PJ2 PostgreSQL 上的 Similarity Join 实现

宁晨然 17307130178 田睿 17300750084

一、系统与源码理解

首先需要对 postgresql 进行系统上的理解，不需要很深入的理解完整的架构，而是通过查询理解 postgresql 的执行过程。postgresql 是一种对象关系数据库服务器(数据库管理系统)，在数据库交互命令行直接输入 sql 语句，可以实现查询处理。

（一）、系统理解

1.服务器连接：需要建立应用程序到 postgresql 服务器的连接，进入交互界面。

2.分析器阶段：根据应用程序发送来的查询，核对语法并且创建一个查询树。

3.重写系统：接受由分析阶段创建的查询树并且搜索可以应用到该查询树上的存储在系统表里的规则，进行规则转换。

4.优化：优化器接收查询树后创建查询计划，将查询计划输入执行器。

5.执行：执行器递归地访问规划树，按照规划制定的方式检索数据行，最后返回生成的数据行。

我们更加关心的是：如何处理一个单次查询？如何评价一次查询？

对于 postgresql 的一次查询流程

execute_sql_string(src/backend/commands/extension.c)

分为以下几个方面：

①PLSQL 语句 string 生成语法树 parseTree。用户在应用程序（linux 命令行）输入的 string 的 SQL 语句转换为原始语法树（parseTree），利用函数 pg_parse_query() 完成 sql 语句到语法树的转换，返回到一个语法树 list。

```
static void
execute_sql_string(const char *sql, const char *filename)
{
    List       *raw_parsetree_list;
    DestReceiver *dest;
    ListCell   *lc1;

    /*
     * Parse the SQL string into a list of raw parse trees.
     */
    raw_parsetree_list = pg_parse_query(sql);

    /* All output from SELECTs goes to the bit bucket */
    dest = CreateDestReceiver(DestNone);
```

②生成查询树 queryTree。语法树转换成查询树时，根据深度优先遍历所有语法树节点，创建一个 Query 对象。由于可能生成多种查询树，所以会生成查询树列表 queryTreeList，pg_analyze_and_rewrite 函数(src/backend/tcop/postgresql.c)负责把生的语法树分析后，重写出查询树列表。

③查询树转计划语句树 planStmtTree。循环查询树 List，调用 pg_plan_queries 把查询树转换成 PlannedStmt，为重写的查询列表生成计划。

```
/*
 * Do parse analysis, rule rewrite, planning, and execution for each raw
 * parsetree.  We must fully execute each query before beginning parse
 * analysis on the next one, since there may be interdependencies.
 */
foreach(lc1, raw_parsetree_list)
{
    Node       *parsetree = (Node *) lfirst(lc1);
    List       *stmt_list;
    ListCell   *lc2;

    stmt_list = pg_analyze_and_rewrite(parsetree,
                                       sql,
                                       NULL,
                                       0);
    stmt_list = pg_plan_queries(stmt_list, 0, NULL);
```

④计划语句树转换为计划执行状态树 planStateTree，并且转换后的结果放到 queryDesc->estate->es_subplanstates 中。在执行器执行过程中，根据 es_subplanstates 链表和树形结构进行执行。

```
foreach(lc2, stmt_list)
{
    Node       *stmt = (Node *) lfirst(lc2);

    if (IsA(stmt, TransactionStmt))
        ereport(ERROR,
                (errcode(ERRCODE_FEATURE_NOT_SUPPORTED),
                 errmsg("transaction control statements are not allowed within an extension script")));

    CommandCounterIncrement();

    PushActiveSnapshot(GetTransactionSnapshot());

    if (IsA(stmt, PlannedStmt) &&
        ((PlannedStmt *) stmt)->utilityStmt == NULL)
    {
        QueryDesc  *qdesc;

        qdesc = CreateQueryDesc((PlannedStmt *) stmt,
                                sql,
                                GetActiveSnapshot(), NULL,
                                dest, NULL, 0);

        ExecutorStart(qdesc, 0);
        ExecutorRun(qdesc, ForwardScanDirection, 0);
        ExecutorFinish(qdesc);
        ExecutorEnd(qdesc);

        FreeQueryDesc(qdesc);
    }
```

其中执行的四个函数：ExecutorStart/ExecutorRun/ExecutorFinish/ExecutorEnd。依次执行，将真正执行计划执行的查询。对于查询的评估，PostgreSQL 为每个收到的查询设计一个查询规划。选择正确的查询结构和数据属性的规划对执行效率至关重要，可以使用 EXPLAIN 命令查看查询规划器创建的任何查询。

（二）、源码理解

postgresql 框架下代码很多，无需关心查询过程处理细节，需要关注的是：

1. 如何添加外部函数 levenshtein_distance/jaccard_index?
2. 何处调用函数?
3. 如何评估函数开销?

