CS2901 Networking and Communication **December 5, 2019**

Lab 5: Grakn

Diego Cánez University of Engineering and Technology

I Introduction

Grakn is a service for knowledge-oriented systems, that manages informations through as an entity-relationship model. It includes native algorithms for performing inference on the data and allows for more flexible storage of data.

Trying out Grakn, I was reminded of an area in Artificial Intelligence called Knowledge Representation and Reasoning (KR&R) that caught my attention some months ago. It was interesting to try out some of those concepts through Grakn's intuitive command line utilities and integration with popular programming languages like Python.

Although the assignment required only to create a social network schema and play with it, I wanted to test Grakn's power in real-life data. With this in mind, I challenged myself to build what I called a "potential-friends-network". The concepts is very simple: If I follow someone and that person also follows me back, than I'm (probably) his friend. If that person is a friend (mutual-follow) with another person, then there are considerable chances I can get to know that person.

The question was, uninevitable: Who could I be friends with? Could I possibly meet a prestiguous academic in the research areas I enjoy?

II Setup

II.1 Twitter's API

Setting up a Twitter's Developer Account is fairly easy, but it gets a little bit more complicated once you have to create an "App". Apps provide the credentials you need for your application but require a Website with SSL certification for authentication. My go-to choice was a DigitalOcean Ubuntu 18 Droplet, a Namecheap domain and a Comodo SSL Certificate.

II.2 Grakn Schema

For convinience, I kept the Grakn Schema as simple as I could. This was the result:

The Grakn Schema for Twitter Followers `twitter/schema.gql`

```
define

################
## ATTRIBUTES ##
################

  name sub attribute,
    datatype string;
```

```
  full-name sub name;

    uuid sub attribute,
        datatype string;

  person sub entity,
    has full-name,
    key uuid;


################
## FRIENDSHIP ##
################

  friendship sub relation,
    relates friend;

  person sub entity,
    plays friend;
```

To set Grakn first run `grakn server start` and then `grakn console --keyspace twitter --file twitter/schema.gql` to load the schema.

II.3 Python Scripting

Twitter's API and some related utilities were handled through `twitter-controller.py`. An example of this was the `get_next_mutuals` function, which returned the mutual-follows of a given node by taking the intersection between the people that follow a certain person a the people that are followed by that person.

The `graph.py` file handled the graph algorithms such as generic implementations DFS and BFS. These were depth-limited because the Twitter Free API only allowed for 15 requests in 15 minutes, which was basically nothing.

The `main.py` file handled the connection with the Grakn Server and defined some functions to be ran as the DFS traversed the graph formed previously. `grakn-write` binded some local parameters and took a node instance and committed it to Grakn.

III Results

Inserting Friendship relations

```
INSERTING FRIENDSHIPS
Inserting: Node(1188655448641200128, dgcnz) -> Node(795881768368803840, Arturo Deza)    SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(1190347314151358464, Johannes Mahr)    SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(1083128336115740673, Dick Dubbelde)    SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(1176942100203458560, Marius C. Vollberg)       SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(875630552329535488, Valentina) SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(923580726754324484, Lynn Sörensen)     SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(981258154750160896, Patrik Andersson)  SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(760986968855425025, Katherine Hermann) SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(711204340124540928, Julian De Freitas) SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(778962856624652288, Carlos R. Ponce)   SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(825460118686924805, Thomas O'Connell)  SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(875430125130596353, Fenil)     SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(1033563949525094402, Nicholas Blauch)  SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(1098366249204097030, JohnMark Taylor)  SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(971175623539359745, Sophia Sanborn)    SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(13348, Robert Scoble)  SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(3243701798, Dina Popovkina, PhD)       SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(25995304, talia konkle)        SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(278310444, Jaspers W.  Huanay) SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(636058159, Venkata Suresh)     SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(2842853434, Hamid EBZD)        SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(22773313, Rebecca Wissinger)   SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(43040835, Akshay Jagadeesh)    SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(37095498, Amir)       SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(828106307412643840, Hossein Mehrzadfar)        SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(29274213, Chaz Firestone)      SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(4009064038, Emma Kate Ward)    SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(170552935, Vijay V)  SUCCESS
Inserting: Node(795881768368803840, Arturo Deza) -> Node(966153758496493568, Robert Aduviri @ )        SUCCESS
```

Fig. 1: A sample of the mutual-followers network.

