哈尔滨工业大学(深圳)

《数据库》实验报告

实验五

查询处理算法的模拟实现

学　院: 计算机科学与技术

姓　名: 梁鑫嵘

学　号: 200110619

专　业: 计算机科学与技术

日　期: 2023-01-03

一、 实验目的

阐述本次实验的目的。

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。

二、 实验环境

阐述本次实验的环境。

系统：Arch Linux / Windows 11

软件：CLion, GCC, MingW-w64, CodeBlocks

三、 实验内容

阐述本次实验的具体内容。

磁盘上储存着关系 R 与 S，其中各有两个属性。基于这些数据完成：

1. 基于线性搜索的关系选择算法
2. 两阶段多路归并排序算法(TPMMS)
3. 基于索引的关系选择算法
4. 基于排序的连接操作算法(Sort-Merge-Join)
5. 基于排序或散列的两趟扫描算法

四、 实验过程

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

在本次实验中实现了一些基础设施，例如：

- lambda 函数：用于构建迭代器（请忽略相关的编辑器报错）
- 迭代器 iterator：用于迭代磁盘或内存中的缓冲区队列内容
- 缓冲区队列 buffered_queue：用于线性写入磁盘或管理缓冲区内容
- 磁盘缓存 cache：用于利用缓冲区减少磁盘 IO

以下代码等均一定程度上依赖上述基础设施。

(1) 实现基于线性搜索的关系选择算法

问题分析：线性搜索，即从头到尾遍历一次，遍历过程中找到需要选择出的数据，同时将找到的数据写入磁盘。为此，可以建立写入队列，然后使用磁盘迭代器对磁盘对应块上的数据项目进行迭代，比对每次得到的数据是否符合要求，如果符合要求则通过写入队列写入缓冲区再写入磁盘。

```
buffer_init();
uint target = 100;
uint count = 0;
buffered_queue *q = buffered_queue_init(1, target, true);
iterate_range(17, 49, lambda(bool, (char* c) {
  if (SEQ(c, "128")) {
    buffered_queue_push(q, c);
    count++;
  }
  return *c != '\0';
}));
buffered_queue_flush(q);
buffered_queue_free(q);
buffer_report_msg("线性搜索");
Log("选择结果如下，储存于地址 %d", target);
iterate_range_show(target, -1);
Log("满足选择条件的元组一共 %d 个", count);
buffer_free();
}
```

图为核心代码，见 question1.c。

实验结果：

```
[/home/chiro/programs/database-lab/lab5/question1.c:9 q1] ==========================
[/home/chiro/programs/database-lab/lab5/question1.c:10 q1] Q1: 基于线性搜索的关系选择算法
[/home/chiro/programs/database-lab/lab5/question1.c:11 q1] select S.C,S.D from S where S.C = 128
[/home/chiro/programs/database-lab/lab5/question1.c:12 q1] ==========================
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 线性搜索: IO 读写一共 35 次
[/home/chiro/programs/database-lab/lab5/question1.c:29 q1] 选择结果如下，储存于地址 100
======== data [100, -1):
[100] (128, 684) (128, 431) (128, 615) (128, 429) (128, 584) (128, 592) (128, 457)
[101] (128, 720) (128, 447) (128, 871)
[/home/chiro/programs/database-lab/lab5/question1.c:31 q1] 满足选择条件的元组一共 10 个
```

(2) 实现两阶段多路归并排序算法（TPMMS）

问题分析：为实现 TPMMS 算法，首先需要将数据分组并在组内排序。

由于内存中最多储存 BLK=8 个缓存块，所以每个组最大为 8 个块，分块代码如下：

```
void TPMMS_sort_subsets(uint left, uint right, uint target) {
  uint blk_total = right - left;
  uint rounds = blk_total / BLK +
                ((blk_total % BLK) == 0 ? 0 : 1);
  for (uint r = 0; r < rounds; r++)
    TPMMS_sort_subset(
        left + r * BLK,
        left + (r + 1) * BLK,
        target + r * BLK,
        r != rounds - 1);
}
```

块内排序的实现方法是加载至多 BLK 个块插入缓冲区队列 buffered_queue 中，然后在缓冲区队列中进行内存中的内排序。内排序算法使用了简单的冒泡排序。

```
void TPMMS_sort_subset(uint left, uint right, uint target, bool continuous) {
  buffered_queue *q = buffered_queue_init(BLK, target, false);
  // 加载数据到 queue
  for (uint addr = left; addr < right; addr++)
    buffered_queue_push_blk(q, addr, continuous);
  // 内排序
  buffered_queue_sort(q, 0);
  // 将内排序结果写回磁盘
  q->flushable = true;
  buffered_queue_flush(q);
  buffered_queue_free(q);
}
```

每个数据分组排序完成后在磁盘上得到了 rounds 个有序的列表，接下来对这 rounds 个有序列表进行归并排序。这里需要申请 rounds 个对应地址范围的迭代器，并每次选择数据最小的一个迭代器插入写地址队列。由于每个迭代器都会占用一个缓冲区块，而且写地址队列也会占用一个缓冲区块，所以最多每次合并 BLK-1 个有序队列。鉴于数据量大小，可以假定 rounds<BLK-1。

```
void TPMMS_merge_sort(uint left, uint right, uint target) {
  Dbg("TPMMS_merge_sort");
  uint blk_total = right - left;
  uint reader_count = blk_total / BLK +
                      ((blk_total % BLK) == 0 ? 0 : 1);
  Assert(reader_count < BLK, "merge sort cannot have readers more than %d", BLK - 1);
  iterator *readers[BLK - 1] = {NULL};
  for (int i = 0; i < reader_count; i++)
    readers[i] = iterator_init(left + i * BLK, left + min((i + 1) * BLK, right), NULL);
  buffered_queue *q = buffered_queue_init(1, target, true);
  iterator *reader;
  // 选择多个读头中最小的一个
  while ((reader = TPMM_reader_select(readers)) != NULL) {
    buffered_queue_push(q, iterator_now(reader));
    iterator_next(reader);
  }
  for (int i = 0; i < reader_count; i++)
    iterator_free(readers[i]);
  buffered_queue_flush(q);
  buffered_queue_free(q);
}
```

归并排序结束，TPMMS 算法也就结束了。算法执行代码如下：

```
void TPMMS(uint left, uint right, uint target) {
    uint temp = 400;
    TPMMS_sort_subsets(left, right, temp);
    TPMMS_merge_sort(temp, temp + (right - left), target);
}

void q2() {
    Log("============================");
    Log("Q2: 两阶段多路归并排序算法(TPMMS)");
    Log("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。");
    Log("============================");
    buffer_init();
    TPMMS(1, 17, 301);
    TPMMS(17, 49, 317);
    buffer_report_msg("TPMMS");
    Log("R 排序后如下，储存于地址 [301, 316]");
    iterate_range_show_some(301, 317);
    Log("S 排序后如下，储存于地址 [317, 348]");
    iterate_range_show_some(317, 349);
    buffer_free();
}
```

