 566)  (125, 901)
(127, 767)
(127, 816)  (128, 447)  (128, 871)  (129, 683)  (130, 516)  (130, 476)
(131, 472)
(131, 617)  (131, 718)  (131, 799)  (132, 606)  (134, 552)  (134, 485)
(135, 691)
(136, 608)  (136, 810)  (136, 743)  (137, 854)  (137, 831)  (137, 442)
(138, 639)
(139, 884)  (143, 465)  (144, 650)  (144, 838)  (144, 667)  (145, 546)
(145, 456)
(146, 690)  (146, 833)  (147, 750)  (148, 876)  (148, 756)  (149, 839)
(150, 839)
(150, 709)  (151, 578)  (153, 613)  (154, 585)  (154, 460)  (156, 433)
(156, 839)
写入数据块241
写入数据块242
写入数据块243
写入数据块244
写入数据块245
写入数据块246
写入数据块247
写入数据块248
结果块0 写入磁盘下标 317
结果块1 写入磁盘下标 318
结果块2 写入磁盘下标 319
结果块3 写入磁盘下标 320
结果块0 写入磁盘下标 321
结果块1 写入磁盘下标 322
结果块2 写入磁盘下标 323
结果块3 写入磁盘下标 324
结果块0 写入磁盘下标 325
结果块1 写入磁盘下标 326
结果块2 写入磁盘下标 327
结果块3 写入磁盘下标 328
```

```
结果块0 写入磁盘下标 329
结果块1 写入磁盘下标 330
结果块2 写入磁盘下标 331
结果块3 写入磁盘下标 332
结果块0 写入磁盘下标 333
结果块1 写入磁盘下标 334
结果块2 写入磁盘下标 335
结果块3 写入磁盘下标 336
结果块0 写入磁盘下标 337
结果块1 写入磁盘下标 338
结果块2 写入磁盘下标 339
结果块3 写入磁盘下标 340
结果块0 写入磁盘下标 341
结果块1 写入磁盘下标 342
结果块2 写入磁盘下标 343
结果块3 写入磁盘下标 344
结果块0 写入磁盘下标 345
结果块1 写入磁盘下标 346
结果块2 写入磁盘下标 347
结果块3 写入磁盘下标 348
**********************************************************************
************
```

基于索引的关系选择算法

- 问题分析：利用任务二中实现的TPMMS对关系进行排序后，将每个磁盘块的首个元组作为当前磁盘块的索引，并将所有磁盘块形成的索引表写入新的磁盘块进行记录，然后找到目标元组所在磁盘块号对应的索引上界和索引下界并在之间进行遍历即可
- 源代码：

```
/**
 * 基于索引的关系选择算法
 * @param start 磁盘块起始下标
 * @param end 磁盘块结束下标
 * @param num 所需要选择的数
 * @param index_col 排序的索引列下标
 */
void index_relationship_select(int start, int end, int num, int index_col) {
    // 声明缓冲区并初始化
    Buffer buf;
    initBuffer(520, 64, &buf);

    // 建立索引
    create_index(start, end, index_col);

    // 根据索引读取到的数据块
    int disk_num = 300 + start;
    // 索引块下标
    int index_num = 400 + start;
    unsigned char *data_blk = readBlockFromDisk(400 + start, &buf);

 printf("**********************************************************************
**********\n");
    printf("完成TPMMS排序\n");
```

```
 printf("**********************************************************************
**********\n");
    printf("所需查找的值为: %d\n", num);

    // 统计符合条件的数值
    int answer_num = 0;

    // 找到磁盘块下标上界
    int upbound_found = 0;
    while (!upbound_found) {
        char str_1[4];
        char str_2[4];
        int tuple[2] = {0};
        for (int i = 0; i < 8; i++) {
            if (i == 7) {
                // 刷新数据块
                freeBlockInBuffer(data_blk, &buf);
                index_num++;
                data_blk = readBlockFromDisk(index_num, &buf);
            } else {
                for (int k = 0; k < 4; k++) {
                    str_1[k] = *(data_blk + i * 8 + k);
                    str_2[k] = *(data_blk + i * 8 + 4 + k);
                }
                tuple[0] = atoi(str_1);
                tuple[1] = atoi(str_2);
                printf("当前索引为(%d, %d)\t", tuple[0], tuple[1]);
                printf("对应磁盘块下标为%d\n", disk_num);
                if (tuple[index_col] > num) {
                    upbound_found = 1;
                    break;
                } else {
                    disk_num++;
                }
            }
        }
    }
    printf("\n");
    printf("找到对应的磁盘块下标上界为: %d\n", disk_num);
    freeBlockInBuffer(data_blk, &buf);

    // 找到磁盘块下标下界
    int lowBound_found = 0;
    while (!lowBound_found) {
        char str_1[4];
        char str_2[4];
        int tuple[2] = {0};
        disk_num--;
        data_blk = readBlockFromDisk(disk_num, &buf);
        for (int k = 0; k < 4; k++) {
            str_1[k] = *(data_blk + k);
            str_2[k] = *(data_blk + 4 + k);
        }
        tuple[0] = atoi(str_1);
```

```
        tuple[1] = atoi(str_2);
        if (tuple[index_col] < num) {
            lowBound_found = 1;
        }
    }
    printf("找到对应的磁盘块下标下界为: %d\n", disk_num);
    freeBlockInBuffer(data_blk, &buf);

    // 找到所有符合条件的值
    int all_found = 0;
    unsigned char *result_blk = getNewBlockInBuffer(&buf);
    int result_num = 0;
    int write_disk = 500 + start;

    while (!all_found) {
        data_blk = readBlockFromDisk(disk_num, &buf);
        printf("\n当前读取块号为%d\n", disk_num);
        int tuple[2];
        for (int i = 0; i < 7; i++) {
            char str_1[4] = {0};
            char str_2[4] = {0};
            tuple[0] = 0;
            tuple[1] = 0;
            for (int k = 0; k < 4; k++) {
                str_1[k] = *(data_blk + i * 8 + k);
                str_2[k] = *(data_blk + i * 8 + 4 + k);
            }
            tuple[0] = atoi(str_1);
            tuple[1] = atoi(str_2);
//            printf("(%d, %d)\t", tuple[0], tuple[1]);
            if (tuple[index_col] == num) {
                printf("找到符合条件的值 (%d, %d)\n", tuple[0], tuple[1]);
                answer_num++;
                if (result_num == 7) {
                    // 写入下一块地址
                    char str_addr[4];
                    itoa(write_disk + 1, str_addr, 10);
                    for (int k = 0; k < 4; k++) {
                        *(result_blk + result_num * 8 + k) = str_addr[k];
                    }
                    // 写回磁盘块
                    writeBlockToDisk(result_blk, write_disk, &buf);
                    write_disk++;
                    // 更新计数器
                    result_num = 0;
                }
                for (int k = 0; k < 4; k++) {
                    *(result_blk + result_num * 8 + k) = str_1[k];
                    *(result_blk + result_num * 8 + 4 + k) = str_2[k];
                }
                result_num++;
            } else if (tuple[index_col] > num) {
                all_found = 1;
                break;
            }
```

```
            }
            disk_num++;
        }
        // 清除最后一块原有的数据
        for (int i = result_num; i < 8; i++) {
            if (i == 7) {
                // 写入下一块地址
                char str_addr[4];
                itoa(write_disk + 1, str_addr, 10);
                for (int k = 0; k < 4; k++) {
                    *(result_blk + 7 * 8 + k) = str_addr[k];
                }
            } else {
                for (int k = 0; k < 4; k++) {
                    *(result_blk + result_num * 8 + k) = 0;
                    *(result_blk + result_num * 8 + 4 + k) = 0;
                }
                result_num++;
            }
        }
        // 写回数据块
        writeBlockToDisk(result_blk, write_disk, &buf);
        freeBlockInBuffer(result_blk, &buf);
        printf("共有 %d 个符合条件的值\n", answer_num);
        printf("IO num: %d\n", buf.numIO);
    }
```

- 运行截图：

```
任务三：
********************************************************************************
************
第 1 轮内排序：
原始数据为：
(124, 605)  (146, 445)  (123, 532)  (124, 844)  (141, 754)  (135, 919)
(136, 454)
(123, 477)  (129, 486)  (130, 436)  (134, 437)  (139, 461)  (129, 430)
(120, 418)
(121, 739)  (130, 656)  (155, 544)  (127, 680)  (145, 805)  (123, 794)
(156, 600)
(151, 704)  (139, 878)  (133, 456)  (129, 591)  (124, 424)  (132, 640)
(154, 728)
(159, 630)  (137, 733)  (139, 576)  (141, 732)  (154, 810)  (148, 570)
(140, 610)
(135, 671)  (125, 587)  (137, 912)  (134, 457)  (125, 886)  (146, 721)
(157, 456)
(149, 873)  (128, 684)  (152, 717)  (155, 600)  (128, 431)  (148, 854)
(120, 736)
(130, 790)  (156, 809)  (129, 492)  (120, 581)  (125, 725)  (159, 447)
(157, 902)
经过排序的数据为：
(120, 418)  (120, 736)  (120, 581)  (121, 739)  (123, 532)  (123, 477)
(123, 794)
(124, 605)  (124, 844)  (124, 424)  (125, 587)  (125, 886)  (125, 725)
(127, 680)
```

```
(128, 684)  (128, 431)  (129, 486)  (129, 430)  (129, 591)  (129, 492)
(130, 436)
(130, 656)  (130, 790)  (132, 640)  (133, 456)  (134, 437)  (134, 457)
(135, 919)
(135, 671)  (136, 454)  (137, 733)  (137, 912)  (139, 461)  (139, 878)
(139, 576)
(140, 610)  (141, 754)  (141, 732)  (145, 805)  (146, 445)  (146, 721)
(148, 570)
(148, 854)  (149, 873)  (151, 704)  (152, 717)  (154, 728)  (154, 810)
(155, 544)
(155, 600)  (156, 600)  (156, 809)  (157, 456)  (157, 902)  (159, 630)
(159, 447)
写入数据块217
写入数据块218
写入数据块219
写入数据块220
写入数据块221
写入数据块222
写入数据块223
写入数据块224
**********************************************************************************
***********
第 2 轮内排序:
原始数据为:
(152, 634)  (122, 554)  (134, 494)  (128, 615)  (120, 775)  (148, 616)
(128, 429)
(142, 718)  (135, 542)  (121, 793)  (137, 460)  (137, 895)  (141, 636)
(148, 635)
(120, 571)  (148, 547)  (146, 843)  (156, 479)  (137, 585)  (145, 625)
(157, 498)
(123, 733)  (156, 698)  (156, 482)  (158, 764)  (144, 609)  (147, 663)
(142, 450)
(156, 857)  (142, 748)  (152, 826)  (147, 561)  (136, 711)  (139, 802)
(125, 496)
(140, 551)  (136, 434)  (153, 642)  (146, 834)  (158, 432)  (141, 629)
(128, 584)
(139, 488)  (145, 689)  (141, 504)  (122, 682)  (151, 444)  (129, 844)
(122, 760)
(150, 835)  (138, 512)  (120, 852)  (145, 521)  (134, 883)  (155, 668)
(139, 815)
经过排序的数据为:
(120, 775)  (120, 571)  (120, 852)  (121, 793)  (122, 554)  (122, 682)
(122, 760)
(123, 733)  (125, 496)  (128, 615)  (128, 429)  (128, 584)  (129, 844)
(134, 494)
(134, 883)  (135, 542)  (136, 711)  (136, 434)  (137, 460)  (137, 895)
(137, 585)
(138, 512)  (139, 802)  (139, 488)  (139, 815)  (140, 551)  (141, 636)
(141, 629)
(141, 504)  (142, 718)  (142, 450)  (142, 748)  (144, 609)  (145, 625)
(145, 689)
(145, 521)  (146, 843)  (146, 834)  (147, 663)  (147, 561)  (148, 616)
(148, 635)
(148, 547)  (150, 835)  (151, 444)  (152, 634)  (152, 826)  (153, 642)
(155, 668)
```