(1):部分源码分析

①-src/backend/utils/fmgr/funcapi.c

用于返回集合或复合类型的函数。需要添加的 levenshtein_distance 和 jaccard_index 函数均在此处。

②-src/backend/executor/execMain.c

真正执行 qdesc 中的 query 操作时，需要经过 execMain 中的四个函数。

③-src/backend/executor/execProcnode.c

包括为给定节点类型调用适当的初始化、获取元组、清理的调度函数。

④-src/backend/executor/execScan.c

用于通用的关系扫描。传递一个节点和一个指向函数的指针，从关系中返回一个元组，然后检查条件是否合适，进行投影元组。

⑤-src/backend/executor/execTuples.c

处理可更新插槽的例程。这些用于与元组相关联的资源管理（例如，释放磁盘缓冲区中元组的缓冲区管脚，或释放临时元组占用的内存）。

⑥-src/include/catalog/pg_proc.h

系统“程序”关系（pg_proc）的定义以及关系的初始内容。可以理解为名字空间。

⑦-src/include/utils/builtins.h

对内置类型的操作声明。外部函数的声明在此处。

其中插入函数最重要的是 funcapi.c，builtins.h，pg_proc.h。

由此可以发现，可以通过如下步骤插入两个算法函数：

①funcapi.c 里面添加函数主体 `Datum levenshtein_distance(PG_FUNCTION_ARGS)`。

②builtin.h 中声明函数。

```
extern Datum levenshtein_distance(PG_FUNCTION_ARGS);
extern Datum jaccard_index(PG_FUNCTION_ARGS);
```

③pg_proc.h 中注册到名字空间。

```
DATA(insert OID = 4000 (  levenshtein_distance PGNSP PGUID 12 1 0 0 f f f t f i 2 0 20 "25 25"
_null_ _null_ _null_ _null_ levenshtein_distance _null_ _null_ _null_ ));
DESCR("levenshtein_distance");
DATA(insert OID = 4001 (  jaccard_index PGNSP PGUID 12 1 0 0 f f f t f i 2 0 700 "25 25"  _null_
_null_ _null_ _null_ jaccard_index _null_ _null_ _null_ ));
DESCR("jaccardindex");
```

(2):如何调用该函数

在 select 语句中直接使用函数名称即可，语法树会根据 pg_proc.h 中的名字空间解析函数名，然后定位该函数的内容，执行函数。所以可以写代码忽略调用过程，只关心算法实现。

(3):如何评估函数开销

postgresql 对每个 QueryDesc 提供了 Instrmentation*成员 totaltime，其中保存了一次查询任务（忽略生成语法树等操作的执行过程）开始时间、结束时间、执行元组个数等信息，可以通过\timing 调用出时间输出函数，每次执行 sql 后自动打印执行时间。

当然还有单独的 explain 体系，可以在 sql 语句加上 EXPLAIN 命令，或者 EXPLAIN ANALYZE，查看查询规划器创建的查询。

二、设计思路与实现方案

（一）、操作步骤

1.环境配置

Linux 18.0.4.2 + PostgreSQL 9.1.3 ： 下载源码解压后，先使用代码设置 configure 内容./configure --enable-depend --enable-cassert --enable-debug CFLAGS="-O0" --prefix=$HOME/pgsql

然后使用$HOME/pgsql/bin/initdb -D $HOME/pgsql/data --locale=C 将 database 初始化。之后可以创建数据库 database，使用 make && make install 编译，后使用 /home/chty627/pgsql/bin/pg_ctl -D /home/chty627/pgsql/data -l logfile start 开服务器。使用$HOME/pgsql/bin/psql postgres 连接上服务器后，create database similarity，可以建立相似度的数据库。