实验结果：

```
[/home/chiro/programs/database-lab/lab5/question2.c:95 q2] ============================
[/home/chiro/programs/database-lab/lab5/question2.c:96 q2] Q2: 两阶段多路归并排序算法(TPMMS)
[/home/chiro/programs/database-lab/lab5/question2.c:97 q2] 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。
[/home/chiro/programs/database-lab/lab5/question2.c:98 q2] ============================
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] TPMMS: IO 读写一共 194 次
[/home/chiro/programs/database-lab/lab5/question2.c:103 q2] R 排序后如下，储存于地址 [301, 316]
======== data [301, 317):
[301] (100, 400) (100, 421) (100, 439) (101, 405) (101, 406) (101, 409) (101, 410)
[302] (101, 470) (102, 413) (102, 465) (102, 492) (104, 440) (104, 450) (105, 428)
    ... ...
[315] (137, 428) (137, 451) (137, 471) (137, 473) (137, 480) (138, 407) (138, 411)
[316] (138, 468) (138, 475) (138, 495) (138, 497) (139, 461) (139, 476) (140, 610)
[/home/chiro/programs/database-lab/lab5/question2.c:105 q2] S 排序后如下，储存于地址 [317, 348]
======== data [317, 349):
[317] (120, 418) (120, 554) (120, 571) (120, 581) (120, 736) (120, 775) (120, 781)
[318] (120, 827) (120, 852) (121, 475) (121, 582) (121, 596) (121, 739) (121, 793)
    ... ...
[347] (156, 809) (156, 839) (156, 857) (157, 456) (157, 498) (157, 691) (157, 902)
[348] (158, 432) (158, 602) (158, 764) (159, 447) (159, 585) (159, 630) (159, 737)
```

限于图片大小，只展示了排序后数据的开头与结尾，具体请查看对应文件。

(3) 实现基于索引的关系选择算法

问题分析：问题的关键在如何实现索引以及索引的储存。这里实现的索引格

式是元组(key, addr)，key 为索引键，addr 为其出现在哪一个磁盘块。读取索引后迭代这些索引，找到 key 和所求对应的 key 相等的元组，再依据这个元组指向的磁盘块地址取到磁盘上有序列表中的某一块，最后从这个块中所求 key 开始迭代，一直迭代到键不为所求的 key。把迭代过程中找到的结果写入写入缓冲区队列。以下为如何建立索引：

```
void create_index_range(uint left, uint right, uint addr) {
  buffered_queue *q = buffered_queue_init(1, addr, true);
  uint cnt = 0;
  uint index = left;
  // storage primary index to secondary place
  // I = (key, addr), addr is where key first appears
  uint key_last = -1;
  iterate_range(left, right, lambda(bool, (char *s) {
    uint key = atoi3(s);
    if (key != key_last) {
      if (key_last != -1) Assert(key > key_last, "data not in order!");
      Dbg("push index (key=%s, addr=%d)", s, index);
      buffered_queue_push(q, itot(key, index));
      key_last = key;
    }
    if ((++cnt) == 7) {
      cnt = 0;
      index++;
    }
    return true;
  }));
  buffered_queue_flush(q);
  buffered_queue_free(q);
}
```

建立索引后如何读取索引以及如何根据索引遍历磁盘块上的内容：

```
void indexed_select_linear(uint left, uint right, uint key, buffered_queue *target) {
  Dbg("indexed_select_linear(%d, %d, key=%d)", left, right, key);
  uint target_addr = -1;
  iterate_range(left, right, lambda(bool, (char *s) {
    uint k = atoi3(s);
    if (k == key) {
      target_addr = atoi3(s + 4);
      Dbg("got target addr: %d", target_addr);
    }
    return target_addr == -1;
  }));
  Assert(target_addr != -1, "cannot find key %d", key);
  bool read_started = false;
  iterate_range(target_addr, -1, lambda(bool, (char *s) {
    uint k = atoi3(s);
    if (!read_started) {
      if (k == key) {
        buffered_queue_push(target, s);
        read_started = true;
      }
      return true;
    } else {
      if (k == key) {
        buffered_queue_push(target, s);
      }
      return k == key;
    }
  }));
}
```

总的实验流程：

```
void q3() {
    Log("============================");
    Log("Q3: 基于索引的关系选择算法");
    Log("排序结果为关系R或S分别建立索引文件，");
    Log("模拟实现 select S.C, S.D from S where S.C = 128");
    Log("============================");
    buffer_init();
    Log("正在排序...");
    TPMMS(1, 17, 301);
    TPMMS(17, 49, 317);
    buffer_report_msg("排序过程");
    buffer_free();
    Log("正在建立索引...");
    buffer_init();
    create_index_range(301, 317, 501);
    create_index_range(317, 349, 517);
    buffer_report_msg("建立索引过程");
    buffer_free();

    buffer_init();
    Log("索引文件位于 [501...], [517...]");
    buffered_queue *q = buffered_queue_init(1, 600, true);
    indexed_select_linear(517, -1, 128, q);
    buffered_queue_flush(q);
    buffered_queue_free(q);
    buffer_report_msg("检索过程");
    q = buffered_queue_init(4, -1, false);
    buffered_queue_load_from(q, 600, -1);
    Log("满足选择条件的元组一共 %d 个，如下", buffered_queue_count(q));
    buffered_queue_show(q);
    buffered_queue_free(q);
    buffer_free();
}
```

实验结果：

```
[/home/chiro/programs/database-lab/lab5/question3.c:66 q3] ============================
[/home/chiro/programs/database-lab/lab5/question3.c:67 q3] Q3: 基于索引的关系选择算法
[/home/chiro/programs/database-lab/lab5/question3.c:68 q3] 排序结果为关系R或S分别建立索引文件,
[/home/chiro/programs/database-lab/lab5/question3.c:69 q3] 模拟实现 select S.C, S.D from S where S.C = 128
[/home/chiro/programs/database-lab/lab5/question3.c:70 q3] ============================
[/home/chiro/programs/database-lab/lab5/question3.c:72 q3] 正在排序...
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 排序过程: IO 读写一共 194 次
[/home/chiro/programs/database-lab/lab5/question3.c:77 q3] 正在建立索引...
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 建立索引过程: IO 读写一共 62 次
[/home/chiro/programs/database-lab/lab5/question3.c:85 q3] 索引文件位于 [501...], [517...]
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 检索过程: IO 读写一共 7 次
[/home/chiro/programs/database-lab/lab5/question3.c:93 q3] 满足选择条件的元组一共 10 个，如下
======== data in queue (total=4, size=1, offset/8=3, addr=-1)
(128, 429) (128, 431) (128, 447) (128, 457) (128, 584) (128, 592) (128, 615)
(128, 684) (128, 720) (128, 871)
```

其仅用 7 次 IO 就将数据找出并写入磁盘，与第一个问题实现的基于线性搜索的关系选择算法中的 35 次相比有巨大的提升。

(4) 实现基于排序的连接操作算法（Sort-Merge-Join）

问题分析：首先对两个源数据区域进行排序，然后使用 Sort-Merge-Join 算法同时迭代两个迭代器，将比对得到满足条件的结果元组写入写入缓冲区队列。

```
// cache *ca = cache_init(1000);
cache *ca = NULL;
iterator *r = iterator_init(301, 317, ca);
iterator *s = iterator_init(317, 349, ca);
uint target = 700;
buffered_queue *q = buffered_queue_init(1, target, true);
uint join_count = 0;
while (!iterator_is_end(r) && !iterator_is_end(s)) {
  while (!iterator_is_end(r) && cmp_greater(iterator_now(s), iterator_now(r)))
    iterator_next(r);
  if (!iterator_is_end(r) && SEQ(iterator_now(s), iterator_now(r))) {
    iterator *r_clone = iterator_clone(r);
    // iterator *r_clone = r;
    // uint count_next = 0;
    while (!iterator_is_end(r_clone) &&
          iterator_now(r_clone) &&
          *iterator_now(r_clone) != '\0' &&
          SEQ(iterator_now(s), iterator_now(r_clone))) {
      Dbg("push (%s, %s) (%s, %s)",
          iterator_now(s), iterator_now(s) + 4,
          iterator_now(r_clone), iterator_now(r_clone) + 4);
      buffered_queue_push(q, iterator_now(s));
      buffered_queue_push(q, iterator_now(r_clone));
      join_count++;
      iterator_next(r_clone);
      // count_next++;
    }
    iterator_free_clone(r_clone);
    // iterator_prev_n(r_clone, count_next);
  }
  iterator_next(s);
}
buffered_queue_flush(q);
buffered_queue_free(q);
```

