School of Computer Science
Dr. Ying Zhou

COMP5338: Advanced Data Models Sem. 2/2020

Neo4j Assignment

02.10.2020

1 Introduction

In this assignment, you will demonstrate that you know how to import semi-structured data into Neo4j as property graph by creating nodes and relationships between them. You also demonstrate that you are able to query the graph to explore various aspects of the real world data.

The data set you will work with has the same format as the one you worked in MongoDB assignment. It consists of tweet objects downloaded from **Twitter** using `Twitter API`. You are asked to import the data as property graph in Neo4j and to implement a set of target queries. A practice data set is released along with the assignment instruction. The expected answer of each query for the practice data set is given in the assignment instruction to help you self check the correctness of the implementation. A test data set will be used by markers to mark the implementation. The practice and the test data set are of similar size and format. It is important that you do not redistribute the data set used in the assignment to any party outside this course. Doing so may violate Twitter's content distribution policy.

2 Data set

The practice data set contains a single JSON file downloaded using keyword search of twitter's `standard search` API. The JSON file contains around 10K downloaded `tweet` objects. We omit the field description of tweet object here. They can be found in the MongoDB assignment instruction.

Each tweet has an unique `ID`. The data set contains more tweet IDs than the number of Tweet objects in the JSON file. There are tweet `IDs` without a corresponding object. Such ID usually appears as the `replyto_id` or `retweet_id` of another tweet.

A tweet could be a *general tweet*, a *retweet* or a *reply* to a tweet. We can tell the type of a tweet if the corresponding object is in the data set. For tweet with only an ID in the data set, the type is *unknown*.

There could be *reply* to a *retweet* or a *reply*; and *retweet* of a *reply*. Some queries in this assignment explore the tree structure formed by *reply* and retweet relations. For convenience, we use the following tree data structure terminologies to refer to tweets in different part of the tree structure:

- `parent` refers to a tweet that receives a reply or retweet
- `child` refers to a retweet or reply of another tweet
- ancestor refers to any high level tweet in the `parent` hierarchy. An ancestor could be a parent, a grand parent or a tweet further up the parent hierarchy of another tweet.
- `descendant` refers to any low level tweet in the child hierarchy. A descendant could be a child, a grand child, or a tweet further down the child hierarchy of another tweet.
- `root` refers to the tweet that does not have known parent in the data set. Note that in theory a `root` should be a general tweet. But since our data set only contains a tiny subset of the entire tweet graph, it is possible that the root tweet is of unknown type with only an ID in the data set.
- `leaf` refers to the tweet that does not have any children.