```
(156, 479)  (156, 698)  (156, 482)  (156, 857)  (157, 498)  (158, 764)
(158, 432)
写入数据块225
写入数据块226
写入数据块227
写入数据块228
写入数据块229
写入数据块230
写入数据块231
写入数据块232
**************************************************************************
************
第 3 轮内排序:
原始数据为:
(140, 915)  (145, 570)  (159, 737)  (129, 535)  (153, 456)  (121, 596)
(125, 625)
(151, 457)  (128, 592)  (126, 774)  (128, 457)  (126, 685)  (121, 582)
(145, 540)
(152, 573)  (131, 808)  (147, 552)  (130, 766)  (128, 720)  (151, 644)
(142, 679)
(146, 463)  (153, 573)  (157, 691)  (138, 700)  (130, 556)  (150, 890)
(134, 900)
(134, 756)  (130, 474)  (159, 585)  (120, 781)  (154, 593)  (155, 916)
(135, 747)
(154, 528)  (149, 523)  (158, 602)  (149, 763)  (122, 463)  (137, 660)
(155, 464)
(142, 841)  (127, 743)  (145, 840)  (131, 892)  (138, 816)  (143, 732)
(138, 674)
(125, 808)  (138, 598)  (123, 468)  (151, 461)  (149, 573)  (153, 641)
(126, 919)
经过排序的数据为:
(120, 781)  (121, 596)  (121, 582)  (122, 463)  (123, 468)  (125, 625)
(125, 808)
(126, 774)  (126, 685)  (126, 919)  (127, 743)  (128, 592)  (128, 457)
(128, 720)
(129, 535)  (130, 766)  (130, 556)  (130, 474)  (131, 808)  (131, 892)
(134, 900)
(134, 756)  (135, 747)  (137, 660)  (138, 700)  (138, 816)  (138, 674)
(138, 598)
(140, 915)  (142, 679)  (142, 841)  (143, 732)  (145, 570)  (145, 540)
(145, 840)
(146, 463)  (147, 552)  (149, 523)  (149, 763)  (149, 573)  (150, 890)
(151, 457)
(151, 644)  (151, 461)  (152, 573)  (153, 456)  (153, 573)  (153, 641)
(154, 593)
(154, 528)  (155, 916)  (155, 464)  (157, 691)  (158, 602)  (159, 737)
(159, 585)
写入数据块233
写入数据块234
写入数据块235
写入数据块236
写入数据块237
写入数据块238
写入数据块239
写入数据块240
```

```
**********************************************************************************
**********
第 4 轮内排序:
原始数据为:
(131, 472)  (120, 827)  (136, 608)  (146, 690)  (147, 750)  (145, 546)
(151, 578)
(123, 889)  (137, 854)  (136, 810)  (123, 791)  (137, 831)  (154, 585)
(134, 552)
(131, 617)  (146, 833)  (129, 683)  (130, 516)  (154, 460)  (150, 839)
(131, 718)
(143, 465)  (144, 650)  (138, 639)  (136, 743)  (131, 799)  (153, 613)
(134, 485)
(144, 838)  (145, 456)  (148, 876)  (150, 709)  (124, 767)  (128, 447)
(125, 901)
(148, 756)  (144, 667)  (122, 756)  (124, 854)  (124, 566)  (128, 871)
(127, 767)
(123, 587)  (135, 691)  (120, 554)  (127, 816)  (132, 606)  (137, 442)
(122, 546)
(139, 884)  (121, 475)  (156, 433)  (156, 839)  (149, 839)  (130, 476)
(122, 646)
经过排序的数据为:
(120, 827)  (120, 554)  (121, 475)  (122, 756)  (122, 546)  (122, 646)
(123, 889)
(123, 791)  (123, 587)  (124, 767)  (124, 854)  (124, 566)  (125, 901)
(127, 767)
(127, 816)  (128, 447)  (128, 871)  (129, 683)  (130, 516)  (130, 476)
(131, 472)
(131, 617)  (131, 718)  (131, 799)  (132, 606)  (134, 552)  (134, 485)
(135, 691)
(136, 608)  (136, 810)  (136, 743)  (137, 854)  (137, 831)  (137, 442)
(138, 639)
(139, 884)  (143, 465)  (144, 650)  (144, 838)  (144, 667)  (145, 546)
(145, 456)
(146, 690)  (146, 833)  (147, 750)  (148, 876)  (148, 756)  (149, 839)
(150, 839)
(150, 709)  (151, 578)  (153, 613)  (154, 585)  (154, 460)  (156, 433)
(156, 839)
写入数据块241
写入数据块242
写入数据块243
写入数据块244
写入数据块245
写入数据块246
写入数据块247
写入数据块248
结果块0 写入磁盘下标 317
结果块1 写入磁盘下标 318
结果块2 写入磁盘下标 319
结果块3 写入磁盘下标 320
结果块0 写入磁盘下标 321
结果块1 写入磁盘下标 322
结果块2 写入磁盘下标 323
结果块3 写入磁盘下标 324
结果块0 写入磁盘下标 325
结果块1 写入磁盘下标 326
```

```
结果块2 写入磁盘下标 327
结果块3 写入磁盘下标 328
结果块0 写入磁盘下标 329
结果块1 写入磁盘下标 330
结果块2 写入磁盘下标 331
结果块3 写入磁盘下标 332
结果块0 写入磁盘下标 333
结果块1 写入磁盘下标 334
结果块2 写入磁盘下标 335
结果块3 写入磁盘下标 336
结果块0 写入磁盘下标 337
结果块1 写入磁盘下标 338
结果块2 写入磁盘下标 339
结果块3 写入磁盘下标 340
结果块0 写入磁盘下标 341
结果块1 写入磁盘下标 342
结果块2 写入磁盘下标 343
结果块3 写入磁盘下标 344
结果块0 写入磁盘下标 345
结果块1 写入磁盘下标 346
结果块2 写入磁盘下标 347
结果块3 写入磁盘下标 348
********************************************************************************
************
完成TPMMS排序
********************************************************************************
************
所需查找的值为: 128
当前索引为(120, 827) 对应磁盘块下标为317
当前索引为(120, 736) 对应磁盘块下标为318
当前索引为(122, 756) 对应磁盘块下标为319
当前索引为(123, 889) 对应磁盘块下标为320
当前索引为(123, 794) 对应磁盘块下标为321
当前索引为(125, 901) 对应磁盘块下标为322
当前索引为(126, 774) 对应磁盘块下标为323
当前索引为(128, 447) 对应磁盘块下标为324
当前索引为(128, 584) 对应磁盘块下标为325
当前索引为(129, 430) 对应磁盘块下标为326

找到对应的磁盘块下标上界为: 326
找到对应的磁盘块下标下界为: 323

当前读取块号为323

当前读取块号为324
找到符合条件的值 (128, 447)
找到符合条件的值 (128, 871)
找到符合条件的值 (128, 592)
找到符合条件的值 (128, 457)
找到符合条件的值 (128, 720)
找到符合条件的值 (128, 615)
找到符合条件的值 (128, 429)

当前读取块号为325
找到符合条件的值 (128, 584)
```

```
找到符合条件的值 (128, 684)
找到符合条件的值 (128, 431)
共有 10 个符合条件的值
IO num: 10
********************************************************************************
************
```

基于排序的连接操作算法

- 问题分析:
  - 变量声明:

```
/**
 * 基于排序的连接操作
 * @param start_1 关系1的起始下标
 * @param end_1 关系1的结束下标
 * @param select_col_1 关系1的排序索引列下标
 * @param start_2 关系2的起始下标
 * @param end_2 关系2的结束下标
 * @param select_col_2 关系2的排序索引列下标
 * @param write_base 写入磁盘块的基底
 */
```

  - 首先对关系进行子表排序

```
sort_blk_data(start_1, end_1, select_col_1, write_base);
sort_blk_data(start_2, end_2, select_col_2, write_base);
```

  - 找到关系R中两个指针所指的较小的值

```
        /*
         * 找到关系R当前两个指针所指的较小的值
         */
        int min_tuple[2] = {1000, 1000};
        int select_group = -1;
//        printf("选择关系R元组\n");
//        printf("当前选择的两组数据分别为: ");
        for (int group_index = 0; group_index < group_1_num;
group_index++) {
            if (cur_index_1[group_index] == 63) {
//                printf("关系R元组的第 %d 组判断完毕\n", group_index);
                finish_1[group_index] = 1;
                continue;
            }
            // 计算当前下标所对应的块内偏移量
            int offset = cur_index_1[group_index] % 8;
            // 更新当前块指针
            if (offset == 7) {
                // 计算下一块下标
                char str_addr[4];
                for (int k = 0; k < 4; k++) {
                    str_addr[k] = *(group_1[group_index] + offset * 8 +
k);
```

```
                }
                int next_disk = write_base + atoi(str_addr);
                // 释放原有磁盘块
                freeBlockInBuffer(group_1[group_index], &buf);
                // 读取新的磁盘块
                group_1[group_index] = readBlockFromDisk(next_disk,
&buf);
                cur_index_1[group_index]++;
                // 更新offset
                offset = cur_index_1[group_index] % 8;
            }
            // 记录数据
            char str_1[4];
            char str_2[4];
            for (int k = 0; k < 4; k++) {
                str_1[k] = *(group_1[group_index] + offset * 8 + k);
                str_2[k] = *(group_1[group_index] + offset * 8 + 4 + k);
            }
            int cur_tuple[2] = {atoi(str_1), atoi(str_2)};
//              printf("(%d, %d)\t", cur_tuple[0], cur_tuple[1]);
            // 数据更小则更新
            if (cur_tuple[select_col_1] <= min_tuple[select_col_1]) {
                min_tuple[0] = cur_tuple[0];
                min_tuple[1] = cur_tuple[1];
                select_group = group_index;
            }
        }
//          printf("\n");
//          printf("最终选择数据为: (%d, %d)\n", min_tuple[0],
min_tuple[1]);
//          printf("两组当前下标分别为:\n");
//          printf("group 0: %d, group 1: %d\n", cur_index_1[0],
cur_index_1[1]);
        // 更新下标
        cur_index_1[select_group]++;
```

- 再判断关系R是否读取完毕

```
        /*
         * 判断关系R是否完全读取完毕
         */
        int finish_num = 0;
        while (finish_num < group_1_num && (finish_1[finish_num] == 1))
{
            finish_num++;
        }
        if (finish_num == group_1_num) {
            printf("\n关系R的元组读取完毕\n");
            printf("\n");
            break;
        }
```

- 再在关系S中选出与关系R对应列值相等的元组