实验结果：

```
[/home/chiro/programs/database-lab/lab5/question4.c:9 q4] ====================================
[/home/chiro/programs/database-lab/lab5/question4.c:10 q4] Q4: 基于排序的连接操作算法(Sort-Merge-Join)
[/home/chiro/programs/database-lab/lab5/question4.c:11 q4] select S.C, S.D, R.A, R.B from S inner join R on
 S.C = R.A
[/home/chiro/programs/database-lab/lab5/question4.c:12 q4] ====================================
[/home/chiro/programs/database-lab/lab5/question4.c:14 q4] 正在排序...
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 排序过程: IO 读写一共 194 次
[/home/chiro/programs/database-lab/lab5/question4.c:18 q4] 排序后数据位于:
[/home/chiro/programs/database-lab/lab5/question4.c:19 q4] R [A, B], block [301, 316]
[/home/chiro/programs/database-lab/lab5/question4.c:20 q4] S [C, D], block [317, 348]
[/home/chiro/programs/database-lab/lab5/question4.c:25 q4] Sort-Merge-Join 算法开始...
[/home/chiro/programs/database-lab/lab5/question4.c:61 q4] 结果如下，储存于 [700, 811]
======== data [700, 812):
[700] (120, 418) (120, 418) (120, 418) (120, 499) (120, 418) (120, 827) (120, 554)
[701] (120, 418) (120, 554) (120, 499) (120, 554) (120, 827) (120, 571) (120, 418)
     ... ...
[810] (139, 884) (139, 476) (140, 551) (140, 610) (140, 610) (140, 610) (140, 915)
[811] (140, 610)
[/home/chiro/programs/database-lab/lab5/question4.c:63 q4] 连接次数: 389
[/home/chiro/programs/database-lab/lab5/main_utils.c:115 buffer_report] Buffer: IO 读写一共 426 次
```

一个重要的点的是，由于算法中需要实现指针回退功能，而这部分功能原本由 LRU Cache 完成，但是 LRU Cache 管理的缓冲区无法同时对两个迭代器使用，于是只能使用克隆迭代器再释放的方法，增加了 IO 次数，需要后期改进。

(5) 实现基于散列()的两趟扫描算法，实现交、并、差其中一种集合操作算法

问题分析：见下方附加题

实验结果：见下方附加题

五、 附加题

对剩余的两种集合操作进行问题分析，并给出程序正确运行的结果截图。

由于基于排序的两趟扫描算法与 TPMMS 十分相像，于是这里基于已经实现的 TPMMS 中的部分算法完成基于排序的两趟扫描算法的三种集合操作。

首先将两趟扫描算法抽象为一个函数，这个函数会首先对两个数据源排序并去重，然后调用第二趟的函数钩子。

```
uint two_stage_scanning(uint s_left, uint s_right, uint r_left, uint r_right, uint target,
                        uint (*fn)(iterator*, iterator*, buffered_queue*)) {
  uint temp1 = 600, temp2 = 700;
  // 对两个数据源排序并去重
  uint skipped_s = sort_deduplicate_two_stage_scanning(s_left, s_right, temp1);
  uint skipped_r = sort_deduplicate_two_stage_scanning(r_left, r_right, temp2);
  buffered_queue *q = buffered_queue_init(1, target, true);
  uint first_right = temp1 + (s_right - s_left) - skipped_s / 7;
  uint second_right = temp2 + (r_right - r_left) - skipped_r / 7;
  iterator *reader_first = iterator_init(temp1, first_right, NULL);
  iterator *reader_second = iterator_init(temp2, second_right, NULL);
  // 调用函数钩子
  uint skipped_stage2 = fn(reader_first, reader_second, q);
  buffered_queue_flush(q);
  buffered_queue_free(q);
  return skipped_s + skipped_r + skipped_stage2;
}
```

对三种集合操作实现三个函数钩子即可，函数钩子的输入是两个迭代器，输出是一个写入缓冲区队列。

对求 R、S 并集的实现，和归并排序类似，不过每次插入时需要保证和上次插入的值不一样。

```
uint union_stage(iterator* reader_first, iterator* reader_second, buffered_queue *target) {
  char last_insert[9] = "";
  char *a = NULL;
  char *b = NULL;
  uint skipped = 0;
  while (true) {
    a = iterator_now(reader_first);
    b = iterator_now(reader_second);
    if (a == NULL || b == NULL) break;
    if (cmp_greater(a, b) ||
        (SEQ3(a, b) && cmp_greater(a + 4, b + 4))) {
      // a > b
      // 与上次插入不同，可以插入 b
      if (!SEQ_T(b, last_insert)) {
        buffered_queue_push(target, b);
        tuple_copy(last_insert, b);
      } else skipped++;
      iterator_next(reader_second);
    } else {
      // a <= b
      // 与上次插入不同，可以插入 a
      if (!SEQ_T(a, last_insert)) {
        buffered_queue_push(target, a);
        tuple_copy(last_insert, a);
      } else skipped++;
      iterator_next(reader_first);
    }
  }
  // 插入剩下的 b
  while (a == NULL && b != NULL) {
    if (!SEQ_T(b, last_insert)) {
      buffered_queue_push(target, b);
      tuple_copy(last_insert, b);
    } else skipped++;
    iterator_next(reader_second);
    b = iterator_now(reader_second);
  }
  // 插入剩下的 a
  while (a != NULL && b == NULL) {
    if (!SEQ_T(a, last_insert)) {
      buffered_queue_push(target, a);
      tuple_copy(last_insert, a);
    } else skipped++;
    iterator_next(reader_first);
    a = iterator_now(reader_first);
  }
```

对求 R、S 交集的实现，也和归并排序类似，不过对当前最小值。需要和上一次最小值的值相同才插入队列。

```
uint intersect_stage(iterator* reader_first, iterator* reader_second, buffered_queue *target) {
  char last_top[9] = "";
  char *a = NULL;
  char *b = NULL;
  uint skipped = 0;
  while (true) {
    a = iterator_now(reader_first);
    b = iterator_now(reader_second);
    if (a == NULL || b == NULL) break;
    if (cmp_greater(a, b) ||
        (SEQ3(a, b) && cmp_greater(a + 4, b + 4))) {
      // a > b, b 与上次的最小值相同则插入 b
      if (SEQ_T(b, last_top)) {
        buffered_queue_push(target, b);
      } else skipped++;
      tuple_copy(last_top, b);
      iterator_next(reader_second);
    } else {
      // a <= b, a 与上次最小的值相同则插入 a
      if (SEQ_T(a, last_top)) {
        buffered_queue_push(target, a);
      } else skipped++;
      tuple_copy(last_top, a);
      iterator_next(reader_first);
    }
  }
  // 去重插入剩下的 b
  while (a == NULL && b != NULL) {
    iterator_next(reader_second);
    if (SEQ_T(b, last_top)) {
      buffered_queue_push(target, b);
      tuple_copy(last_top, b);
    } else skipped++;
    b = iterator_now(reader_second);
  }
  // 去重插入剩下的 a
  while (a != NULL && b == NULL) {
    iterator_next(reader_first);
    if (SEQ_T(a, last_top)) {
      buffered_queue_push(target, a);
      tuple_copy(last_top, a);
    } else skipped++;
    a = iterator_now(reader_first);
  }
  return skipped;
}
```