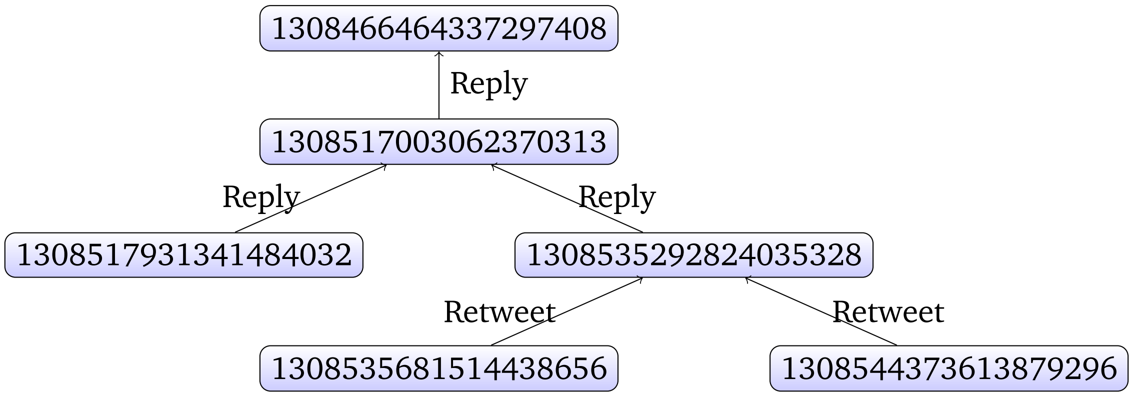


Figure 1: An Example Tweets Tree formed by Reply and Retweet

Figure 1 shows an example tree consists of six tweets connected by *reply* and *retweet* edges. The actual tweets objects in the data set are shown in next page. You may notice is that the data set contains only five tweet objects. Tweet “1308466464337297408” is the `root` of the tree and does not have a corresponding object in the data set. But the ID appears as the `replyto_id` of the tweet object “1308517003062370313”. This `root` tweet does not have a known parent in the data set. The tree has three leaf tweets: “1308517931341484032”, “1308535681514438656” and “1308544373613879296”. Tweet “1308535292824035328” has two `ancestors`: the `root` and “1308517003062370313”; and two `descendants`: tweet “1308535681514438656” and “1308544373613879296”.

```
{
    "id" : NumberLong(1308517003062370313),
    "created_at" : "2020-09-22 21:22:19",
    "text" : ""We will definitely be standing on the shoulders of ’Schitt’s Creek.’"\n\n\"The Simpsons\" writer @mrtimlong says new C... https://t.co/SS42H32o2F",
    "user_id" : 14133037,
    "replyto_id" : NumberLong(1308466464337297408),
    "replyto_user_id" : 14133037,
    "user_mentions" : [ { "id" : 227771301, "indices" : [ 94, 104]}]
}
{
    "id" : NumberLong(1308517931341484032),
    "created_at" : "2020-09-22 21:26:00",
    "text" : "@dfriend @mrtimlong Absolutely they will get more eyes on them just by virtue of what Schitt’s Creek accomplished.... https://t.co/5fJPJbznP4",
    "user_id" : 289496163,
    "replyto_id" : NumberLong(1308517003062370313),
    "replyto_user_id" : 14133037,
    "user_mentions" : [ {"id":14133037,"indices":[0, 8]}, {"id":227771301, "indices":[9,19]}]
}
{
    "id" : NumberLong(1308535292824035328),
    "created_at" : "2020-09-22 22:34:59",
    "text" : "I don’t think Canada can ever be second-guessed the way it might’ve been in the past.\"\n\nThe \"Schitt’s Creek\"... https://t.co/Hn9bv1eB5B",
    "user_id" : 14133037,
    "replyto_id" : NumberLong(1308517003062370313),
    "replyto_user_id" : 14133037
}
{
    "id" : NumberLong(1308544373613879296),
    "created_at" : "2020-09-22 23:11:04",
    "text" : "RT @dfriend: "I don’t think Canada can ever be second-guessed ...",
    "user_id" : 824157,
    "retweet_id" : NumberLong(1308535292824035328),
    "retweet_user_id" : 14133037,
    "user_mentions" : [ {"id" : 14133037,"indices" : [ 3, 11]}]
}
{
    "id" : NumberLong(1308535681514438656),
    "created_at" : "2020-09-22 22:36:32",
    "text" : "RT @dfriend: "I don’t think Canada ...",
    "user_id" : 55610483,
    "retweet_id" : NumberLong(1308535292824035328),
    "retweet_user_id" : 14133037,
    "user_mentions" : [ {"id" : 14133037,"indices" : [ 3, 11]}]
}
```

3 Query workload

- `[Q1]` Find out the number of *general* *tweets* that do not have a reply nor a retweet in the data set. This question is the same as `Q6` in MongoDB assignment. And the sample answer for the practice data set is: 911

- `[Q2]` Find out the top 5 hashtags sorted by their occurrence in *general* or *reply* tweets. We do not count retweet, which has the same textual content as the `parent tweet`. This question is the same as `Q2` in MongoDB assignment. The sample answers for the practice data set are:

Venus	Quran	QURAN	space	NASA
57	18	14	8	7

  Note that the order does not matter if a few hashtags have the same occurrence number. The 5th tag in the sample result could be 'phosphine',which also occurs 7 times.

- `[Q3]` Find out the most popular tweet ranked by the number of `descendants` in the database. Print out the tweet id and the number of descendants.

  The sample answer for the practice data set are:

Tweet ID	Descendant Count
1306104147209584640	5159

- `[Q4]` Find out the most popular tweet ranked by the number of unique users as author of its `descendant` tweets. Print out the tweet id and the number of authors.

  The sample answer for the practice data set are:

Tweet ID	User Count
1306104147209584640	5159

- `[Q5]` Find out the longest discussion path in the data set. A discussion path is a sequence of tweets from a `root` tweet to a leaf tweet along the *reply* and *retweet* edges. Print out the path length and the path as a list of tweet ids. If there are multiple paths with the same maximum length, you only need to print one of them as the result.

  The sample answer for the practice data set are:

path length	Tweet IDs in Path
9	[1306516021499494402, 1306519149175169024, 1306519571323527171, 1306520043132350465, 1306520148052905985, 1306520712761413634, 1306522086215544839, 1306526400334110720, 1306528866400313345, 1306529423470985218]

- `[Q6]` Some tweets mention users in their texts. This information is stored in the field `mentions`. There are in general two cases a user might be mentioned:

  – A *retweet* or *reply* may mention its ancestors' authors. In particular, a retweet may automatically mention its ancestors' authors. An example can be seen from tweet "1308517931341484032", which mentions its' parent tweet's author "14133037".

  – A tweet may mention a user that is not the author of any ancestor tweet but is related with the content. An example can be seen in tweet "1308517003062370313", which mentions user "227771301" (@mrtimlong). This user is not the author of any ancestor tweet.