此处导入数据 similarity.sql

$HOME/pgsql/bin/psql -d similarity -f /home/chty627/similarity_data.sql

2.插入外部函数 levenshtein_distance 和 jaccard_index

操作方式同“源码理解”中的插入函数，此处注意注册到命名空间中的 OID 不能重复，选择了 4000 和 4001，测试无重复。

3.重启测试

测试内容同 PPT 上内容相同，先测试比较小的数据。

select jaccard_index('sunday','saturday'); select levenshtein_distance('sunday', 'sunday');测试无误后，再测试后面的大数据集。

```
select count(*) from restaurantphone rp, addressphone ap where levenshtein_distance(rp.phone, ap.phone) < 4;
select count(*) from restaurantaddress ra, restaurantphone rp where levenshtein_distance(ra.name, rp.name) < 3;
select count(*) from restaurantaddress ra, addressphone ap where levenshtein_distance(ra.address, ap.address) < 4;
select count(*) from restaurantphone rp, addressphone ap where
```

```
jaccard_index(rp.phone, ap.phone) > 0.6;
select count(*) from restaurantaddress ra, restaurantphone rp where jaccard_index(ra.name, rp.name) > 0.65;
select count(*) from restaurantaddress ra, addressphone ap where jaccard_index(ra.address, ap.address) > 0.8;
```

（二）、设计思路

按照以上内容测试无误后，则可以开始进行 funcapi.c 中的主函数编辑。设计思路则为：

①根据 PG_GETARG_DATUM(0/1)获取函数中的两个参数，text_to_cstring()转换为字符串。

②对两个字符串进行运算。

③测试时间并优化

三、关键代码说明

1.levenshtein_distance

2.jaccard_index

四、实验与结果

1.levenshtein_distance

①栈临时变量

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where levenshtein_distance(rp.phone, ap.phone) < 4;

 count
-------
  3252
(1 row)

Time: 12344.738 ms
similarity=#
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where levenshtein_distance(ra.name, rp.name) < 3;
 count
-------
  2313
(1 row)

Time: 20156.379 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where levenshtein_distance(ra.address, ap.address) < 4;
 count
-------
  2874
(1 row)

Time: 38353.983 ms
```

②static 静态数组 + 初始化

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where levenshtein_distance(rp.phone, ap.phone) < 4;
 count
-------
  3252
(1 row)

Time: 11711.933 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where levenshtein_distance(ra.name, rp.name) < 3;
 count
-------
  2313
(1 row)

Time: 20642.471 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where levenshtein_distance(ra.address, ap.address) < 4;
 count
-------
  2874
(1 row)

Time: 40932.444 ms
```

2.jaccard_index

①暴力排序法

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 12200.501 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2398
(1 row)

Time: 24274.595 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2186
(1 row)

Time: 44585.124 ms
```

②static 静态数组 + memset + M = 53

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 6860.428 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2391
(1 row)

Time: 10863.338 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2196
(1 row)

Time: 14833.912 ms
```

③static 静态数组 + memset + M = 103

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 6779.537 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2391
(1 row)

Time: 10415.210 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2196
(1 row)

Time: 13853.398 ms
```

④static 静态数组 + memset + M = 131

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 7015.794 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2391
(1 row)

Time: 10530.191 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2196
(1 row)

Time: 14077.952 ms
```

⑤ static 静态数组 + memset + M = 53 + hash2c（两个字符 hash）

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 6964.955 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2398
(1 row)

Time: 9947.087 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2186
(1 row)

Time: 14070.510 ms
```

⑥static 静态数组 + memset + merge

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 15493.826 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2397
(1 row)

Time: 30419.955 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2186
(1 row)

Time: 46200.839 ms
```

⑦static 静态数组 + memset + merge （修改）

```
similarity=# \timing
Timing is on.
similarity=# select count(*) from restaurantphone rp, addressphone ap where jacc
ard_index(rp.phone, ap.phone) > 0.6;
 count
-------
  1653
(1 row)

Time: 14392.393 ms
similarity=# select count(*) from restaurantaddress ra, restaurantphone rp where
 jaccard_index(ra.name, rp.name) > 0.65;
 count
-------
  2397
(1 row)

Time: 28698.061 ms
similarity=# select count(*) from restaurantaddress ra, addressphone ap where ja
ccard_index(ra.address, ap.address) > 0.8;
 count
-------
  2186
(1 row)

Time: 42585.701 ms
```