最后是对于 S – R 的实现，仔细分类讨论即可。

```
uint difference_set_stage(iterator* reader_first, iterator* reader_second, buffered_queue *target)
  char *a = NULL;
  char *b = NULL;
  uint skipped = 0;
  while (true) {
    a = iterator_now(reader_first);
    b = iterator_now(reader_second);
    if (a == NULL || b == NULL) break;
    if (cmp_greater(a, b) ||
        (SEQ3(a, b) && cmp_greater(a + 4, b + 4))) {
      // a > b
      skipped++;
      // a != b, do not insert b
      iterator_next(reader_second);
    } else {
      // a <= b
      if (!SEQ_T(a, b)) {
        // a != b, insert a
        buffered_queue_push(target, a);
      } else {
        // a == b, do not insert a or b
        skipped += 2;
        iterator_next(reader_second);
      }
      iterator_next(reader_first);
    }
  }
  while (a == NULL && b != NULL) {
    skipped++;
    iterator_next(reader_second);
    b = iterator_now(reader_second);
  }
  while (a != NULL && b == NULL) {
    buffered_queue_push(target, a);
    iterator_next(reader_first);
    a = iterator_now(reader_first);
  }
  return skipped;
}
```

实验结果：

```
[/home/chiro/programs/database-lab/lab5/question5.c:230 q5] ======================
[/home/chiro/programs/database-lab/lab5/question5.c:231 q5] Q5: 基于排序的两趟扫描算法
[/home/chiro/programs/database-lab/lab5/question5.c:232 q5] ======================
[/home/chiro/programs/database-lab/lab5/question5.c:236 q5] 计算 S union R
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 计算 S union R: IO 读写一共 392 次
[/home/chiro/programs/database-lab/lab5/question5.c:240 q5] 计算结果储存于 [300, 346]
[/home/chiro/programs/database-lab/lab5/question5.c:246 q5] S union R 结果元组数量为 323
======== data [300, 347):
[300] (100, 400) (100, 421) (100, 439) (101, 405) (101, 406) (101, 409) (101, 410)
[301] (101, 470) (102, 413) (102, 465) (102, 492) (104, 440) (104, 450) (105, 428)
     ... ...
[345] (157, 902) (158, 432) (158, 602) (158, 764) (159, 447) (159, 585) (159, 630)
[346] (159, 737)
[/home/chiro/programs/database-lab/lab5/question5.c:253 q5] 计算 S intersects R
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 计算 S intersects R: IO 读写一共 347
 次
[/home/chiro/programs/database-lab/lab5/question5.c:257 q5] 计算结果储存于 [350, 351]
[/home/chiro/programs/database-lab/lab5/question5.c:263 q5] S intersects R 结果元组数量为 13
======== data [350, 352):
[350] (120, 418) (120, 827) (122, 546) (123, 477) (125, 886) (127, 767) (128, 447)
[351] (129, 430) (130, 436) (130, 656) (134, 437) (139, 461) (140, 610)
[/home/chiro/programs/database-lab/lab5/question5.c:270 q5] 计算 S - R
[/home/chiro/programs/database-lab/lab5/main_utils.c:123 buffer_report_msg] 计算 S - R: IO 读写一共 376 次
[/home/chiro/programs/database-lab/lab5/question5.c:274 q5] 计算结果储存于 [360, 390]
[/home/chiro/programs/database-lab/lab5/question5.c:280 q5] S - R 结果元组数量为 211
======== data [360, 391):
[360] (120, 554) (120, 571) (120, 581) (120, 736) (120, 775) (120, 781) (120, 852)
[361] (121, 475) (121, 582) (121, 596) (121, 739) (121, 793) (122, 463) (122, 554)
     ... ...
[389] (157, 902) (158, 432) (158, 602) (158, 764) (159, 447) (159, 585) (159, 630)
[390] (159, 737)
```

六、 总结

总结本次实验的遇到并解决的问题、收获及反思。

在本次实验中，由于规定了实现语言是 C 语言，许多使用习惯了的高级语言特性需要自己手搓，还经常出现内存问题，所以实验难度较高。本实验中通过使用 GCC 中的 lambda 函数、调试宏、单元测试、差分测试、CMake 等保证了 C 语言项目的规范、严谨、正确性。本实验提供了 extmem 调用接口来完成对磁盘的读写，所以如果能使用我们都学过的有更多抽象特性的语言，写起来会轻松很多。

由于手搓的东西比较多，代码量也相对比较大，许多人一个文件写了 1200+ 行。既然已经使用 CodeBlocks，可以建立多文件的代码框架，减少大文件，以方便同学们完成实验。

```
→ chiro@chiro-pc  ~/programs/database-lab/lab5 git:(master) × cloc *.c *.h debug_macros
      23 text files.
      23 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 1.96  T=0.01 s (1616.1 files/s, 192662.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                               13            159            365           1619
C/C++ Header                    10            156             68            375
-------------------------------------------------------------------------------
SUM:                            23            315            433           1994
-------------------------------------------------------------------------------
→ chiro@chiro-pc  ~/programs/database-lab/lab5 git:(master) ×
```

与 OS 实验 5 相比，其使用了更通用的 CMake 构建系统，能适配更多 IDE/编辑器，能更加方便同学们的使用习惯。

附录：整个实验程序的完整输出：

```
[main.c:8 main] Lab5 program launched!
[main.c:10 main] 表结构和数据存储位置:
[main.c:11 main] R [A, B], block [1,  16]
[main.c:12 main] S [C, D], block [17, 48]
[question1.c:9 q1] ========================
[question1.c:10 q1] Q1: 基于线性搜索的关系选择算法
[question1.c:11 q1] select S.C,S.D from S where S.C = 128
[question1.c:12 q1] ========================
[main_utils.c:123 buffer_report_msg] 线性搜索: IO 读写一共 35 次
[question1.c:28 q1] 选择结果如下，储存于地址 100
======== data [100, -1):
[100] (128, 684) (128, 431) (128, 615) (128, 429) (128, 584) (128, 592) (128, 457)
[101] (128, 720) (128, 447) (128, 871)
[question1.c:30 q1] 满足选择条件的元组一共 10 个
[question2.c:88 q2] ============================
[question2.c:89 q2] Q2: 两阶段多路归并排序算法(TPMMS)
[question2.c:90 q2] 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。
[question2.c:91 q2] ============================
[main_utils.c:123 buffer_report_msg] TPMMS: IO 读写一共 194 次
[question2.c:96 q2] R 排序后如下，储存于地址 [301, 316]
======== data [301, 317):
[301] (100, 400) (100, 421) (100, 439) (101, 405) (101, 406) (101, 409) (101, 410)
[302] (101, 470) (102, 413) (102, 465) (102, 492) (104, 440) (104, 450) (105, 428)
[303] (105, 476) (105, 497) (106, 461) (107, 411) (107, 434) (107, 477) (108, 436)
```

[304] (108, 482) (109, 404) (109, 409) (109, 472) (110, 405) (110, 413) (110, 450)