```
        /*
         * 找到关系S中与关系R选出的值相等的元组
         */
//         printf("选择关系S元组\n");
        int last_group_2_index[group_2_num];
        int last_blk_2[group_2_num];

        for (int i = 0; i < group_2_num; i++) {
            last_group_2_index[i] = cur_index_2[i];
        }
        for (int i = 0; i < group_2_num; i++) {
            last_blk_2[i] = cur_blk_2[i];
        }

        int equal_num = 0;
        for (int group_index = 0; group_index < group_2_num;
group_index++) {
            // 如果当前组已经读取完毕
            if (cur_index_2[group_index] == 63) {
                finish_2[group_index] = 1;
                continue;
            }
            // 当前组所有符合条件元素都已经选择完毕的标记变量
            int cur_group_all_found = 0;
            while (!cur_group_all_found) {
                // 计算当前下标所对应的块内偏移量
                int offset = cur_index_2[group_index] % 8;
                // 更新当前块指针
                if (offset == 7) {
                    char str_addr[4];
                    for (int k = 0; k < 4; k++) {
                        str_addr[k] = *(group_2[group_index] + offset *
8 + k);
                    }
                    int next_disk = write_base + atoi(str_addr);
                    // 释放原有磁盘块
                    freeBlockInBuffer(group_2[group_index], &buf);
                    // 读取新的磁盘块
                    group_2[group_index] = readBlockFromDisk(next_disk,
&buf);
                    cur_index_2[group_index]++;
                    // 更新offset
                    offset = cur_index_2[group_index] % 8;
                    // 更新磁盘块下标
                    cur_blk_2[group_index] = write_base + start_2 +
group_index * 8 + cur_index_2[group_index] / 8;
                }
                // 记录数据
                char str_1[4];
                char str_2[4];
                read_from_blk(group_2[group_index], offset, str_1,
str_2);
                int cur_tuple[2] = {atoi(str_1), atoi(str_2)};
//                 printf("当前组号为: %d\t 块号为: %d\t 偏移量为: %d\n",
group_index, cur_blk_2[group_index], cur_index_2[group_index]);
```

```
//                printf("当前数据为: (%d, %d)\n", cur_tuple[0],
cur_tuple[1]);
                // 判断结果块是否写满的标记变量
                int all_full = 0;
                // 找到小于的元素则继续往后移
                if (cur_tuple[select_col_2] < min_tuple[select_col_1]) {
//                    printf("当前选择数据为(%d, %d)\t 小于所选数据\n",
cur_tuple[0], cur_tuple[1]);
                    // 移动当前指针对应下标
                    cur_index_2[group_index]++;
                    // 移动所需要恢复的下标位置
                    last_group_2_index[group_index]++;
                    // 移动需要恢复的磁盘块号
                    if (last_group_2_index[group_index] % 8 == 0) {
                        last_blk_2[group_index]++;
                    }
                }
                    // 找到相等的元素则写入结果块
                else if (cur_tuple[select_col_2] ==
min_tuple[select_col_1]) {
                    equal_num++;
                    join_num++;
//                    printf("当前选择数据为(%d, %d)\t 匹配成功\n",
cur_tuple[0], cur_tuple[1]);
//                    printf("当前连接数为: %d\n", join_num);
                    char str_A[4];
                    char str_B[4];
                    char str_C[4];
                    char str_D[4];
                    itoa(min_tuple[0], str_A, 10);
                    itoa(min_tuple[1], str_B, 10);
                    itoa(cur_tuple[0], str_C, 10);
                    itoa(cur_tuple[1], str_D, 10);
                    for (int j = 0; j < 8 - group_1_num - group_2_num;
j++) {
                        // 所有结果块都已经写满
                        if (result_num[j] == 6 && j == 7 - group_1_num -
group_2_num) {
                            for (int k = 0; k < (8 - group_1_num -
group_2_num); k++) {
                                // 写回当前磁盘块
                                char str_addr[4];
                                itoa(write_disk + 1, str_addr, 10);
                                for (int m = 0; m < 4; m++) {
                                    *(result_blk[k] + 7 * 8 + m) =
str_addr[m];
                                }
                                writeBlockToDisk(result_blk[k],
write_disk, &buf);
                                printf("结果块%d 写入磁盘下标 %d\n", k,
write_disk);
                                write_disk++;
                                // 申请新的磁盘块
                                result_blk[k] =
getNewBlockInBuffer(&buf);
```

```
                                result_num[k] = 0;
                            }
                        }
                            // 当前块写满
                        else if (result_num[j] == 6 && j < (7 -
group_1_num - group_2_num)) {
                            continue;
                        }
                        for (int k = 0; k < 4; k++) {
                            *(result_blk[j] + result_num[j] * 8 + k) =
str_C[k];
                            *(result_blk[j] + result_num[j] * 8 + 4 + k)
= str_D[k];
                            *(result_blk[j] + result_num[j] * 8 + 8 + k)
= str_A[k];
                            *(result_blk[j] + result_num[j] * 8 + 12 +
k) = str_B[k];
                        }
                        result_num[j] += 2;
                        break;
                    }
                    cur_index_2[group_index]++;
                }
                    // 如果当前所指元素大于选定的元素则开始判断下一组
                else if (cur_tuple[select_col_2] >
min_tuple[select_col_1]) {
//                    printf("当前元组数据大于所选数据，第 %d 组判断完毕\n",
group_index);
                    cur_group_all_found = 1;
                }
            }
        }
//        printf("每个组需要恢复到的磁盘块号为: ");
        for (int i = 0; i < group_2_num; i++) {
//            printf("%d\t", last_blk_2[i]);
        }
//        printf("\n");
        // 还原每个指针读取的磁盘块下标
        for (int i = 0; i < group_2_num; i++) {
            cur_blk_2[i] = last_blk_2[i];
            freeBlockInBuffer(group_2[i], &buf);
            group_2[i] = readBlockFromDisk(cur_blk_2[i], &buf);
        }
//        printf("每个组需要恢复到的下标标号为: ");
        for (int i = 0; i < group_2_num; i++) {
//            printf("%d\t", last_group_2_index[i]);
        }
//        printf("\n");
        // 还原关系S对应组的下标
        for (int i = 0; i < group_2_num; i++) {
            cur_index_2[i] = last_group_2_index[i];
        }
//        printf("当前数据(%d, %d) 连接总数为: %d\n", min_tuple[0],
min_tuple[1], equal_num);
//        printf("****************************");
```

```
    }
```

- 整体源代码

```
/**
 * 基于排序的连接操作
 * @param start_1 关系1的起始下标
 * @param end_1 关系1的结束下标
 * @param select_col_1 关系1的排序索引列下标
 * @param start_2 关系2的起始下标
 * @param end_2 关系2的结束下标
 * @param select_col_2 关系2的排序索引列下标
 * @param write_base 写入磁盘块的基底
 */
void
sort_merge_join(int start_1, int end_1, int select_col_1, int start_2, int
end_2, int select_col_2, int write_base) {
    Buffer buf;
    initBuffer(520, 64, &buf);

    // 对原始数据进行内排序

 printf("**************************************************************************
**********************\n");
    sort_blk_data(start_1, end_1, select_col_1, write_base);
    sort_blk_data(start_2, end_2, select_col_2, write_base);
    printf("分别对关系R和关系S完成内排序\n");

 printf("**************************************************************************
**********************\n");

    // 计算每个属性的组数
    int group_1_num = (end_1 + 1 - start_1) / 8;
    int group_2_num = (end_2 + 1 - start_2) / 8;

    // 指向每个组的指针
    unsigned char *group_1[group_1_num];
    unsigned char *group_2[group_2_num];
    memset(group_1, 0, sizeof group_1);
    memset(group_2, 0, sizeof group_2);

    // 指针指向对应的磁盘块
    for (int i = 0; i < group_1_num; i++) {
        int addr = write_base + start_1 + i * 8;
        group_1[i] = readBlockFromDisk(addr, &buf);
    }
    for (int i = 0; i < group_2_num; i++) {
        int addr = write_base + start_2 + i * 8;
        group_2[i] = readBlockFromDisk(addr, &buf);
    }

    // 定义每个组当前读取到的数据下标
    int cur_index_1[group_1_num];
    int cur_index_2[group_2_num];
    memset(cur_index_1, 0, sizeof cur_index_1);
```

```
    memset(cur_index_2, 0, sizeof cur_index_2);

    // 定义第二组当前读取到的磁盘下标
    int cur_blk_2[group_2_num];
    for (int i = 0; i < group_2_num; i++) {
        cur_blk_2[i] = write_base + start_2 + i * 8;
    }

    // 读取完毕的标记变量
    int finish_1[group_1_num];
    int finish_2[group_2_num];
    memset(finish_1, 0, group_1_num);
    memset(finish_2, 0, group_2_num);

    // 写回磁盘块的首下标
    int write_disk = write_base + 100;
    // 声明结果块
    unsigned char *result_blk[8 - group_1_num - group_2_num];
    for (int i = 0; i < (8 - group_1_num - group_2_num); i++) {
        result_blk[i] = getNewBlockInBuffer(&buf);
    }
    int result_num[8 - group_1_num - group_2_num];
    memset(result_num, 0, sizeof result_num);

    int join_num = 0;
    // 连接操作
    while (1) {
//        printf("********************************\n");
        /*
         * 找到关系R当前两个指针所指的较小的值
         */
        int min_tuple[2] = {1000, 1000};
        int select_group = -1;
//        printf("选择关系R元组\n");
//        printf("当前选择的两组数据分别为: ");
        for (int group_index = 0; group_index < group_1_num; group_index++)
{
            if (cur_index_1[group_index] == 63) {
//                printf("关系R元组的第 %d 组判断完毕\n", group_index);
                finish_1[group_index] = 1;
                continue;
            }
            // 计算当前下标所对应的块内偏移量
            int offset = cur_index_1[group_index] % 8;
            // 更新当前块指针
            if (offset == 7) {
                // 计算下一块下标
                char str_addr[4];
                for (int k = 0; k < 4; k++) {
                    str_addr[k] = *(group_1[group_index] + offset * 8 + k);
                }
                int next_disk = write_base + atoi(str_addr);
                // 释放原有磁盘块
                freeBlockInBuffer(group_1[group_index], &buf);
                // 读取新的磁盘块
```

```
                group_1[group_index] = readBlockFromDisk(next_disk, &buf);
                cur_index_1[group_index]++;
                // 更新offset
                offset = cur_index_1[group_index] % 8;
            }
            // 记录数据
            char str_1[4];
            char str_2[4];
            for (int k = 0; k < 4; k++) {
                str_1[k] = *(group_1[group_index] + offset * 8 + k);
                str_2[k] = *(group_1[group_index] + offset * 8 + 4 + k);
            }
            int cur_tuple[2] = {atoi(str_1), atoi(str_2)};
//            printf("(%d, %d)\t", cur_tuple[0], cur_tuple[1]);
            // 数据更小则更新
            if (cur_tuple[select_col_1] <= min_tuple[select_col_1]) {
                min_tuple[0] = cur_tuple[0];
                min_tuple[1] = cur_tuple[1];
                select_group = group_index;
            }
        }
//        printf("\n");
//        printf("最终选择数据为: (%d, %d)\n", min_tuple[0], min_tuple[1]);
//        printf("两组当前下标分别为:\n");
//        printf("group 0: %d, group 1: %d\n", cur_index_1[0],
cur_index_1[1]);
        // 更新下标
        cur_index_1[select_group]++;

        /*
         * 判断关系R是否完全读取完毕
         */
        int finish_num = 0;
        while (finish_num < group_1_num && (finish_1[finish_num] == 1)) {
            finish_num++;
        }
        if (finish_num == group_1_num) {
            printf("\n关系R的元组读取完毕\n");
            printf("\n");
            break;
        }

