Step 1

Query1:

Relational Algebra

SQLite:

CREATE TABLE friends (uid1 integer, uid2 integer);

Run Time: real 0.196 user 0.000000 sys 0.000000

INSERT INTO friends

SELECT smallA.uid1, smallA.uid2 FROM

(SELECT \* FROM followers AS A WHERE A.uid1 < A.uid2) AS smallA

JOIN

(SELECT \* FROM followers AS B WHERE B.uid1 > B.uid2) AS smallB

ON smallA.uid1 = smallB.uid2 and smallA.uid2 = smallB.uid1;

Run Time: real 393.437 user 205.016000 sys 60.644000

SELECT count(\*) FROM friends;

Note: from the assignment, it requires us provide both count and list of true friends. So, we insert the results into a new table first then produce the count.

Query2:

Relational Algebra

SQLite

SELECT c.uid, sum(c.co) AS s FROM (

SELECT A.uid1 as uid, -count(\*) AS co FROM followers AS A GROUP BY A.uid1

UNION

SELECT B.uid2 as uid, count(\*) AS co FROM followers as B GROUP BY B.uid2

) c

GROUP BY c.uid ORDER BY s DESC LIMIT 10;

Step 3:

Research Results:

Total number of true friends: 21776094

Total number of distinct users: 11316811

List of top 10 celebrities:

uid: 5994113 , in-out: 564220, indegree: 564512, outdegree: 292

uid: 7496 , in-out: 344850, indegree: 350885, outdegree: 6035

uid: 1349110 , in-out: 340491, indegree: 341963, outdegree: 1472

uid: 1629776 , in-out: 170111, indegree: 172231, outdegree: 2120

uid: 8121005 , in-out: 155933, indegree: 155967, outdegree: 34

uid: 2041453 , in-out: 152069, indegree: 152689, outdegree: 620

uid: 797152 , in-out: 118752, indegree: 118826, outdegree: 74

uid: 6623784 , in-out: 115819, indegree: 116002, outdegree: 183

uid: 645019 , in-out: 107639, indegree: 107914, outdegree: 275

uid: 3403 , in-out: 97931, indegree: 102877, outdegree: 4946

Summary:

The total number of true friends is almost twice as large as the distinct users. That shows in average every user is following 2 other users and is followed by those two users (in average 2 true friends per user). From the top 10 celebrities list we can see that the in-out degree difference is decreasing very fast from top 1 to 10. This means the number of followers for a user will increase very slowly at the beginning, but once the user has a certain number of followers (i.e. become famous) it will increase very fast.

Performance Results:

SQLite:

Query 1: 0.196s + 393.437s + 3.535s = 397.168s = 6.619 minutes

Query 2: 216.425s = 3.607 minutes

C:

Query 1: 1:09.45 minutes = 69s

Query2: 1:18.54 minutes = 78s

Our implementation is faster. Because for query 1, the join part, our implementation sorted followers by uid1 then by uid2 (followers1) and sorted followers by uid2 then by uid1(followers2). Then we merge the two followers. We only need to loop over each followers table once, because for sorted data if (1, 3) from followers doesn’t match with (5, 1) from followers2 we can move to next tuple in followers. The query for Sqlite does not using sorted data, if (1, 3) doesn’t match (5, 1) maybe the next tuple from followers2 is (3, 1). It need to loop over the table couple times in worst case.

We also did one optimization on both Sqlite and C, because we are doing self-join, join (1,3) and (3,1) is same as join (3, 1) and (1, 3). Before join we select all the tuple with uid1 < uid2 from followers1 and uid1 > uid2 from followers2. This improve the performance a lot, before doing that Sqlite need 800s to get results.

For query2, we tried using join to implement Sqlite query, but it took 1500s to get results. Because Sqlite doesn’t support full out join, by using a single join query we can only get the joined tuple with uid in both uid1 and uid2 column. To use join to get results, we need to union the results of joined tuple with uid in both uid1 and uid2 column with tuple with uid in only uid1 or uid2 column. Instead of doing that, we group by uid1 column and group by uid2 column and get the count for each group. Union the results and then group by id again. Now each group will have an uid with indegree, outdegree. This is easier to get the results. But group by also takes time if the data is not sorted by grouping key.