[305] (110, 491) (111, 445) (112, 467) (114, 410) (114, 414) (114, 425) (115, 401)

[306] (116, 414) (116, 420) (116, 421) (116, 424) (116, 452) (116, 470) (117, 403)

[307] (117, 412) (117, 426) (117, 438) (117, 442) (117, 475) (118, 414) (118, 478)

[308] (119, 406) (119, 428) (119, 431) (120, 418) (120, 499) (120, 827) (121, 438)

[309] (121, 464) (122, 474) (122, 546) (123, 422) (123, 452) (123, 477) (124, 410)

[310] (124, 412) (124, 426) (124, 468) (124, 499) (125, 886) (126, 423) (126, 485)

[311] (127, 767) (128, 447) (128, 453) (128, 459) (129, 402) (129, 430) (129, 455)

[312] (129, 475) (129, 488) (130, 411) (130, 417) (130, 436) (130, 495) (130, 656)

[313] (131, 454) (131, 479) (131, 492) (132, 422) (132, 483) (133, 428) (133, 441)

[314] (133, 455) (133, 467) (134, 437) (134, 459) (134, 486) (135, 441) (137, 420)

[315] (137, 428) (137, 451) (137, 471) (137, 473) (137, 480) (138, 407) (138, 411)

[316] (138, 468) (138, 475) (138, 495) (138, 497) (139, 461) (139, 476) (140, 610)

[question2.c:98 q2] S 排序后如下，储存于地址 [317, 348]

======== data [317, 349):

[317] (120, 418) (120, 554) (120, 571) (120, 581) (120, 736) (120, 775) (120, 781)

[318] (120, 827) (120, 852) (121, 475) (121, 582) (121, 596) (121, 739) (121, 793)

[319] (122, 463) (122, 546) (122, 554) (122, 646) (122, 682) (122, 756) (122, 760)

[320] (123, 468) (123, 477) (123, 532) (123, 587) (123, 733) (123, 791) (123, 794)

[321] (123, 889) (124, 424) (124, 566) (124, 605) (124, 767) (124, 844) (124, 854)

[322] (125, 496) (125, 587) (125, 625) (125, 725) (125, 808) (125, 886) (125, 901)

[323] (126, 685) (126, 774) (126, 919) (127, 680) (127, 743) (127, 767) (127, 816)

[324] (128, 429) (128, 431) (128, 447) (128, 457) (128, 584) (128, 592) (128, 615)

[325] (128, 684) (128, 720) (128, 871) (129, 430) (129, 486) (129, 492) (129, 535)

[326] (129, 591) (129, 683) (129, 844) (130, 436) (130, 474) (130, 476) (130, 516)

[327] (130, 556) (130, 656) (130, 766) (130, 790) (131, 472) (131, 617) (131, 718)

[328] (131, 799) (131, 808) (131, 892) (132, 606) (132, 640) (133, 456) (134, 437)

[329] (134, 457) (134, 485) (134, 494) (134, 552) (134, 756) (134, 883) (134, 900)

[330] (135, 542) (135, 671) (135, 691) (135, 747) (135, 919) (136, 434) (136, 454)

[331] (136, 608) (136, 711) (136, 743) (136, 810) (137, 442) (137, 460) (137, 585)

[332] (137, 660) (137, 733) (137, 831) (137, 854) (137, 895) (137, 912) (138, 512)

[333] (138, 598) (138, 639) (138, 674) (138, 700) (138, 816) (139, 461) (139, 488)

[334] (139, 576) (139, 802) (139, 815) (139, 878) (139, 884) (140, 551) (140, 610)

[335] (140, 915) (141, 504) (141, 629) (141, 636) (141, 732) (141, 754) (142, 450)

[336] (142, 679) (142, 718) (142, 748) (142, 841) (143, 465) (143, 732) (144, 609)

[337] (144, 650) (144, 667) (144, 838) (145, 456) (145, 521) (145, 540) (145, 546)

[338] (145, 570) (145, 625) (145, 689) (145, 805) (145, 840) (146, 445) (146, 463)

[339] (146, 690) (146, 721) (146, 833) (146, 834) (146, 843) (147, 552) (147, 561)

[340] (147, 663) (147, 750) (148, 547) (148, 570) (148, 616) (148, 635) (148, 756)

[341] (148, 854) (148, 876) (149, 523) (149, 573) (149, 763) (149, 839) (149, 873)

[342] (150, 709) (150, 835) (150, 839) (150, 890) (151, 444) (151, 457) (151, 461)

[343] (151, 578) (151, 644) (151, 704) (152, 573) (152, 634) (152, 717) (152, 826)

```
[344] (153, 456) (153, 573) (153, 613) (153, 641) (153, 642) (154, 460) (154, 528)
[345] (154, 585) (154, 593) (154, 728) (154, 810) (155, 464) (155, 544) (155, 600)
[346] (155, 668) (155, 916) (156, 433) (156, 479) (156, 482) (156, 600) (156, 698)
[347] (156, 809) (156, 839) (156, 857) (157, 456) (157, 498) (157, 691) (157, 902)
[348] (158, 432) (158, 602) (158, 764) (159, 447) (159, 585) (159, 630) (159, 737)
[question3.c:66 q3] ============================
[question3.c:67 q3] Q3: 基于索引的关系选择算法
[question3.c:68 q3] 排序结果为关系 R 或 S 分别建立索引文件，
[question3.c:69 q3] 模拟实现 select S.C, S.D from S where S.C = 128
[question3.c:70 q3] ============================
[question3.c:72 q3] 正在排序...
[main_utils.c:123 buffer_report_msg] 排序过程: IO 读写一共 194 次
[question3.c:77 q3] 正在建立索引...
[main_utils.c:123 buffer_report_msg] 建立索引过程: IO 读写一共 62 次
[question3.c:85 q3] 索引文件位于 [501...], [517...]
[main_utils.c:123 buffer_report_msg] 检索过程: IO 读写一共 7 次
[question3.c:93 q3] 满足选择条件的元组一共 10 个，如下
======== data in queue (total=4, size=1, offset/8=3, addr=-1)
(128, 429) (128, 431) (128, 447) (128, 457) (128, 584) (128, 592) (128, 615)
(128, 684) (128, 720) (128, 871)
[question4.c:9 q4] ====================================
[question4.c:10 q4] Q4: 基于排序的连接操作算法(Sort-Merge-Join)
[question4.c:11 q4] select S.C, S.D, R.A, R.B from S inner join R on S.C = R.A
[question4.c:12 q4] ====================================
[question4.c:14 q4] 正在排序...
[main_utils.c:123 buffer_report_msg] 排序过程: IO 读写一共 194 次
[question4.c:18 q4] 排序后数据位于:
[question4.c:19 q4] R [A, B], block [301, 316]
[question4.c:20 q4] S [C, D], block [317, 348]
[question4.c:25 q4] Sort-Merge-Join 算法开始...
[question4.c:61 q4] 结果如下，储存于 [700, 811]
======== data [700, 812):
[700] (120, 418) (120, 418) (120, 418) (120, 499) (120, 418) (120, 827) (120, 554)
[701] (120, 418) (120, 554) (120, 499) (120, 554) (120, 827) (120, 571) (120, 418)
[702] (120, 571) (120, 499) (120, 571) (120, 827) (120, 581) (120, 418) (120, 581)
[703] (120, 499) (120, 581) (120, 827) (120, 736) (120, 418) (120, 736) (120, 499)
[704] (120, 736) (120, 827) (120, 775) (120, 418) (120, 775) (120, 499) (120, 775)
[705] (120, 827) (120, 781) (120, 418) (120, 781) (120, 499) (120, 781) (120, 827)
[706] (120, 827) (120, 418) (120, 827) (120, 499) (120, 827) (120, 827) (120, 852)
[707] (120, 418) (120, 852) (120, 499) (120, 852) (120, 827) (121, 475) (121, 438)
[708] (121, 475) (121, 464) (121, 582) (121, 438) (121, 582) (121, 464) (121, 596)
[709] (121, 438) (121, 596) (121, 464) (121, 739) (121, 438) (121, 739) (121, 464)
```