        /*
         * 找到关系S中与关系R选出的值相等的元组
         */
//        printf("选择关系S元组\n");
        int last_group_2_index[group_2_num];
        int last_blk_2[group_2_num];

        for (int i = 0; i < group_2_num; i++) {
            last_group_2_index[i] = cur_index_2[i];
        }
        for (int i = 0; i < group_2_num; i++) {
            last_blk_2[i] = cur_blk_2[i];
        }
```

```
        int equal_num = 0;
        for (int group_index = 0; group_index < group_2_num; group_index++)
{
            // 如果当前组已经读取完毕
            if (cur_index_2[group_index] == 63) {
                finish_2[group_index] = 1;
                continue;
            }
            // 当前组所有符合条件元素都已经选择完毕的标记变量
            int cur_group_all_found = 0;
            while (!cur_group_all_found) {
                // 计算当前下标所对应的块内偏移量
                int offset = cur_index_2[group_index] % 8;
                // 更新当前块指针
                if (offset == 7) {
                    char str_addr[4];
                    for (int k = 0; k < 4; k++) {
                        str_addr[k] = *(group_2[group_index] + offset * 8 +
k);
                    }
                    int next_disk = write_base + atoi(str_addr);
                    // 释放原有磁盘块
                    freeBlockInBuffer(group_2[group_index], &buf);
                    // 读取新的磁盘块
                    group_2[group_index] = readBlockFromDisk(next_disk,
&buf);
                    cur_index_2[group_index]++;
                    // 更新offset
                    offset = cur_index_2[group_index] % 8;
                    // 更新磁盘块下标
                    cur_blk_2[group_index] = write_base + start_2 +
group_index * 8 + cur_index_2[group_index] / 8;
                }
                // 记录数据
                char str_1[4];
                char str_2[4];
                read_from_blk(group_2[group_index], offset, str_1, str_2);
                int cur_tuple[2] = {atoi(str_1), atoi(str_2)};
//                printf("当前组号为: %d\t 块号为: %d\t 偏移量为: %d\n",
group_index, cur_blk_2[group_index], cur_index_2[group_index]);
//                printf("当前数据为: (%d, %d)\n", cur_tuple[0],
cur_tuple[1]);
                // 判断结果块是否写满的标记变量
                int all_full = 0;
                // 找到小于的元素则继续往后移
                if (cur_tuple[select_col_2] < min_tuple[select_col_1]) {
//                    printf("当前选择数据为(%d, %d)\t 小于所选数据\n",
cur_tuple[0], cur_tuple[1]);
                    // 移动当前指针对应下标
                    cur_index_2[group_index]++;
                    // 移动所需要恢复的下标位置
                    last_group_2_index[group_index]++;
                    // 移动需要恢复的磁盘块号
                    if (last_group_2_index[group_index] % 8 == 0) {
```

```
                        last_blk_2[group_index]++;
                    }
                }
                    // 找到相等的元素则写入结果块
                else if (cur_tuple[select_col_2] == min_tuple[select_col_1])
{
                    equal_num++;
                    join_num++;
//                    printf("当前选择数据为(%d, %d)\t 匹配成功\n",
cur_tuple[0], cur_tuple[1]);
//                    printf("当前连接数为: %d\n", join_num);
                    char str_A[4];
                    char str_B[4];
                    char str_C[4];
                    char str_D[4];
                    itoa(min_tuple[0], str_A, 10);
                    itoa(min_tuple[1], str_B, 10);
                    itoa(cur_tuple[0], str_C, 10);
                    itoa(cur_tuple[1], str_D, 10);
                    for (int j = 0; j < 8 - group_1_num - group_2_num; j++)
{
                        // 所有结果块都已经写满
                        if (result_num[j] == 6 && j == 7 - group_1_num -
group_2_num) {
                            for (int k = 0; k < (8 - group_1_num -
group_2_num); k++) {
                                // 写回当前磁盘块
                                char str_addr[4];
                                itoa(write_disk + 1, str_addr, 10);
                                for (int m = 0; m < 4; m++) {
                                    *(result_blk[k] + 7 * 8 + m) =
str_addr[m];
                                }
                                writeBlockToDisk(result_blk[k], write_disk,
&buf);
                                printf("结果块%d 写入磁盘下标 %d\n", k,
write_disk);
                                write_disk++;
                                // 申请新的磁盘块
                                result_blk[k] = getNewBlockInBuffer(&buf);
                                result_num[k] = 0;
                            }
                        }
                            // 当前块写满
                        else if (result_num[j] == 6 && j < (7 - group_1_num
- group_2_num)) {
                            continue;
                        }
                        for (int k = 0; k < 4; k++) {
                            *(result_blk[j] + result_num[j] * 8 + k) =
str_C[k];
                            *(result_blk[j] + result_num[j] * 8 + 4 + k) =
str_D[k];
                            *(result_blk[j] + result_num[j] * 8 + 8 + k) =
str_A[k];
```

```
                            *(result_blk[j] + result_num[j] * 8 + 12 + k) =
str_B[k];
                        }
                        result_num[j] += 2;
                        break;
                    }
                    cur_index_2[group_index]++;
                }
                    // 如果当前所指元素大于选定的元素则开始判断下一组
                else if (cur_tuple[select_col_2] > min_tuple[select_col_1])
{
//                      printf("当前元组数据大于所选数据，第 %d 组判断完毕\n",
group_index);
                    cur_group_all_found = 1;
                }
            }
        }
//        printf("每个组需要恢复到的磁盘块号为: ");
        for (int i = 0; i < group_2_num; i++) {
//            printf("%d\t", last_blk_2[i]);
        }
//        printf("\n");
        // 还原每个指针读取的磁盘块下标
        for (int i = 0; i < group_2_num; i++) {
            cur_blk_2[i] = last_blk_2[i];
            freeBlockInBuffer(group_2[i], &buf);
            group_2[i] = readBlockFromDisk(cur_blk_2[i], &buf);
        }
//        printf("每个组需要恢复到的下标标号为: ");
        for (int i = 0; i < group_2_num; i++) {
//            printf("%d\t", last_group_2_index[i]);
        }
//        printf("\n");
        // 还原关系S对应组的下标
        for (int i = 0; i < group_2_num; i++) {
            cur_index_2[i] = last_group_2_index[i];
        }
//        printf("当前数据(%d, %d) 连接总数为: %d\n", min_tuple[0],
min_tuple[1], equal_num);
//        printf("****************************");
    }
    // 最后数据未写满时也要写入结果
    int result_blk_num = join_num / 3;
    int last_blk_num = join_num % 3;
    for (int k = 0; k < (8 - group_1_num - group_2_num); k++) {
        // 最后一块
        if (write_disk == write_base + 100 + result_blk_num) {
            // 没写满需要删去部分内容
            for (int i = 2 * last_blk_num; i < 7; i++) {
                for (int m = 0; m < 4; m++) {
                    *(result_blk[k] + i * 8 + m) = 0;
                    *(result_blk[k] + i * 8 + 4 + m) = 0;
                }
            }
        }
```

```
            // 多余的块不写
        else if (write_disk > write_base + 100 + result_blk_num) {
            continue;
        }
        // 写回当前磁盘块
        char str_addr[4];
        itoa(write_disk + 1, str_addr, 10);
        for (int m = 0; m < 4; m++) {
            *(result_blk[k] + 7 * 8 + m) = str_addr[m];
        }
        writeBlockToDisk(result_blk[k], write_disk, &buf);
        printf("结果块%d 写入磁盘下标 %d\n", k, write_disk);
        write_disk++;
        // 申请新的磁盘块
        result_blk[k] = getNewBlockInBuffer(&buf);
        result_num[k] = 0;
    }
    printf("连接总数为: %d\n", join_num);
}
```

- 运行结果

```
任务四:
**************************************************************************************
************
**************************************************************************************
************
第 1 轮内排序:
原始数据为:
(131, 479)  (123, 477)  (107, 411)  (139, 476)  (106, 461)  (119, 406)
(105, 497)
(137, 473)  (137, 451)  (134, 459)  (138, 411)  (100, 421)  (109, 472)
(135, 441)
(133, 455)  (130, 436)  (121, 438)  (124, 468)  (117, 438)  (123, 422)
(131, 454)
(109, 404)  (139, 461)  (116, 421)  (122, 474)  (138, 497)  (114, 410)
(118, 414)
(102, 465)  (117, 475)  (137, 420)  (108, 436)  (120, 418)  (101, 410)
(101, 406)
(105, 428)  (109, 409)  (101, 470)  (124, 412)  (104, 450)  (119, 428)
(107, 477)
(115, 401)  (129, 455)  (133, 467)  (105, 476)  (137, 428)  (114, 425)
(102, 413)
(120, 827)  (122, 546)  (125, 886)  (127, 767)  (128, 447)  (130, 656)
(140, 610)
经过排序的数据为:
(100, 421)  (101, 410)  (101, 406)  (101, 470)  (102, 465)  (102, 413)
(104, 450)
(105, 497)  (105, 428)  (105, 476)  (106, 461)  (107, 411)  (107, 477)
(108, 436)
(109, 472)  (109, 404)  (109, 409)  (114, 410)  (114, 425)  (115, 401)
(116, 421)
(117, 438)  (117, 475)  (118, 414)  (119, 406)  (119, 428)  (120, 418)
(120, 827)
```

```
(121, 438)  (122, 474)  (122, 546)  (123, 477)  (123, 422)  (124, 468)
(124, 412)
(125, 886)  (127, 767)  (128, 447)  (129, 455)  (130, 436)  (130, 656)
(131, 479)
(131, 454)  (133, 455)  (133, 467)  (134, 459)  (135, 441)  (137, 473)
(137, 451)
(137, 420)  (137, 428)  (138, 411)  (138, 497)  (139, 476)  (139, 461)
(140, 610)
写入数据块601
写入数据块602
写入数据块603
写入数据块604
写入数据块605
写入数据块606
写入数据块607
写入数据块608
********************************************************************************
**********
第 2 轮内排序:
原始数据为:
(137, 471)  (120, 499)  (133, 441)  (126, 423)  (129, 488)  (116, 470)
(124, 410)
(100, 439)  (101, 405)  (138, 468)  (123, 452)  (117, 442)  (104, 440)
(133, 428)
(132, 483)  (116, 452)  (110, 405)  (138, 475)  (128, 453)  (124, 426)
(116, 414)
(102, 492)  (129, 475)  (129, 402)  (131, 492)  (101, 409)  (114, 414)
(107, 434)
(121, 464)  (116, 424)  (130, 411)  (110, 413)  (128, 459)  (129, 430)
(117, 403)
(100, 400)  (117, 426)  (110, 491)  (138, 495)  (138, 407)  (130, 495)
(137, 480)
(126, 485)  (134, 486)  (108, 482)  (119, 431)  (111, 445)  (112, 467)
(117, 412)
(124, 499)  (130, 417)  (118, 478)  (116, 420)  (132, 422)  (110, 450)
(134, 437)
经过排序的数据为:
(100, 439)  (100, 400)  (101, 405)  (101, 409)  (102, 492)  (104, 440)
(107, 434)
(108, 482)  (110, 405)  (110, 413)  (110, 491)  (110, 450)  (111, 445)
(112, 467)
(114, 414)  (116, 470)  (116, 452)  (116, 414)  (116, 424)  (116, 420)
(117, 442)
(117, 403)  (117, 426)  (117, 412)  (118, 478)  (119, 431)  (120, 499)
(121, 464)
(123, 452)  (124, 410)  (124, 426)  (124, 499)  (126, 423)  (126, 485)
(128, 453)
(128, 459)  (129, 488)  (129, 475)  (129, 402)  (129, 430)  (130, 411)
(130, 495)
(130, 417)  (131, 492)  (132, 483)  (132, 422)  (133, 441)  (133, 428)
(134, 486)
(134, 437)  (137, 471)  (137, 480)  (138, 468)  (138, 475)  (138, 495)
(138, 407)
写入数据块609
写入数据块610
```