五、性能优化

1.levenshtein_distance

```
similarity=# EXPLAIN  ANALYZE select count(*) from restaurantaddress ra, restaur
antphone rp where levenshtein_distance(ra.name, rp.name) < 3;
                                                              QUERY PLAN

-----------------------------------------------------------------------------
-----------------------------------------------------------
 Aggregate  (cost=14214.88..14214.89 rows=1 width=0) (actual time=21384.912..213
84.912 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..13571.75 rows=257250 width=0) (actual time=43.12
0..21382.907 rows=2313 loops=1)
         Join Filter: (levenshtein_distance((ra.name)::text, (rp.name)::text) <
3)
         ->  Seq Scan on restaurantaddress ra  (cost=0.00..37.25 rows=1225 width
=68) (actual time=0.040..5.249 rows=2439 loops=1)
         ->  Materialize  (cost=0.00..30.45 rows=630 width=118) (actual time=0.0
00..0.190 rows=2463 loops=2439)
               ->  Seq Scan on restaurantphone rp  (cost=0.00..27.30 rows=630 wi
dth=118) (actual time=0.011..0.560 rows=2463 loops=1)
 Total runtime: 21385.101 ms
(7 rows)
```

```
similarity=# EXPLAIN  ANALYZE select count(*) from restaurantphone rp, addressph
one ap where levenshtein_distance(rp.phone, ap.phone) < 4;
                                                                  QUERY PLAN

------------------------------------------------------------------------------
------------------------------------------------------
 Aggregate  (cost=13647.43..13647.44 rows=1 width=0) (actual time=11703.012..117
03.012 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..13030.03 rows=246960 width=0) (actual time=0.057
..11701.213 rows=3252 loops=1)
         Join Filter: (levenshtein_distance((rp.phone)::text, (ap.phone)::text)
< 4)
         ->  Seq Scan on addressphone ap  (cost=0.00..35.76 rows=1176 width=68)
(actual time=0.015..3.754 rows=2429 loops=1)
         ->  Materialize  (cost=0.00..30.45 rows=630 width=118) (actual time=0.0
00..0.176 rows=2463 loops=2429)
               ->  Seq Scan on restaurantphone rp  (cost=0.00..27.30 rows=630 wi
dth=118) (actual time=0.010..0.791 rows=2463 loops=1)
 Total runtime: 11703.204 ms
(7 rows)
```

```
similarity=# EXPLAIN  ANALYZE select count(*) from restaurantaddress ra, address
phone ap where levenshtein_distance(ra.address, ap.address) < 4;
                                                                  QUERY PLAN

------------------------------------------------------------------------------
------------------------------------------------------
 Aggregate  (cost=26486.95..26486.96 rows=1 width=0) (actual time=40764.658..407
64.658 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..25286.45 rows=480200 width=0) (actual time=0.086
..40762.257 rows=2874 loops=1)
         Join Filter: (levenshtein_distance((ra.address)::text, (ap.address)::te
xt) < 4)
         ->  Seq Scan on restaurantaddress ra  (cost=0.00..37.25 rows=1225 width
=68) (actual time=0.009..5.488 rows=2439 loops=1)
         ->  Materialize  (cost=0.00..41.64 rows=1176 width=68) (actual time=0.0
00..0.176 rows=2429 loops=2439)
               ->  Seq Scan on addressphone ap  (cost=0.00..35.76 rows=1176 widt
h=68) (actual time=0.008..0.534 rows=2429 loops=1)
 Total runtime: 40764.816 ms
(7 rows)
```