[710] (121, 793) (121, 438) (121, 793) (121, 464) (122, 463) (122, 474) (122, 463)

[711] (122, 546) (122, 546) (122, 474) (122, 546) (122, 546) (122, 554) (122, 474)

[712] (122, 554) (122, 546) (122, 646) (122, 474) (122, 646) (122, 546) (122, 682)

[713] (122, 474) (122, 682) (122, 546) (122, 756) (122, 474) (122, 756) (122, 546)

[714] (122, 760) (122, 474) (122, 760) (122, 546) (123, 468) (123, 422) (123, 468)

[715] (123, 452) (123, 468) (123, 477) (123, 477) (123, 422) (123, 477) (123, 452)

[716] (123, 477) (123, 477) (123, 532) (123, 422) (123, 532) (123, 452) (123, 532)

[717] (123, 477) (123, 587) (123, 422) (123, 587) (123, 452) (123, 587) (123, 477)

[718] (123, 733) (123, 422) (123, 733) (123, 452) (123, 733) (123, 477) (123, 791)

[719] (123, 422) (123, 791) (123, 452) (123, 791) (123, 477) (123, 794) (123, 422)

[720] (123, 794) (123, 452) (123, 794) (123, 477) (123, 889) (123, 422) (123, 889)

[721] (123, 452) (123, 889) (123, 477) (124, 424) (124, 410) (124, 424) (124, 412)

[722] (124, 424) (124, 426) (124, 424) (124, 468) (124, 424) (124, 499) (124, 566)

[723] (124, 410) (124, 566) (124, 412) (124, 566) (124, 426) (124, 566) (124, 468)

[724] (124, 566) (124, 499) (124, 605) (124, 410) (124, 605) (124, 412) (124, 605)

[725] (124, 426) (124, 605) (124, 468) (124, 605) (124, 499) (124, 767) (124, 410)

[726] (124, 767) (124, 412) (124, 767) (124, 426) (124, 767) (124, 468) (124, 767)

[727] (124, 499) (124, 844) (124, 410) (124, 844) (124, 412) (124, 844) (124, 426)

[728] (124, 844) (124, 468) (124, 844) (124, 499) (124, 854) (124, 410) (124, 854)

[729] (124, 412) (124, 854) (124, 426) (124, 854) (124, 468) (124, 854) (124, 499)

[730] (125, 496) (125, 886) (125, 587) (125, 886) (125, 625) (125, 886) (125, 725)

[731] (125, 886) (125, 808) (125, 886) (125, 886) (125, 886) (125, 901) (125, 886)

[732] (126, 685) (126, 423) (126, 685) (126, 485) (126, 774) (126, 423) (126, 774)

[733] (126, 485) (126, 919) (126, 423) (126, 919) (126, 485) (127, 680) (127, 767)

[734] (127, 743) (127, 767) (127, 767) (127, 767) (127, 816) (127, 767) (128, 429)

[735] (128, 447) (128, 429) (128, 453) (128, 429) (128, 459) (128, 431) (128, 447)

[736] (128, 431) (128, 453) (128, 431) (128, 459) (128, 447) (128, 447) (128, 447)

[737] (128, 453) (128, 447) (128, 459) (128, 457) (128, 447) (128, 457) (128, 453)

[738] (128, 457) (128, 459) (128, 584) (128, 447) (128, 584) (128, 453) (128, 584)

[739] (128, 459) (128, 592) (128, 447) (128, 592) (128, 453) (128, 592) (128, 459)

[740] (128, 615) (128, 447) (128, 615) (128, 453) (128, 615) (128, 459) (128, 684)

[741] (128, 447) (128, 684) (128, 453) (128, 684) (128, 459) (128, 720) (128, 447)

[742] (128, 720) (128, 453) (128, 720) (128, 459) (128, 871) (128, 447) (128, 871)

[743] (128, 453) (128, 871) (128, 459) (129, 430) (129, 402) (129, 430) (129, 430)

[744] (129, 430) (129, 455) (129, 430) (129, 475) (129, 430) (129, 488) (129, 486)

[745] (129, 402) (129, 486) (129, 430) (129, 486) (129, 455) (129, 486) (129, 475)

[746] (129, 486) (129, 488) (129, 492) (129, 402) (129, 492) (129, 430) (129, 492)

[747] (129, 455) (129, 492) (129, 475) (129, 492) (129, 488) (129, 535) (129, 402)

[748] (129, 535) (129, 430) (129, 535) (129, 455) (129, 535) (129, 475) (129, 535)

[749] (129, 488) (129, 591) (129, 402) (129, 591) (129, 430) (129, 591) (129, 455)

[750] (129, 591) (129, 475) (129, 591) (129, 488) (129, 683) (129, 402) (129, 683)

[751] (129, 430) (129, 683) (129, 455) (129, 683) (129, 475) (129, 683) (129, 488)

[752] (129, 844) (129, 402) (129, 844) (129, 430) (129, 844) (129, 455) (129, 844)

[753] (129, 475) (129, 844) (129, 488) (130, 436) (130, 411) (130, 436) (130, 417)

[754] (130, 436) (130, 436) (130, 436) (130, 495) (130, 436) (130, 656) (130, 474)

[755] (130, 411) (130, 474) (130, 417) (130, 474) (130, 436) (130, 474) (130, 495)

[756] (130, 474) (130, 656) (130, 476) (130, 411) (130, 476) (130, 417) (130, 476)

[757] (130, 436) (130, 476) (130, 495) (130, 476) (130, 656) (130, 516) (130, 411)

[758] (130, 516) (130, 417) (130, 516) (130, 436) (130, 516) (130, 495) (130, 516)

[759] (130, 656) (130, 556) (130, 411) (130, 556) (130, 417) (130, 556) (130, 436)

[760] (130, 556) (130, 495) (130, 556) (130, 656) (130, 656) (130, 411) (130, 656)

[761] (130, 417) (130, 656) (130, 436) (130, 656) (130, 495) (130, 656) (130, 656)

[762] (130, 766) (130, 411) (130, 766) (130, 417) (130, 766) (130, 436) (130, 766)

[763] (130, 495) (130, 766) (130, 656) (130, 790) (130, 411) (130, 790) (130, 417)

[764] (130, 790) (130, 436) (130, 790) (130, 495) (130, 790) (130, 656) (131, 472)

[765] (131, 454) (131, 472) (131, 479) (131, 472) (131, 492) (131, 617) (131, 454)

[766] (131, 617) (131, 479) (131, 617) (131, 492) (131, 718) (131, 454) (131, 718)

[767] (131, 479) (131, 718) (131, 492) (131, 799) (131, 454) (131, 799) (131, 479)

[768] (131, 799) (131, 492) (131, 808) (131, 454) (131, 808) (131, 479) (131, 808)