  We are interested in the second case and would like to find out the top user with most mentions not from its descendant tweets. Print out the user id and the number of time it is mentioned by a tweet that is not a descendant. If there are multiple top users with the same mention count, you only need to print one as the result.

  The sample answers for the practice data set are:

user id	count
1233245849427136513	363

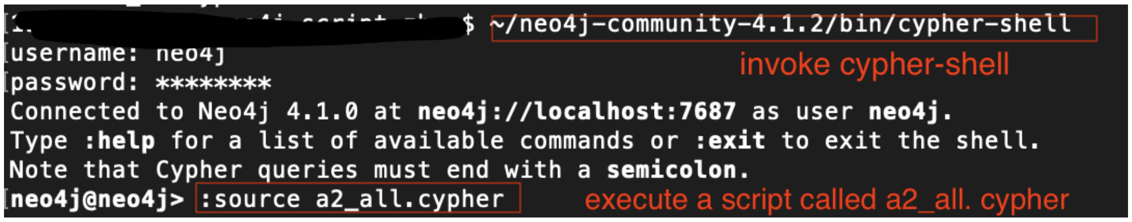
4 Implementation Requirements

The data loading, query and clean up should be implemented as three separate `cypher` shell scripts:

- `setup.cypher`: this script contains all statements related with loading the JSON file and constructing the graph.
- `queries.cypher`: this script contains query implementations. Running this script should print the result of each query in correct order. There is not much control on neo4j's output format. You are asked to to specify "verbose" format. You should only print the target query result and the query sequence number should be included in one of the result column for easy marking. A sample output for the practice data set is given in next page. Q5 results are shortened to fit in the page.
- `cleanup.cypher`: this script contains statements to clean up the graph including the indexes created.

Three sample scripts are released along with the assignment instruction. They show the format of a `cypher` script. `setup_sample.cypher` loads the "persons.json" file used in week6 lab into a simple graph. The script contains two statements: the first one creates an index on `name` field of `Person` node, the second one loads data from local file. Each statement ends with a semicolon. `queries_sample.cypher` runs two queries. Each query is implemented as a statement ending with a semicolon. `cleanup_sample.cypher` removes all nodes/relationships/indexes.

The script can be executed using `:source` command inside the `cypher-shell` or be invoked from command line. Figure 2 shows how to execute it inside `cypher-shell`.



```
[redacted] $ ~/neo4j-community-4.1.2/bin/cypher-shell
username: neo4j
password: ********
Connected to Neo4j 4.1.0 at neo4j://localhost:7687 as user neo4j.
Type :help for a list of available commands or :exit to exit the shell.
Note that Cypher queries must end with a semicolon.
neo4j@neo4j> :source a2_all.cypher
```

Figure 2: Run Cypher Script inside Cypher-shell

In MacOS, the `cat` command can be used to invoke cypher script from command line:

```
$neo4jhome> cat a2sample.cypher | bin/cypher-shell -u neo4j -p <password> --format verbose
```

In Windows, the `type` command can be used to invoke cypher script from command line:

```
$neo4jhome> type a2sample.cypher | bin/cypher-shell.bat -u neo4j -p <password> --format verbose
```

```
+------------+
| Q1_results |
+------------+
| 911        |
+------------+
+-------------------------+
| Q2_tag       | tag_count |
+-------------------------+
| "Venus"      | 57        |
| "Quran"      | 18        |
| "QURAN"      | 14        |
| "space"      | 8         |
| "NASA"       | 7         |
+-------------------------+
+--------------------------------------+
| Q3_root_id          | descendant_count |
+--------------------------------------+
| 1306104147209584640 | 5159             |
+--------------------------------------+
+--------------------------------+
| Q4_root_id          | user_count |
+--------------------------------+
| 1306104147209584640 | 5159       |
+--------------------------------+
-----------------------------------------------------------------+
| Q5_length | tweets                                              |
+-----------------------------------------------------------------+
| 9         |  [1306516021499494402, 1306519149175169024, ...]  |
+-----------------------------------------------------------------+
+-----------------------------------+
| user_id             | mention_count |
+-----------------------------------+
| 1233245849427136513 | 363           |
+-----------------------------------+
```

5 Deliverable and Submission Guidelines

The final deliverable of this assignment consists of two elements:

- A zip file with all three `cypher` shell scripts due for submission on Canvas by **week 8 Monday October 19, 2020 23:59pm**.

- A **5-10 minutes** zoom demo with your tutor in week 8. In the demo the **tutor** will run your submitted script on a test data set of similar size and of the same format.

6 Mark Distribution

The target query is worth 1.5 points each. The marking is based on whether or not your query produces the correct results on the test data set during the demo. The test data set is collected using the same Twitter API with different search term(s). There is no intentionally added corner case(s) in the test data set.
There is 1 point allocated for script conforming to the required format. Your script needs to produce at least the correct result following the implementation requirements to receive the format point.