2.jaccard_index

```
similarity=# EXPLAIN  ANALYZE select count(*) from restaurantphone rp, addressph
one ap where jaccard_index(rp.phone, ap.phone) > 0.6;
                                                                  QUERY PLAN

------------------------------------------------------------------------------
------------------------------------------------------
 Aggregate  (cost=13647.43..13647.44 rows=1 width=0) (actual time=7037.450..7037
.450 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..13030.03 rows=246960 width=0) (actual time=0.016
..7036.649 rows=1653 loops=1)
         Join Filter: (jaccard_index((rp.phone)::text, (ap.phone)::text) > 0.6::
double precision)
         ->  Seq Scan on addressphone ap  (cost=0.00..35.76 rows=1176 width=68)
(actual time=0.005..1.733 rows=2429 loops=1)
         ->  Materialize  (cost=0.00..30.45 rows=630 width=118) (actual time=0.0
00..0.158 rows=2463 loops=2429)
               ->  Seq Scan on restaurantphone rp  (cost=0.00..27.30 rows=630 wi
dth=118) (actual time=0.004..0.514 rows=2463 loops=1)
 Total runtime: 7037.584 ms
(7 rows)
```

```
similarity=# EXPLAIN  ANALYZE select count(*) from restaurantaddress ra, restaur
antphone rp where jaccard_index(ra.name, rp.name) > 0.65;
                                                                  QUERY PLAN

------------------------------------------------------------------------------
------------------------------------------------------
 Aggregate  (cost=14214.88..14214.89 rows=1 width=0) (actual time=10562.367..105
62.367 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..13571.75 rows=257250 width=0) (actual time=11.34
0..10561.223 rows=2391 loops=1)
         Join Filter: (jaccard_index((ra.name)::text, (rp.name)::text) > 0.65::d
ouble precision)
         ->  Seq Scan on restaurantaddress ra  (cost=0.00..37.25 rows=1225 width
=68) (actual time=0.010..2.265 rows=2439 loops=1)
         ->  Materialize  (cost=0.00..30.45 rows=630 width=118) (actual time=0.0
00..0.166 rows=2463 loops=2439)
               ->  Seq Scan on restaurantphone rp  (cost=0.00..27.30 rows=630 wi
dth=118) (actual time=0.006..0.552 rows=2463 loops=1)
 Total runtime: 10562.521 ms
(7 rows)
```

```
similarity=# EXPLAIN  ANALYZE select count(*) from restaurantaddress ra, address
phone ap where jaccard_index(ra.address, ap.address) > 0.8;
                                                                  QUERY PLAN

------------------------------------------------------------------------------
------------------------------------------------------
 Aggregate  (cost=26486.95..26486.96 rows=1 width=0) (actual time=14007.411..140
07.411 rows=1 loops=1)
   ->  Nested Loop  (cost=0.00..25286.45 rows=480200 width=0) (actual time=0.033
..14006.082 rows=2196 loops=1)
         Join Filter: (jaccard_index((ra.address)::text, (ap.address)::text) > 0
.8::double precision)
         ->  Seq Scan on restaurantaddress ra  (cost=0.00..37.25 rows=1225 width
=68) (actual time=0.010..2.746 rows=2439 loops=1)
         ->  Materialize  (cost=0.00..41.64 rows=1176 width=68) (actual time=0.0
00..0.165 rows=2429 loops=2439)
               ->  Seq Scan on addressphone ap  (cost=0.00..35.76 rows=1176 widt
h=68) (actual time=0.007..0.552 rows=2429 loops=1)
 Total runtime: 14007.681 ms
(7 rows)
```

六、其他

（一）、分工

本次 PJ2 由两人合作完成:

1.宁晨然 17307130178:

①环境配置及试运行②框架代码阅读和整体分析③levenshtein_distance/jaccard_index 算法和优化思路④结果测试与分析

2.田睿 17300750084

①环境搭建和算法剖析②插入外部函数解析③levenshtein_distance/jaccard_index 算法并多种优化方式

（二）、实验难点

本次实验在源码阅读、环境配置和算法优化上耗费很多时间。

源码阅读上，postgresql 的框架很大，刚上手很难找到我们需要的函数、需要修改的内容。在阅读了网络上一些框架解读后，发现只需要从 sql 的执行过程入手即可，也无需关心算法语法树的优化过程，而外部函数的插入有固定的模式。

环境配置，一开始装入数据库后，并且插入函数后，发现无法找到对应 OID 的函数，也遇到了函数执行时无法连接到数据库的情况，还有数据库无法连接（开服务器问题）。解决方法是删除整个数据库，从 configure 重新开始、重启电脑等，对于函数执行过程中无法连接数据库，是因为函数体中有不规范的 C 语言，例如数组长度为变量、int 声明在 for 循环初始化中等，有许多 C 语言规范的问题导致了连接中断。

算法优化，对于第一个算法很简单，但是第二个由于牵涉到集合操作，而 C 语言特别不擅长处理动态数组的问题，也没有可行的容器进行集合删去重复、并交操作，所以采取了多种算法思路，中间花费了很多时间。

（三）、实验感想

本次实验总体而言很顺利，两人合作分工进行代码剖析、讨论，有时一人没有看懂的框架另一人提纲挈领的点拨能加快阅读代码的速度，并且讨论算法时两人会有更多不同的思路，经过讨论再到实现就有了不同的方案的解读。

算法实现方面使用了很多数据结构和数据库的综合内容，是对学过的知识的活学活用，加深了对于数据库本学期内容的理解。

[reference]

[1]:https://www.cnblogs.com/flying-tiger/p/8039055.html

[2]: https://blog.csdn.net/tencupofkaiwater/article/details/80967948