[769] (131, 492) (131, 892) (131, 454) (131, 892) (131, 479) (131, 892) (131, 492)

[770] (132, 606) (132, 422) (132, 606) (132, 483) (132, 640) (132, 422) (132, 640)

[771] (132, 483) (133, 456) (133, 428) (133, 456) (133, 441) (133, 456) (133, 455)

[772] (133, 456) (133, 467) (134, 437) (134, 437) (134, 437) (134, 459) (134, 437)

[773] (134, 486) (134, 457) (134, 437) (134, 457) (134, 459) (134, 457) (134, 486)

[774] (134, 485) (134, 437) (134, 485) (134, 459) (134, 485) (134, 486) (134, 494)

[775] (134, 437) (134, 494) (134, 459) (134, 494) (134, 486) (134, 552) (134, 437)

[776] (134, 552) (134, 459) (134, 552) (134, 486) (134, 756) (134, 437) (134, 756)

[777] (134, 459) (134, 756) (134, 486) (134, 883) (134, 437) (134, 883) (134, 459)

[778] (134, 883) (134, 486) (134, 900) (134, 437) (134, 900) (134, 459) (134, 900)

[779] (134, 486) (135, 542) (135, 441) (135, 671) (135, 441) (135, 691) (135, 441)

[780] (135, 747) (135, 441) (135, 919) (135, 441) (137, 442) (137, 420) (137, 442)

[781] (137, 428) (137, 442) (137, 451) (137, 442) (137, 471) (137, 442) (137, 473)

[782] (137, 442) (137, 480) (137, 460) (137, 420) (137, 460) (137, 428) (137, 460)

[783] (137, 451) (137, 460) (137, 471) (137, 460) (137, 473) (137, 460) (137, 480)

[784] (137, 585) (137, 420) (137, 585) (137, 428) (137, 585) (137, 451) (137, 585)

[785] (137, 471) (137, 585) (137, 473) (137, 585) (137, 480) (137, 660) (137, 420)

[786] (137, 660) (137, 428) (137, 660) (137, 451) (137, 660) (137, 471) (137, 660)

[787] (137, 473) (137, 660) (137, 480) (137, 733) (137, 420) (137, 733) (137, 428)

[788] (137, 733) (137, 451) (137, 733) (137, 471) (137, 733) (137, 473) (137, 733)

[789] (137, 480) (137, 831) (137, 420) (137, 831) (137, 428) (137, 831) (137, 451)

[790] (137, 831) (137, 471) (137, 831) (137, 473) (137, 831) (137, 480) (137, 854)

[791] (137, 420) (137, 854) (137, 428) (137, 854) (137, 451) (137, 854) (137, 471)

[792] (137, 854) (137, 473) (137, 854) (137, 480) (137, 895) (137, 420) (137, 895)

[793] (137, 428) (137, 895) (137, 451) (137, 895) (137, 471) (137, 895) (137, 473)

[794] (137, 895) (137, 480) (137, 912) (137, 420) (137, 912) (137, 428) (137, 912)

[795] (137, 451) (137, 912) (137, 471) (137, 912) (137, 473) (137, 912) (137, 480)

[796] (138, 512) (138, 407) (138, 512) (138, 411) (138, 512) (138, 468) (138, 512)

[797] (138, 475) (138, 512) (138, 495) (138, 512) (138, 497) (138, 598) (138, 407)

[798] (138, 598) (138, 411) (138, 598) (138, 468) (138, 598) (138, 475) (138, 598)

[799] (138, 495) (138, 598) (138, 497) (138, 639) (138, 407) (138, 639) (138, 411)

[800] (138, 639) (138, 468) (138, 639) (138, 475) (138, 639) (138, 495) (138, 639)

[801] (138, 497) (138, 674) (138, 407) (138, 674) (138, 411) (138, 674) (138, 468)

[802] (138, 674) (138, 475) (138, 674) (138, 495) (138, 674) (138, 497) (138, 700)

[803] (138, 407) (138, 700) (138, 411) (138, 700) (138, 468) (138, 700) (138, 475)

[804] (138, 700) (138, 495) (138, 700) (138, 497) (138, 816) (138, 407) (138, 816)

[805] (138, 411) (138, 816) (138, 468) (138, 816) (138, 475) (138, 816) (138, 495)

[806] (138, 816) (138, 497) (139, 461) (139, 461) (139, 461) (139, 476) (139, 488)

[807] (139, 461) (139, 488) (139, 476) (139, 576) (139, 461) (139, 576) (139, 476)

[808] (139, 802) (139, 461) (139, 802) (139, 476) (139, 815) (139, 461) (139, 815)

[809] (139, 476) (139, 878) (139, 461) (139, 878) (139, 476) (139, 884) (139, 461)

[810] (139, 884) (139, 476) (140, 551) (140, 610) (140, 610) (140, 610) (140, 915)

[811] (140, 610)

[question4.c:63 q4] 连接次数: 389

[main_utils.c:115 buffer_report] Buffer: IO 读写一共 426 次

[question5.c:240 q5] ====================

[question5.c:241 q5] Q5: 基于排序的两趟扫描算法

[question5.c:242 q5] ====================

[question5.c:246 q5] 计算 S union R

[main_utils.c:123 buffer_report_msg] 计算 S union R: IO 读写一共 392 次

[question5.c:250 q5] 计算结果储存于 [300, 346]

[question5.c:256 q5] S union R 结果元组数量为 323

======== data [300, 347):

[300] (100, 400) (100, 421) (100, 439) (101, 405) (101, 406) (101, 409) (101, 410)

[301] (101, 470) (102, 413) (102, 465) (102, 492) (104, 440) (104, 450) (105, 428)

[302] (105, 476) (105, 497) (106, 461) (107, 411) (107, 434) (107, 477) (108, 436)

[303] (108, 482) (109, 404) (109, 409) (109, 472) (110, 405) (110, 413) (110, 450)

[304] (110, 491) (111, 445) (112, 467) (114, 410) (114, 414) (114, 425) (115, 401)

[305] (116, 414) (116, 420) (116, 421) (116, 424) (116, 452) (116, 470) (117, 403)

[306] (117, 412) (117, 426) (117, 438) (117, 442) (117, 475) (118, 414) (118, 478)

[307] (119, 406) (119, 428) (119, 431) (120, 418) (120, 499) (120, 554) (120, 571)

[308] (120, 581) (120, 736) (120, 775) (120, 781) (120, 827) (120, 852) (121, 438)

[309] (121, 464) (121, 475) (121, 582) (121, 596) (121, 739) (121, 793) (122, 463)

[310] (122, 474) (122, 546) (122, 554) (122, 646) (122, 682) (122, 756) (122, 760)

[311] (123, 422) (123, 452) (123, 468) (123, 477) (123, 532) (123, 587) (123, 733)

[312] (123, 791) (123, 794) (123, 889) (124, 410) (124, 412) (124, 424) (124, 426)

[313] (124, 468) (124, 499) (124, 566) (124, 605) (124, 767) (124, 844) (124, 854)