```
写入数据块611
写入数据块612
写入数据块613
写入数据块614
写入数据块615
写入数据块616
**********************************************************************
**********
第 1 轮内排序:
原始数据为:
(124, 605)  (146, 445)  (123, 532)  (124, 844)  (141, 754)  (135, 919)
(136, 454)
(123, 477)  (129, 486)  (130, 436)  (134, 437)  (139, 461)  (129, 430)
(120, 418)
(121, 739)  (130, 656)  (155, 544)  (127, 680)  (145, 805)  (123, 794)
(156, 600)
(151, 704)  (139, 878)  (133, 456)  (129, 591)  (124, 424)  (132, 640)
(154, 728)
(159, 630)  (137, 733)  (139, 576)  (141, 732)  (154, 810)  (148, 570)
(140, 610)
(135, 671)  (125, 587)  (137, 912)  (134, 457)  (125, 886)  (146, 721)
(157, 456)
(149, 873)  (128, 684)  (152, 717)  (155, 600)  (128, 431)  (148, 854)
(120, 736)
(130, 790)  (156, 809)  (129, 492)  (120, 581)  (125, 725)  (159, 447)
(157, 902)
经过排序的数据为:
(120, 418)  (120, 736)  (120, 581)  (121, 739)  (123, 532)  (123, 477)
(123, 794)
(124, 605)  (124, 844)  (124, 424)  (125, 587)  (125, 886)  (125, 725)
(127, 680)
(128, 684)  (128, 431)  (129, 486)  (129, 430)  (129, 591)  (129, 492)
(130, 436)
(130, 656)  (130, 790)  (132, 640)  (133, 456)  (134, 437)  (134, 457)
(135, 919)
(135, 671)  (136, 454)  (137, 733)  (137, 912)  (139, 461)  (139, 878)
(139, 576)
(140, 610)  (141, 754)  (141, 732)  (145, 805)  (146, 445)  (146, 721)
(148, 570)
(148, 854)  (149, 873)  (151, 704)  (152, 717)  (154, 728)  (154, 810)
(155, 544)
(155, 600)  (156, 600)  (156, 809)  (157, 456)  (157, 902)  (159, 630)
(159, 447)
写入数据块617
写入数据块618
写入数据块619
写入数据块620
写入数据块621
写入数据块622
写入数据块623
写入数据块624
**********************************************************************
**********
第 2 轮内排序:
原始数据为:
```

(152, 634) (122, 554) (134, 494) (128, 615) (120, 775) (148, 616)
(128, 429)
(142, 718) (135, 542) (121, 793) (137, 460) (137, 895) (141, 636)
(148, 635)
(120, 571) (148, 547) (146, 843) (156, 479) (137, 585) (145, 625)
(157, 498)
(123, 733) (156, 698) (156, 482) (158, 764) (144, 609) (147, 663)
(142, 450)
(156, 857) (142, 748) (152, 826) (147, 561) (136, 711) (139, 802)
(125, 496)
(140, 551) (136, 434) (153, 642) (146, 834) (158, 432) (141, 629)
(128, 584)
(139, 488) (145, 689) (141, 504) (122, 682) (151, 444) (129, 844)
(122, 760)
(150, 835) (138, 512) (120, 852) (145, 521) (134, 883) (155, 668)
(139, 815)
经过排序的数据为:
(120, 775) (120, 571) (120, 852) (121, 793) (122, 554) (122, 682)
(122, 760)
(123, 733) (125, 496) (128, 615) (128, 429) (128, 584) (129, 844)
(134, 494)
(134, 883) (135, 542) (136, 711) (136, 434) (137, 460) (137, 895)
(137, 585)
(138, 512) (139, 802) (139, 488) (139, 815) (140, 551) (141, 636)
(141, 629)
(141, 504) (142, 718) (142, 450) (142, 748) (144, 609) (145, 625)
(145, 689)
(145, 521) (146, 843) (146, 834) (147, 663) (147, 561) (148, 616)
(148, 635)
(148, 547) (150, 835) (151, 444) (152, 634) (152, 826) (153, 642)
(155, 668)
(156, 479) (156, 698) (156, 482) (156, 857) (157, 498) (158, 764)
(158, 432)
写入数据块625
写入数据块626
写入数据块627
写入数据块628
写入数据块629
写入数据块630
写入数据块631
写入数据块632
****
第 3 轮内排序:
原始数据为:
(140, 915) (145, 570) (159, 737) (129, 535) (153, 456) (121, 596)
(125, 625)
(151, 457) (128, 592) (126, 774) (128, 457) (126, 685) (121, 582)
(145, 540)
(152, 573) (131, 808) (147, 552) (130, 766) (128, 720) (151, 644)
(142, 679)
(146, 463) (153, 573) (157, 691) (138, 700) (130, 556) (150, 890)
(134, 900)
(134, 756) (130, 474) (159, 585) (120, 781) (154, 593) (155, 916)
(135, 747)

```
(154, 528)  (149, 523)  (158, 602)  (149, 763)  (122, 463)  (137, 660)
(155, 464)
(142, 841)  (127, 743)  (145, 840)  (131, 892)  (138, 816)  (143, 732)
(138, 674)
(125, 808)  (138, 598)  (123, 468)  (151, 461)  (149, 573)  (153, 641)
(126, 919)
经过排序的数据为:
(120, 781)  (121, 596)  (121, 582)  (122, 463)  (123, 468)  (125, 625)
(125, 808)
(126, 774)  (126, 685)  (126, 919)  (127, 743)  (128, 592)  (128, 457)
(128, 720)
(129, 535)  (130, 766)  (130, 556)  (130, 474)  (131, 808)  (131, 892)
(134, 900)
(134, 756)  (135, 747)  (137, 660)  (138, 700)  (138, 816)  (138, 674)
(138, 598)
(140, 915)  (142, 679)  (142, 841)  (143, 732)  (145, 570)  (145, 540)
(145, 840)
(146, 463)  (147, 552)  (149, 523)  (149, 763)  (149, 573)  (150, 890)
(151, 457)
(151, 644)  (151, 461)  (152, 573)  (153, 456)  (153, 573)  (153, 641)
(154, 593)
(154, 528)  (155, 916)  (155, 464)  (157, 691)  (158, 602)  (159, 737)
(159, 585)
写入数据块633
写入数据块634
写入数据块635
写入数据块636
写入数据块637
写入数据块638
写入数据块639
写入数据块640
********************************************************************************
**********
第 4 轮内排序:
原始数据为:
(131, 472)  (120, 827)  (136, 608)  (146, 690)  (147, 750)  (145, 546)
(151, 578)
(123, 889)  (137, 854)  (136, 810)  (123, 791)  (137, 831)  (154, 585)
(134, 552)
(131, 617)  (146, 833)  (129, 683)  (130, 516)  (154, 460)  (150, 839)
(131, 718)
(143, 465)  (144, 650)  (138, 639)  (136, 743)  (131, 799)  (153, 613)
(134, 485)
(144, 838)  (145, 456)  (148, 876)  (150, 709)  (124, 767)  (128, 447)
(125, 901)
(148, 756)  (144, 667)  (122, 756)  (124, 854)  (124, 566)  (128, 871)
(127, 767)
(123, 587)  (135, 691)  (120, 554)  (127, 816)  (132, 606)  (137, 442)
(122, 546)
(139, 884)  (121, 475)  (156, 433)  (156, 839)  (149, 839)  (130, 476)
(122, 646)
经过排序的数据为:
(120, 827)  (120, 554)  (121, 475)  (122, 756)  (122, 546)  (122, 646)
(123, 889)
```

(123, 791) (123, 587) (124, 767) (124, 854) (124, 566) (125, 901)
(127, 767)
(127, 816) (128, 447) (128, 871) (129, 683) (130, 516) (130, 476)
(131, 472)
(131, 617) (131, 718) (131, 799) (132, 606) (134, 552) (134, 485)
(135, 691)
(136, 608) (136, 810) (136, 743) (137, 854) (137, 831) (137, 442)
(138, 639)
(139, 884) (143, 465) (144, 650) (144, 838) (144, 667) (145, 546)
(145, 456)
(146, 690) (146, 833) (147, 750) (148, 876) (148, 756) (149, 839)
(150, 839)
(150, 709) (151, 578) (153, 613) (154, 585) (154, 460) (156, 433)
(156, 839)
写入数据块641
写入数据块642
写入数据块643
写入数据块644
写入数据块645
写入数据块646
写入数据块647
写入数据块648
分别对关系R和关系S完成内排序
****
****
结果块0 写入磁盘下标 700
结果块1 写入磁盘下标 701
结果块0 写入磁盘下标 702
结果块1 写入磁盘下标 703
结果块0 写入磁盘下标 704
结果块1 写入磁盘下标 705
结果块0 写入磁盘下标 706
结果块1 写入磁盘下标 707
结果块0 写入磁盘下标 708
结果块1 写入磁盘下标 709
结果块0 写入磁盘下标 710
结果块1 写入磁盘下标 711
结果块0 写入磁盘下标 712
结果块1 写入磁盘下标 713
结果块0 写入磁盘下标 714
结果块1 写入磁盘下标 715
结果块0 写入磁盘下标 716
结果块1 写入磁盘下标 717
结果块0 写入磁盘下标 718
结果块1 写入磁盘下标 719
结果块0 写入磁盘下标 720
结果块1 写入磁盘下标 721
结果块0 写入磁盘下标 722
结果块1 写入磁盘下标 723
结果块0 写入磁盘下标 724
结果块1 写入磁盘下标 725
结果块0 写入磁盘下标 726
结果块1 写入磁盘下标 727
结果块0 写入磁盘下标 728
结果块1 写入磁盘下标 729

结果块0 写入磁盘下标 730
结果块1 写入磁盘下标 731
结果块0 写入磁盘下标 732
结果块1 写入磁盘下标 733
结果块0 写入磁盘下标 734
结果块1 写入磁盘下标 735
结果块0 写入磁盘下标 736
结果块1 写入磁盘下标 737
结果块0 写入磁盘下标 738
结果块1 写入磁盘下标 739
结果块0 写入磁盘下标 740
结果块1 写入磁盘下标 741
结果块0 写入磁盘下标 742
结果块1 写入磁盘下标 743
结果块0 写入磁盘下标 744
结果块1 写入磁盘下标 745
结果块0 写入磁盘下标 746
结果块1 写入磁盘下标 747
结果块0 写入磁盘下标 748
结果块1 写入磁盘下标 749
结果块0 写入磁盘下标 750
结果块1 写入磁盘下标 751
结果块0 写入磁盘下标 752
结果块1 写入磁盘下标 753
结果块0 写入磁盘下标 754
结果块1 写入磁盘下标 755
结果块0 写入磁盘下标 756
结果块1 写入磁盘下标 757
结果块0 写入磁盘下标 758
结果块1 写入磁盘下标 759
结果块0 写入磁盘下标 760
结果块1 写入磁盘下标 761
结果块0 写入磁盘下标 762
结果块1 写入磁盘下标 763
结果块0 写入磁盘下标 764
结果块1 写入磁盘下标 765
结果块0 写入磁盘下标 766
结果块1 写入磁盘下标 767
结果块0 写入磁盘下标 768
结果块1 写入磁盘下标 769
结果块0 写入磁盘下标 770
结果块1 写入磁盘下标 771
结果块0 写入磁盘下标 772
结果块1 写入磁盘下标 773
结果块0 写入磁盘下标 774
结果块1 写入磁盘下标 775
结果块0 写入磁盘下标 776
结果块1 写入磁盘下标 777
结果块0 写入磁盘下标 778
结果块1 写入磁盘下标 779
结果块0 写入磁盘下标 780
结果块1 写入磁盘下标 781
结果块0 写入磁盘下标 782
结果块1 写入磁盘下标 783
结果块0 写入磁盘下标 784

结果块1 写入磁盘下标 785
结果块0 写入磁盘下标 786
结果块1 写入磁盘下标 787
结果块0 写入磁盘下标 788
结果块1 写入磁盘下标 789
结果块0 写入磁盘下标 790
结果块1 写入磁盘下标 791
结果块0 写入磁盘下标 792
结果块1 写入磁盘下标 793
结果块0 写入磁盘下标 794
结果块1 写入磁盘下标 795
结果块0 写入磁盘下标 796
结果块1 写入磁盘下标 797
结果块0 写入磁盘下标 798
结果块1 写入磁盘下标 799
结果块0 写入磁盘下标 800
结果块1 写入磁盘下标 801
结果块0 写入磁盘下标 802
结果块1 写入磁盘下标 803
结果块0 写入磁盘下标 804
结果块1 写入磁盘下标 805
结果块0 写入磁盘下标 806
结果块1 写入磁盘下标 807
结果块0 写入磁盘下标 808
结果块1 写入磁盘下标 809
结果块0 写入磁盘下标 810
结果块1 写入磁盘下标 811
结果块0 写入磁盘下标 812
结果块1 写入磁盘下标 813
结果块0 写入磁盘下标 814
结果块1 写入磁盘下标 815
结果块0 写入磁盘下标 816
结果块1 写入磁盘下标 817
结果块0 写入磁盘下标 818
结果块1 写入磁盘下标 819
结果块0 写入磁盘下标 820
结果块1 写入磁盘下标 821
结果块0 写入磁盘下标 822
结果块1 写入磁盘下标 823
结果块0 写入磁盘下标 824
结果块1 写入磁盘下标 825
结果块0 写入磁盘下标 826
结果块1 写入磁盘下标 827

关系R的元组读取完毕

结果块0 写入磁盘下标 828
结果块1 写入磁盘下标 829
连接总数为: 389
****

基于排序的两趟扫描算法

- 选择操作：交操作
- 问题分析：任务五相比于任务四，就是在相等的判断条件中进行了微调，从指定列相等改为全相等即可
- 核心代码：

  对关系S的移动，分三种情况讨论

  当前元组小于关系R中所选元组：需要恢复指针位置后移，当前指针位置后移

  当前元组等于关系R中所选元组：写入结果块

  当前元组大于关系R中所选元组：当前分组判断完毕，跳出循环