So, in our implementation, we sorted data the same as we did in query1. Then we did a count on each row with same id and write to disk. (indegree.dat, outdegree.dat). We read from the sorted data line by line, we the indegree.dat and outdegree.dat are still sorted by uid. Then when we do the merge part, we only need to loop over each file once.

Following are the output from SQLite and C

SQLite Result Query 1:

CREATE TABLE friends (uid1 integer, uid2 integer);

Run Time: real 0.196 user 0.000000 sys 0.000000

INSERT INTO friends

SELECT smallA.uid1, smallA.uid2 FROM

(SELECT \* FROM followers AS A WHERE A.uid1 < A.uid2) AS smallA

JOIN

(SELECT \* FROM followers AS B WHERE B.uid1 > B.uid2) AS smallB

ON smallA.uid1 = smallB.uid2 and smallA.uid2 = smallB.uid1;

Run Time: real 393.437 user 205.016000 sys 60.644000

SELECT count(\*) FROM friends;

21776094

Run Time: real 3.535 user 0.068000 sys 0.568000

SQLite Result Query 2:

SELECT c.uid, sum(c.co) AS s FROM (

SELECT A.uid1 as uid, -count(\*) AS co FROM followers AS A GROUP BY A.uid1

UNION

SELECT B.uid2 as uid, count(\*) AS co FROM followers as B GROUP BY B.uid2

) c

GROUP BY c.uid ORDER BY s DESC LIMIT 10;

5994113|564220

7496|344850

1349110|340491

1629776|170111

8121005|155933

2041453|152069

797152|118752

6623784|115819

645019|107639

3403|97931

Run Time: real 216.425 user 199.192000 sys 10.260000

My Query1 result:

b2210-30:/s/csc443/yufangzh/A1P3/A1part3$ /usr/bin/time -v ./query1 data1.dat 16384

total number match 21776094

Command being timed: "./query1 data1.dat 16384"

User time (seconds): 21.04

System time (seconds): 3.44

Percent of CPU this job got: 35%

Elapsed (wall clock) time (h:mm:ss or m:ss): 1:09.45

Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0

Average stack size (kbytes): 0

Average total size (kbytes): 0

Maximum resident set size (kbytes): 1265276

Average resident set size (kbytes): 0

Major (requiring I/O) page faults: 0

Minor (reclaiming a frame) page faults: 16588

Voluntary context switches: 122378

Involuntary context switches: 2886

Swaps: 0

File system inputs: 0

File system outputs: 5673512

Socket messages sent: 0

Socket messages received: 0

Signals delivered: 0

Page size (bytes): 4096

Exit status: 0

Total number of true friends is 21776094

My Query2 Result:

b2210-30:/s/csc443/yufangzh/A1P3/A1part3$ /usr/bin/time -v ./query2 data1.dat 16384

distinct user number: 11316811

uid: 5994113 , in-out: 564220, indegree: 564512, outdegree: 292

uid: 7496 , in-out: 344850, indegree: 350885, outdegree: 6035

uid: 1349110 , in-out: 340491, indegree: 341963, outdegree: 1472

uid: 1629776 , in-out: 170111, indegree: 172231, outdegree: 2120

uid: 8121005 , in-out: 155933, indegree: 155967, outdegree: 34

uid: 2041453 , in-out: 152069, indegree: 152689, outdegree: 620

uid: 797152 , in-out: 118752, indegree: 118826, outdegree: 74

uid: 6623784 , in-out: 115819, indegree: 116002, outdegree: 183

uid: 645019 , in-out: 107639, indegree: 107914, outdegree: 275

uid: 3403 , in-out: 97931, indegree: 102877, outdegree: 4946

Command being timed: "./query2 data1.dat 16384"

User time (seconds): 22.25

System time (seconds): 3.75

Percent of CPU this job got: 33%

Elapsed (wall clock) time (h:mm:ss or m:ss): 1:18.54

Average shared text size (kbytes): 0

Average unshared data size (kbytes): 0

Average stack size (kbytes): 0

Average total size (kbytes): 0

Maximum resident set size (kbytes): 410976

Average resident set size (kbytes): 0

Major (requiring I/O) page faults: 0

Minor (reclaiming a frame) page faults: 19401

Voluntary context switches: 106368

Involuntary context switches: 3052

Swaps: 0

File system inputs: 0

File system outputs: 6104304

Socket messages sent: 0

Socket messages received: 0

Signals delivered: 0

Page size (bytes): 4096

Exit status: 0