[314] (125, 496) (125, 587) (125, 625) (125, 725) (125, 808) (125, 886) (125, 901)
[315] (126, 423) (126, 485) (126, 685) (126, 774) (126, 919) (127, 680) (127, 743)
[316] (127, 767) (127, 816) (128, 429) (128, 431) (128, 447) (128, 453) (128, 457)
[317] (128, 459) (128, 584) (128, 592) (128, 615) (128, 684) (128, 720) (128, 871)
[318] (129, 402) (129, 430) (129, 455) (129, 475) (129, 486) (129, 488) (129, 492)
[319] (129, 535) (129, 591) (129, 683) (129, 844) (130, 411) (130, 417) (130, 436)
[320] (130, 474) (130, 476) (130, 495) (130, 516) (130, 556) (130, 656) (130, 766)
[321] (130, 790) (131, 454) (131, 472) (131, 479) (131, 492) (131, 617) (131, 718)
[322] (131, 799) (131, 808) (131, 892) (132, 422) (132, 483) (132, 606) (132, 640)
[323] (133, 428) (133, 441) (133, 455) (133, 456) (133, 467) (134, 437) (134, 457)
[324] (134, 459) (134, 485) (134, 486) (134, 494) (134, 552) (134, 756) (134, 883)
[325] (134, 900) (135, 441) (135, 542) (135, 671) (135, 691) (135, 747) (135, 919)
[326] (136, 434) (136, 454) (136, 608) (136, 711) (136, 743) (136, 810) (137, 420)
[327] (137, 428) (137, 442) (137, 451) (137, 460) (137, 471) (137, 473) (137, 480)
[328] (137, 585) (137, 660) (137, 733) (137, 831) (137, 854) (137, 895) (137, 912)
[329] (138, 407) (138, 411) (138, 468) (138, 475) (138, 495) (138, 497) (138, 512)
[330] (138, 598) (138, 639) (138, 674) (138, 700) (138, 816) (139, 461) (139, 476)
[331] (139, 488) (139, 576) (139, 802) (139, 815) (139, 878) (139, 884) (140, 551)
[332] (140, 610) (140, 915) (141, 504) (141, 629) (141, 636) (141, 732) (141, 754)
[333] (142, 450) (142, 679) (142, 718) (142, 748) (142, 841) (143, 465) (143, 732)
[334] (144, 609) (144, 650) (144, 667) (144, 838) (145, 456) (145, 521) (145, 540)
[335] (145, 546) (145, 570) (145, 625) (145, 689) (145, 805) (145, 840) (146, 445)
[336] (146, 463) (146, 690) (146, 721) (146, 833) (146, 834) (146, 843) (147, 552)
[337] (147, 561) (147, 663) (147, 750) (148, 547) (148, 570) (148, 616) (148, 635)
[338] (148, 756) (148, 854) (148, 876) (149, 523) (149, 573) (149, 763) (149, 839)
[339] (149, 873) (150, 709) (150, 835) (150, 839) (150, 890) (151, 444) (151, 457)
[340] (151, 461) (151, 578) (151, 644) (151, 704) (152, 573) (152, 634) (152, 717)
[341] (152, 826) (153, 456) (153, 573) (153, 613) (153, 641) (153, 642) (154, 460)
[342] (154, 528) (154, 585) (154, 593) (154, 728) (154, 810) (155, 464) (155, 544)
[343] (155, 600) (155, 668) (155, 916) (156, 433) (156, 479) (156, 482) (156, 600)
[344] (156, 698) (156, 809) (156, 839) (156, 857) (157, 456) (157, 498) (157, 691)
[345] (157, 902) (158, 432) (158, 602) (158, 764) (159, 447) (159, 585) (159, 630)
[346] (159, 737)
[question5.c:263 q5] 计算 S intersects R
[main_utils.c:123 buffer_report_msg] 计算 S intersects R: IO 读写一共 347 次
[question5.c:267 q5] 计算结果储存于 [350, 351]
[question5.c:273 q5] S intersects R 结果元组数量为 13
======== data [350, 352):
[350] (120, 418) (120, 827) (122, 546) (123, 477) (125, 886) (127, 767) (128, 447)
[351] (129, 430) (130, 436) (130, 656) (134, 437) (139, 461) (140, 610)
[question5.c:280 q5] 计算 S - R
[main_utils.c:123 buffer_report_msg] 计算 S - R: IO 读写一共 376 次

[question5.c:284 q5] 计算结果储存于 [360, 390]

[question5.c:290 q5] S - R 结果元组数量为 211

======== data [360, 391):

[360] (120, 554) (120, 571) (120, 581) (120, 736) (120, 775) (120, 781) (120, 852)

[361] (121, 475) (121, 582) (121, 596) (121, 739) (121, 793) (122, 463) (122, 554)

[362] (122, 646) (122, 682) (122, 756) (122, 760) (123, 468) (123, 532) (123, 587)

[363] (123, 733) (123, 791) (123, 794) (123, 889) (124, 424) (124, 566) (124, 605)

[364] (124, 767) (124, 844) (124, 854) (125, 496) (125, 587) (125, 625) (125, 725)

[365] (125, 808) (125, 901) (126, 685) (126, 774) (126, 919) (127, 680) (127, 743)

[366] (127, 816) (128, 429) (128, 431) (128, 457) (128, 584) (128, 592) (128, 615)

[367] (128, 684) (128, 720) (128, 871) (129, 486) (129, 492) (129, 535) (129, 591)

[368] (129, 683) (129, 844) (130, 474) (130, 476) (130, 516) (130, 556) (130, 766)

[369] (130, 790) (131, 472) (131, 617) (131, 718) (131, 799) (131, 808) (131, 892)

[370] (132, 606) (132, 640) (133, 456) (134, 457) (134, 485) (134, 494) (134, 552)

[371] (134, 756) (134, 883) (134, 900) (135, 542) (135, 671) (135, 691) (135, 747)

[372] (135, 919) (136, 434) (136, 454) (136, 608) (136, 711) (136, 743) (136, 810)

[373] (137, 442) (137, 460) (137, 585) (137, 660) (137, 733) (137, 831) (137, 854)

[374] (137, 895) (137, 912) (138, 512) (138, 598) (138, 639) (138, 674) (138, 700)

[375] (138, 816) (139, 488) (139, 576) (139, 802) (139, 815) (139, 878) (139, 884)

[376] (140, 551) (140, 915) (141, 504) (141, 629) (141, 636) (141, 732) (141, 754)

[377] (142, 450) (142, 679) (142, 718) (142, 748) (142, 841) (143, 465) (143, 732)

[378] (144, 609) (144, 650) (144, 667) (144, 838) (145, 456) (145, 521) (145, 540)

[379] (145, 546) (145, 570) (145, 625) (145, 689) (145, 805) (145, 840) (146, 445)

[380] (146, 463) (146, 690) (146, 721) (146, 833) (146, 834) (146, 843) (147, 552)

[381] (147, 561) (147, 663) (147, 750) (148, 547) (148, 570) (148, 616) (148, 635)

[382] (148, 756) (148, 854) (148, 876) (149, 523) (149, 573) (149, 763) (149, 839)

[383] (149, 873) (150, 709) (150, 835) (150, 839) (150, 890) (151, 444) (151, 457)

[384] (151, 461) (151, 578) (151, 644) (151, 704) (152, 573) (152, 634) (152, 717)

[385] (152, 826) (153, 456) (153, 573) (153, 613) (153, 641) (153, 642) (154, 460)

[386] (154, 528) (154, 585) (154, 593) (154, 728) (154, 810) (155, 464) (155, 544)

[387] (155, 600) (155, 668) (155, 916) (156, 433) (156, 479) (156, 482) (156, 600)

[388] (156, 698) (156, 809) (156, 839) (156, 857) (157, 456) (157, 498) (157, 691)

[389] (157, 902) (158, 432) (158, 602) (158, 764) (159, 447) (159, 585) (159, 630)

[390] (159, 737)