```
                // 找到小于的元素则继续往后移
                if (cur_tuple[select_col_2] < min_tuple[select_col_1]) {
//                    printf("当前选择数据为(%d, %d)\t 小于所选数据\n",
cur_tuple[0], cur_tuple[1]);
                    // 移动当前指针对应下标
                    cur_index_2[group_index]++;
                    // 移动所需要恢复的下标位置
                    last_group_2_index[group_index]++;
                    // 移动需要恢复的磁盘块号
                    if (last_group_2_index[group_index] % 8 == 0) {
                        last_blk_2[group_index]++;
                    }
                }
                    // 找到相等的元素则写入结果块
                else if (cur_tuple[select_col_2] == min_tuple[select_col_1])
{
                    int other_col_1 = 1 - select_col_1;
                    int other_col_2 = 1 - select_col_2;
                    if (cur_tuple[other_col_2] == min_tuple[other_col_1]) {
                        equal_num++;
                        join_num++;
//                        printf("当前选择数据为(%d, %d)\t 匹配成功\n",
cur_tuple[0], cur_tuple[1]);
//                        printf("当前连接数为: %d\n", join_num);
                        char str_A[4];
                        char str_B[4];
                        char str_C[4];
                        char str_D[4];
                        itoa(min_tuple[0], str_A, 10);
                        itoa(min_tuple[1], str_B, 10);
                        itoa(cur_tuple[0], str_C, 10);
                        itoa(cur_tuple[1], str_D, 10);
                        for (int j = 0; j < 8 - group_1_num - group_2_num;
j++) {
                            // 所有结果块都已经写满
                            if (result_num[j] == 6 && j == 7 - group_1_num -
group_2_num) {
                                for (int k = 0; k < (8 - group_1_num -
group_2_num); k++) {
                                    // 写回当前磁盘块
                                    char str_addr[4];
```

```
                                    itoa(write_disk + 1, str_addr, 10);
                                    for (int m = 0; m < 4; m++) {
                                        *(result_blk[k] + 7 * 8 + m) =
str_addr[m];
                                    }
                                    writeBlockToDisk(result_blk[k],
write_disk, &buf);
                                    printf("结果块%d 写入磁盘下标 %d\n", k,
write_disk);
                                    write_disk++;
                                    // 申请新的磁盘块
                                    result_blk[k] =
getNewBlockInBuffer(&buf);
                                    result_num[k] = 0;
                                }
                            }
                                // 当前块写满
                            else if (result_num[j] == 6 && j < (7 -
group_1_num - group_2_num)) {
                                continue;
                            }
                            for (int k = 0; k < 4; k++) {
                                *(result_blk[j] + result_num[j] * 8 + k) =
str_C[k];
                                *(result_blk[j] + result_num[j] * 8 + 4 + k)
= str_D[k];
                                *(result_blk[j] + result_num[j] * 8 + 8 + k)
= str_A[k];
                                *(result_blk[j] + result_num[j] * 8 + 12 +
k) = str_B[k];
                            }
                            result_num[j] += 2;
                            break;
                        }
                    }
                    cur_index_2[group_index]++;
                }
                    // 如果当前所指元素大于选定的元素则开始判断下一组
                else if (cur_tuple[select_col_2] > min_tuple[select_col_1])
{
//                      printf("当前元组数据大于所选数据，第 %d 组判断完毕\n",
group_index);
                    cur_group_all_found = 1;
                }
```

- 源代码分析：

```
/**
 * 基于排序的两趟扫描算法实现交操作
 * @param start_1 关系1的起始下标
 * @param end_1 关系1的结束下标
 * @param select_col_1 关系1的排序索引列下标
 * @param start_2 关系2的起始下标
```

```
 * @param end_2 关系2的结束下标
 * @param select_col_2 关系2的排序索引列下标
 * @param write_base 写入磁盘块的基底
 */
void intersect(int start_1, int end_1, int select_col_1, int start_2, int
end_2, int select_col_2, int write_base) {
    Buffer buf;
    initBuffer(520, 64, &buf);

    // 对原始数据进行内排序

 printf("**************************************************************************
**********************\n");
    sort_blk_data(start_1, end_1, select_col_1, write_base);
    sort_blk_data(start_2, end_2, select_col_2, write_base);
    printf("分别对关系R和关系S完成内排序\n");

 printf("**************************************************************************
**********************\n");

    // 计算每个属性的组数
    int group_1_num = (end_1 + 1 - start_1) / 8;
    int group_2_num = (end_2 + 1 - start_2) / 8;

    // 指向每个组的指针
    unsigned char *group_1[group_1_num];
    unsigned char *group_2[group_2_num];
    memset(group_1, 0, sizeof group_1);
    memset(group_2, 0, sizeof group_2);

    // 指针指向对应的磁盘块
    for (int i = 0; i < group_1_num; i++) {
        int addr = write_base + start_1 + i * 8;
        group_1[i] = readBlockFromDisk(addr, &buf);
    }
    for (int i = 0; i < group_2_num; i++) {
        int addr = write_base + start_2 + i * 8;
        group_2[i] = readBlockFromDisk(addr, &buf);
    }

    // 定义每个组当前读取到的数据下标
    int cur_index_1[group_1_num];
    int cur_index_2[group_2_num];
    memset(cur_index_1, 0, sizeof cur_index_1);
    memset(cur_index_2, 0, sizeof cur_index_2);

    // 定义第二组当前读取到的磁盘下标
    int cur_blk_2[group_2_num];
    for (int i = 0; i < group_2_num; i++) {
        cur_blk_2[i] = write_base + start_2 + i * 8;
    }

    // 读取完毕的标记变量
    int finish_1[group_1_num];
    int finish_2[group_2_num];
```

```
    memset(finish_1, 0, group_1_num);
    memset(finish_2, 0, group_2_num);

    // 写回磁盘块的首下标
    int write_disk = write_base + 100;
    // 声明结果块
    unsigned char *result_blk[8 - group_1_num - group_2_num];
    for (int i = 0; i < (8 - group_1_num - group_2_num); i++) {
        result_blk[i] = getNewBlockInBuffer(&buf);
    }
    int result_num[8 - group_1_num - group_2_num];
    memset(result_num, 0, sizeof result_num);

    int join_num = 0;
    // 连接操作
    while (1) {
//        printf("*********************************\n");
        /*
         * 找到关系R当前两个指针所指的较小的值
         */
        int min_tuple[2] = {1000, 1000};
        int select_group = -1;
//        printf("选择关系R元组\n");
//        printf("当前选择的两组数据分别为: ");
        for (int group_index = 0; group_index < group_1_num; group_index++)
{
            if (cur_index_1[group_index] == 63) {
//                printf("关系R元组的第 %d 组判断完毕\n", group_index);
                finish_1[group_index] = 1;
                continue;
            }
            // 计算当前下标所对应的块内偏移量
            int offset = cur_index_1[group_index] % 8;
            // 更新当前块指针
            if (offset == 7) {
                // 计算下一块下标
                char str_addr[4];
                for (int k = 0; k < 4; k++) {
                    str_addr[k] = *(group_1[group_index] + offset * 8 + k);
                }
                int next_disk = write_base + atoi(str_addr);
                // 释放原有磁盘块
                freeBlockInBuffer(group_1[group_index], &buf);
                // 读取新的磁盘块
                group_1[group_index] = readBlockFromDisk(next_disk, &buf);
                cur_index_1[group_index]++;
                // 更新offset
                offset = cur_index_1[group_index] % 8;
            }
            // 记录数据
            char str_1[4];
            char str_2[4];
            for (int k = 0; k < 4; k++) {
                str_1[k] = *(group_1[group_index] + offset * 8 + k);
                str_2[k] = *(group_1[group_index] + offset * 8 + 4 + k);
```

```
            }
            int cur_tuple[2] = {atoi(str_1), atoi(str_2)};
//            printf("(%d, %d)\t", cur_tuple[0], cur_tuple[1]);
            // 数据更小则更新
            if (cur_tuple[select_col_1] <= min_tuple[select_col_1]) {
                min_tuple[0] = cur_tuple[0];
                min_tuple[1] = cur_tuple[1];
                select_group = group_index;
            }
        }
//        printf("\n");
//        printf("最终选择数据为: (%d, %d)\n", min_tuple[0], min_tuple[1]);
//        printf("两组当前下标分别为:\n");
//        printf("group 0: %d, group 1: %d\n", cur_index_1[0],
cur_index_1[1]);
        // 更新下标
        cur_index_1[select_group]++;

        /*
         * 判断关系R是否完全读取完毕
         */
        int finish_num = 0;
        while (finish_num < group_1_num && (finish_1[finish_num] == 1)) {
            finish_num++;
        }
        if (finish_num == group_1_num) {
            break;
        }

        /*
         * 找到关系S中与关系R选出的值相等的元组
         */
//        printf("选择关系S元组\n");
        int last_group_2_index[group_2_num];
        int last_blk_2[group_2_num];

        for (int i = 0; i < group_2_num; i++) {
            last_group_2_index[i] = cur_index_2[i];
        }
        for (int i = 0; i < group_2_num; i++) {
            last_blk_2[i] = cur_blk_2[i];
        }

        int equal_num = 0;
        for (int group_index = 0; group_index < group_2_num; group_index++)
{
            // 如果当前组已经读取完毕
            if (cur_index_2[group_index] == 63) {
                finish_2[group_index] = 1;
                continue;
            }
            // 当前组所有符合条件元素都已经选择完毕的标记变量
            int cur_group_all_found = 0;
            while (!cur_group_all_found) {
                // 计算当前下标所对应的块内偏移量
```

```
                int offset = cur_index_2[group_index] % 8;
                // 更新当前块指针
                if (offset == 7) {
                    char str_addr[4];
                    for (int k = 0; k < 4; k++) {
                        str_addr[k] = *(group_2[group_index] + offset * 8 +
k);
                    }
                    int next_disk = write_base + atoi(str_addr);
                    // 释放原有磁盘块
                    freeBlockInBuffer(group_2[group_index], &buf);
                    // 读取新的磁盘块
                    group_2[group_index] = readBlockFromDisk(next_disk,
&buf);
                    cur_index_2[group_index]++;
                    // 更新offset
                    offset = cur_index_2[group_index] % 8;
                    // 更新磁盘块下标
                    cur_blk_2[group_index] = write_base + start_2 +
group_index * 8 + cur_index_2[group_index] / 8;
                }
                // 记录数据
                char str_1[4];
                char str_2[4];
                read_from_blk(group_2[group_index], offset, str_1, str_2);
                int cur_tuple[2] = {atoi(str_1), atoi(str_2)};
//                printf("当前组号为: %d\t 块号为: %d\t 偏移量为: %d\n",
group_index, cur_blk_2[group_index], cur_index_2[group_index]);
//                printf("当前数据为: (%d, %d)\n", cur_tuple[0],
cur_tuple[1]);
                // 判断结果块是否写满的标记变量
                int all_full = 0;
                // 找到小于的元素则继续往后移
                if (cur_tuple[select_col_2] < min_tuple[select_col_1]) {
//                    printf("当前选择数据为(%d, %d)\t 小于所选数据\n",
cur_tuple[0], cur_tuple[1]);
                    // 移动当前指针对应下标
                    cur_index_2[group_index]++;
                    // 移动所需要恢复的下标位置
                    last_group_2_index[group_index]++;
                    // 移动需要恢复的磁盘块号
                    if (last_group_2_index[group_index] % 8 == 0) {
                        last_blk_2[group_index]++;
                    }
                }
                // 找到相等的元素则写入结果块
                else if (cur_tuple[select_col_2] == min_tuple[select_col_1])
{
                    int other_col_1 = 1 - select_col_1;
                    int other_col_2 = 1 - select_col_2;
                    if (cur_tuple[other_col_2] == min_tuple[other_col_1]) {
                        equal_num++;
                        join_num++;
//                        printf("当前选择数据为(%d, %d)\t 匹配成功\n",
cur_tuple[0], cur_tuple[1]);
```

```
//                          printf("当前连接数为: %d\n", join_num);
                        char str_A[4];
                        char str_B[4];
                        char str_C[4];
                        char str_D[4];
                        itoa(min_tuple[0], str_A, 10);
                        itoa(min_tuple[1], str_B, 10);
                        itoa(cur_tuple[0], str_C, 10);
                        itoa(cur_tuple[1], str_D, 10);
                        for (int j = 0; j < 8 - group_1_num - group_2_num;
j++) {
                            // 所有结果块都已经写满
                            if (result_num[j] == 6 && j == 7 - group_1_num -
group_2_num) {
                                for (int k = 0; k < (8 - group_1_num -
group_2_num); k++) {
                                    // 写回当前磁盘块
                                    char str_addr[4];
                                    itoa(write_disk + 1, str_addr, 10);
                                    for (int m = 0; m < 4; m++) {
                                        *(result_blk[k] + 7 * 8 + m) =
str_addr[m];
                                    }
                                    writeBlockToDisk(result_blk[k],
write_disk, &buf);
                                    printf("结果块%d 写入磁盘下标 %d\n", k,
write_disk);
                                    write_disk++;
                                    // 申请新的磁盘块
                                    result_blk[k] =
getNewBlockInBuffer(&buf);
                                    result_num[k] = 0;
                                }
                            }
                                // 当前块写满
                            else if (result_num[j] == 6 && j < (7 -
group_1_num - group_2_num)) {
                                continue;
                            }
                            for (int k = 0; k < 4; k++) {
                                *(result_blk[j] + result_num[j] * 8 + k) =
str_C[k];
                                *(result_blk[j] + result_num[j] * 8 + 4 + k)
= str_D[k];
                                *(result_blk[j] + result_num[j] * 8 + 8 + k)
= str_A[k];
                                *(result_blk[j] + result_num[j] * 8 + 12 +
k) = str_B[k];
                            }
                            result_num[j] += 2;
                            break;
                        }
                    }
                    cur_index_2[group_index]++;
                }
```

```
                // 如果当前所指元素大于选定的元素则开始判断下一组
                else if (cur_tuple[select_col_2] > min_tuple[select_col_1])
{
//                    printf("当前元组数据大于所选数据，第 %d 组判断完毕\n",
group_index);
                    cur_group_all_found = 1;
                }
            }
        }
//        printf("每个组需要恢复到的磁盘块号为: ");
//        for (int i = 0; i < group_2_num; i++) {
//            printf("%d\t", last_blk_2[i]);
//        }
//        printf("\n");
        // 还原每个指针读取的磁盘块下标
        for (int i = 0; i < group_2_num; i++) {
            cur_blk_2[i] = last_blk_2[i];
            freeBlockInBuffer(group_2[i], &buf);
            group_2[i] = readBlockFromDisk(cur_blk_2[i], &buf);
        }
//        printf("每个组需要恢复到的下标标号为: ");
//        for (int i = 0; i < group_2_num; i++) {
//            printf("%d\t", last_group_2_index[i]);
//        }
//        printf("\n");
        // 还原关系S对应组的下标
        for (int i = 0; i < group_2_num; i++) {
            cur_index_2[i] = last_group_2_index[i];
        }
//        printf("当前数据(%d, %d) 交总数为: %d\n", min_tuple[0], min_tuple[1],
equal_num);
//        printf("****************************");
    }
    // 最后数据未写满时也要写入结果
    int result_blk_num = join_num / 3;
    int last_blk_num = join_num % 3;
    for (int k = 0; k < (8 - group_1_num - group_2_num); k++) {
        // 最后一块
        if (write_disk == write_base + 100 + result_blk_num) {
            // 没写满需要删去部分内容
            for (int i = 2 * last_blk_num; i < 7; i++) {
                for (int m = 0; m < 4; m++) {
                    *(result_blk[k] + i * 8 + m) = 0;
                    *(result_blk[k] + i * 8 + 4 + m) = 0;
                }
            }
        }
            // 多余的块不写
        else if (write_disk > write_base + 100 + result_blk_num) {
            continue;
        }
        // 写回当前磁盘块
        char str_addr[4];
        itoa(write_disk + 1, str_addr, 10);
        for (int m = 0; m < 4; m++) {
```

```
            *(result_blk[k] + 7 * 8 + m) = str_addr[m];
        }
        writeBlockToDisk(result_blk[k], write_disk, &buf);
        printf("结果块%d 写入磁盘下标 %d\n", k, write_disk);
        write_disk++;
        // 申请新的磁盘块
        result_blk[k] = getNewBlockInBuffer(&buf);
        result_num[k] = 0;
    }
    printf("交总数为: %d\n", join_num);
}
```

- 运行结果:

```
任务五:
**************************************************************************************
************
**************************************************************************************
************
第 1 轮内排序:
原始数据为:
(131, 479)  (123, 477)  (107, 411)  (139, 476)  (106, 461)  (119, 406)
(105, 497)
(137, 473)  (137, 451)  (134, 459)  (138, 411)  (100, 421)  (109, 472)
(135, 441)
(133, 455)  (130, 436)  (121, 438)  (124, 468)  (117, 438)  (123, 422)
(131, 454)
(109, 404)  (139, 461)  (116, 421)  (122, 474)  (138, 497)  (114, 410)
(118, 414)
(102, 465)  (117, 475)  (137, 420)  (108, 436)  (120, 418)  (101, 410)
(101, 406)
(105, 428)  (109, 409)  (101, 470)  (124, 412)  (104, 450)  (119, 428)
(107, 477)
(115, 401)  (129, 455)  (133, 467)  (105, 476)  (137, 428)  (114, 425)
(102, 413)
(120, 827)  (122, 546)  (125, 886)  (127, 767)  (128, 447)  (130, 656)
(140, 610)
经过排序的数据为:
(100, 421)  (101, 410)  (101, 406)  (101, 470)  (102, 465)  (102, 413)
(104, 450)
(105, 497)  (105, 428)  (105, 476)  (106, 461)  (107, 411)  (107, 477)
(108, 436)
(109, 472)  (109, 404)  (109, 409)  (114, 410)  (114, 425)  (115, 401)
(116, 421)
(117, 438)  (117, 475)  (118, 414)  (119, 406)  (119, 428)  (120, 418)
(120, 827)
(121, 438)  (122, 474)  (122, 546)  (123, 477)  (123, 422)  (124, 468)
(124, 412)
(125, 886)  (127, 767)  (128, 447)  (129, 455)  (130, 436)  (130, 656)
(131, 479)
(131, 454)  (133, 455)  (133, 467)  (134, 459)  (135, 441)  (137, 473)
(137, 451)
(137, 420)  (137, 428)  (138, 411)  (138, 497)  (139, 476)  (139, 461)
(140, 610)
写入数据块901
```

```
写入数据块902
写入数据块903
写入数据块904
写入数据块905
写入数据块906
写入数据块907
写入数据块908
**********
**********
第 2 轮内排序:
原始数据为:
(137, 471)  (120, 499)  (133, 441)  (126, 423)  (129, 488)  (116, 470)
(124, 410)
(100, 439)  (101, 405)  (138, 468)  (123, 452)  (117, 442)  (104, 440)
(133, 428)
(132, 483)  (116, 452)  (110, 405)  (138, 475)  (128, 453)  (124, 426)
(116, 414)
(102, 492)  (129, 475)  (129, 402)  (131, 492)  (101, 409)  (114, 414)
(107, 434)
(121, 464)  (116, 424)  (130, 411)  (110, 413)  (128, 459)  (129, 430)
(117, 403)
(100, 400)  (117, 426)  (110, 491)  (138, 495)  (138, 407)  (130, 495)
(137, 480)
(126, 485)  (134, 486)  (108, 482)  (119, 431)  (111, 445)  (112, 467)
(117, 412)
(124, 499)  (130, 417)  (118, 478)  (116, 420)  (132, 422)  (110, 450)
(134, 437)
经过排序的数据为:
(100, 439)  (100, 400)  (101, 405)  (101, 409)  (102, 492)  (104, 440)
(107, 434)
(108, 482)  (110, 405)  (110, 413)  (110, 491)  (110, 450)  (111, 445)
(112, 467)
(114, 414)  (116, 470)  (116, 452)  (116, 414)  (116, 424)  (116, 420)
(117, 442)
(117, 403)  (117, 426)  (117, 412)  (118, 478)  (119, 431)  (120, 499)
(121, 464)
(123, 452)  (124, 410)  (124, 426)  (124, 499)  (126, 423)  (126, 485)
(128, 453)
(128, 459)  (129, 488)  (129, 475)  (129, 402)  (129, 430)  (130, 411)
(130, 495)
(130, 417)  (131, 492)  (132, 483)  (132, 422)  (133, 441)  (133, 428)
(134, 486)
(134, 437)  (137, 471)  (137, 480)  (138, 468)  (138, 475)  (138, 495)
(138, 407)
写入数据块909
写入数据块910
写入数据块911
写入数据块912
写入数据块913
写入数据块914
写入数据块915
写入数据块916
**********
**********
第 1 轮内排序:
```

原始数据为:
(124, 605) (146, 445) (123, 532) (124, 844) (141, 754) (135, 919)
(136, 454)
(123, 477) (129, 486) (130, 436) (134, 437) (139, 461) (129, 430)
(120, 418)
(121, 739) (130, 656) (155, 544) (127, 680) (145, 805) (123, 794)
(156, 600)
(151, 704) (139, 878) (133, 456) (129, 591) (124, 424) (132, 640)
(154, 728)
(159, 630) (137, 733) (139, 576) (141, 732) (154, 810) (148, 570)
(140, 610)
(135, 671) (125, 587) (137, 912) (134, 457) (125, 886) (146, 721)
(157, 456)
(149, 873) (128, 684) (152, 717) (155, 600) (128, 431) (148, 854)
(120, 736)
(130, 790) (156, 809) (129, 492) (120, 581) (125, 725) (159, 447)
(157, 902)
经过排序的数据为:
(120, 418) (120, 736) (120, 581) (121, 739) (123, 532) (123, 477)
(123, 794)
(124, 605) (124, 844) (124, 424) (125, 587) (125, 886) (125, 725)
(127, 680)
(128, 684) (128, 431) (129, 486) (129, 430) (129, 591) (129, 492)
(130, 436)
(130, 656) (130, 790) (132, 640) (133, 456) (134, 437) (134, 457)
(135, 919)
(135, 671) (136, 454) (137, 733) (137, 912) (139, 461) (139, 878)
(139, 576)
(140, 610) (141, 754) (141, 732) (145, 805) (146, 445) (146, 721)
(148, 570)
(148, 854) (149, 873) (151, 704) (152, 717) (154, 728) (154, 810)
(155, 544)
(155, 600) (156, 600) (156, 809) (157, 456) (157, 902) (159, 630)
(159, 447)
写入数据块917
写入数据块918
写入数据块919
写入数据块920
写入数据块921
写入数据块922
写入数据块923
写入数据块924
****
****
第 2 轮内排序:
原始数据为:
(152, 634) (122, 554) (134, 494) (128, 615) (120, 775) (148, 616)
(128, 429)
(142, 718) (135, 542) (121, 793) (137, 460) (137, 895) (141, 636)
(148, 635)
(120, 571) (148, 547) (146, 843) (156, 479) (137, 585) (145, 625)
(157, 498)
(123, 733) (156, 698) (156, 482) (158, 764) (144, 609) (147, 663)
(142, 450)

```
(156, 857)  (142, 748)  (152, 826)  (147, 561)  (136, 711)  (139, 802)
(125, 496)
(140, 551)  (136, 434)  (153, 642)  (146, 834)  (158, 432)  (141, 629)
(128, 584)
(139, 488)  (145, 689)  (141, 504)  (122, 682)  (151, 444)  (129, 844)
(122, 760)
(150, 835)  (138, 512)  (120, 852)  (145, 521)  (134, 883)  (155, 668)
(139, 815)
经过排序的数据为:
(120, 775)  (120, 571)  (120, 852)  (121, 793)  (122, 554)  (122, 682)
(122, 760)
(123, 733)  (125, 496)  (128, 615)  (128, 429)  (128, 584)  (129, 844)
(134, 494)
(134, 883)  (135, 542)  (136, 711)  (136, 434)  (137, 460)  (137, 895)
(137, 585)
(138, 512)  (139, 802)  (139, 488)  (139, 815)  (140, 551)  (141, 636)
(141, 629)
(141, 504)  (142, 718)  (142, 450)  (142, 748)  (144, 609)  (145, 625)
(145, 689)
(145, 521)  (146, 843)  (146, 834)  (147, 663)  (147, 561)  (148, 616)
(148, 635)
(148, 547)  (150, 835)  (151, 444)  (152, 634)  (152, 826)  (153, 642)
(155, 668)
(156, 479)  (156, 698)  (156, 482)  (156, 857)  (157, 498)  (158, 764)
(158, 432)
写入数据块925
写入数据块926
写入数据块927
写入数据块928
写入数据块929
写入数据块930
写入数据块931
写入数据块932
**********************************************************************
**********
第 3 轮内排序:
原始数据为:
(140, 915)  (145, 570)  (159, 737)  (129, 535)  (153, 456)  (121, 596)
(125, 625)
(151, 457)  (128, 592)  (126, 774)  (128, 457)  (126, 685)  (121, 582)
(145, 540)
(152, 573)  (131, 808)  (147, 552)  (130, 766)  (128, 720)  (151, 644)
(142, 679)
(146, 463)  (153, 573)  (157, 691)  (138, 700)  (130, 556)  (150, 890)
(134, 900)
(134, 756)  (130, 474)  (159, 585)  (120, 781)  (154, 593)  (155, 916)
(135, 747)
(154, 528)  (149, 523)  (158, 602)  (149, 763)  (122, 463)  (137, 660)
(155, 464)
(142, 841)  (127, 743)  (145, 840)  (131, 892)  (138, 816)  (143, 732)
(138, 674)
(125, 808)  (138, 598)  (123, 468)  (151, 461)  (149, 573)  (153, 641)
(126, 919)
经过排序的数据为:
```

(120, 781) (121, 596) (121, 582) (122, 463) (123, 468) (125, 625)
(125, 808)
(126, 774) (126, 685) (126, 919) (127, 743) (128, 592) (128, 457)
(128, 720)
(129, 535) (130, 766) (130, 556) (130, 474) (131, 808) (131, 892)
(134, 900)
(134, 756) (135, 747) (137, 660) (138, 700) (138, 816) (138, 674)
(138, 598)
(140, 915) (142, 679) (142, 841) (143, 732) (145, 570) (145, 540)
(145, 840)
(146, 463) (147, 552) (149, 523) (149, 763) (149, 573) (150, 890)
(151, 457)
(151, 644) (151, 461) (152, 573) (153, 456) (153, 573) (153, 641)
(154, 593)
(154, 528) (155, 916) (155, 464) (157, 691) (158, 602) (159, 737)
(159, 585)
写入数据块933
写入数据块934
写入数据块935
写入数据块936
写入数据块937
写入数据块938
写入数据块939
写入数据块940
****
****
第 4 轮内排序:
原始数据为:
(131, 472) (120, 827) (136, 608) (146, 690) (147, 750) (145, 546)
(151, 578)
(123, 889) (137, 854) (136, 810) (123, 791) (137, 831) (154, 585)
(134, 552)
(131, 617) (146, 833) (129, 683) (130, 516) (154, 460) (150, 839)
(131, 718)
(143, 465) (144, 650) (138, 639) (136, 743) (131, 799) (153, 613)
(134, 485)
(144, 838) (145, 456) (148, 876) (150, 709) (124, 767) (128, 447)
(125, 901)
(148, 756) (144, 667) (122, 756) (124, 854) (124, 566) (128, 871)
(127, 767)
(123, 587) (135, 691) (120, 554) (127, 816) (132, 606) (137, 442)
(122, 546)
(139, 884) (121, 475) (156, 433) (156, 839) (149, 839) (130, 476)
(122, 646)
经过排序的数据为:
(120, 827) (120, 554) (121, 475) (122, 756) (122, 546) (122, 646)
(123, 889)
(123, 791) (123, 587) (124, 767) (124, 854) (124, 566) (125, 901)
(127, 767)
(127, 816) (128, 447) (128, 871) (129, 683) (130, 516) (130, 476)
(131, 472)
(131, 617) (131, 718) (131, 799) (132, 606) (134, 552) (134, 485)
(135, 691)
(136, 608) (136, 810) (136, 743) (137, 854) (137, 831) (137, 442)
(138, 639)

```
(139, 884)  (143, 465)  (144, 650)  (144, 838)  (144, 667)  (145, 546)
(145, 456)
(146, 690)  (146, 833)  (147, 750)  (148, 876)  (148, 756)  (149, 839)
(150, 839)
(150, 709)  (151, 578)  (153, 613)  (154, 585)  (154, 460)  (156, 433)
(156, 839)
写入数据块941
写入数据块942
写入数据块943
写入数据块944
写入数据块945
写入数据块946
写入数据块947
写入数据块948
分别对关系R和关系S完成内排序
********************************************************************************
************
结果块0 写入磁盘下标 1000
结果块1 写入磁盘下标 1001
结果块0 写入磁盘下标 1002
结果块1 写入磁盘下标 1003
结果块0 写入磁盘下标 1004
交总数为: 13
********************************************************************************
************
```

总结

本次实验五从头到尾完成时间大约耗时一个星期。

这个实验第一个让我学到的东西是：耐心。因为C语言中使用到指针的相关代码往往容易出现指针不知道跑到哪里去的情况，导致读取出来的数据与数据块中的数据不对应，这就需要我一步一步单步执行的进行调试，这个过程耐心非常重要。

第二个让我学习到的知识是：内存调试窗口。由于对指针调试的不熟悉，导致一开始在调试指针的过程中总会出现奇奇怪怪的错误，也不知道出错在哪里。在询问了老师之后学习到了内存调试这个功能，可以清楚的看到指针所指的内存块和内存块中存放的内容，这让我的调试过程变得顺利了不少。

第三个让我学习到的知识：指针用法的深入了解。上一次这么大规模的使用C语言进行程序的编写应该还是在大一的时候，但那个时候对于指针这个比较深奥难懂的概念其实并没有很了解。而大二和大三的过程中学习到的实验一般都会使用Java或Python等语言进行编写，这类语言没有指针这一概念，因此对于指针的使用一直都是一知半解。而这次实验让我再次走进C语言的世界和指针进行深入交流，让我对指针的用法有了更好的掌握。

最后，这是数据库课程的最后一次实验，总体而言，本学期的数据库课程实验设计难度适中，而且在新冠疫情的情况下老师还非常人性化的延长了相关实验的截止日期，在家期间每次询问老师都能得到耐心、专业的解答，再次感谢实验老师一学期的辛苦付出！