My potential acquaintances

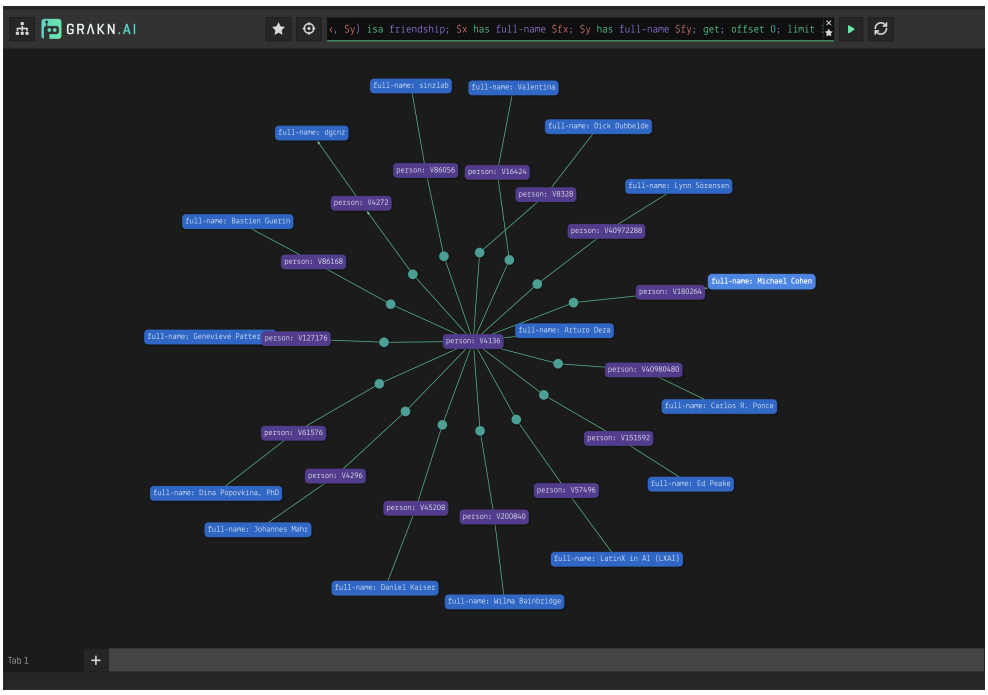

Fig. 2: A sample of the mutual-followers network.

Overall, the result was interesting and the following figure of the Grakn Workbase shows such work.

My potential acquaintances graph was promising. A few months ago, I had met a PhD candidate in AI called Arturo Deza and such connection is clearly seen as the strongest one. Probably because of his success, he had a lot of mutual-follows (such quantity can't be clearly seen because of the `limit 30` query) and that meant that I *could*, in theory, be friends with those people. It was a fun experiment, but the Twitter API Request Rate Limit didn't allow me to progress further than this.

IV Appendix

Graph Utilities and Algorithms

```
from collections import deque
import math


class Node:
    def __init__(self, id_, name_):
        self.id = id_
        self.name = name_
        self.neighbors = []

    def __eq__(self, other):
        if other is None:
            return False
        return self.id == other.id

    def __hash__(self):
        return hash(self.id)

    def __str__(self):
        return f"Node({self.id}, {self.name})"

    def __repr__(self):
        return f"Node({self.id}, {self.name})"


def bfs(u, get_neighbors, f, max_depth=math.inf):
    depth = 0
    queue = deque()
    visited = set()
    queue.append(u)
    queue.append(None)

    while (len(queue) > 0 and depth < max_depth):
        x = queue.popleft()
        if x is None:
            depth += 1
            queue.append(None)
            continue
        visited.add(x)
        for v in get_neighbors(x):
            if v not in visited and v not in queue:
                queue.append(v)
        f(x)

    return visited
```

```
46
47
48 def dfs(u, get_neighbors, max_depth=math.inf, f=lambda *args: None):
49     if max_depth is None:
50         max_depth = math.inf
51     visited = set()
52     visited.add(u)
53     f(None, u)
54     u.neighbors = dfs_traverse(u, get_neighbors, visited, max_depth, f)
55     return u
56
57
58 def dfs_traverse(u, get_neighbors, visited, depth, f):
59     neighbors = []
60
61     if (depth > 0):
62         for v in get_neighbors(u):
63             if v not in visited:
64                 visited.add(v)
65                 f(u, v)
66                 neighbors.append(v)
67                 v.neighbors = dfs_traverse(v, get_neighbors, visited,
68                                            depth - 1, f)
69
70     return neighbors
```

Twitter API Controller

```
1 from graph import Node
2 from dotenv import load_dotenv
3 import twitter
4 import arrow
5 import os
6
7 load_dotenv()
8
9 CONSUMER_KEY = os.getenv("CONSUMER_KEY")
10 CONSUMER_SECRET = os.getenv("CONSUMER_SECRET")
11 ACCESS_TOKEN_KEY = os.getenv("ACCESS_TOKEN_KEY")
12 ACCESS_TOKEN_SECRET = os.getenv("ACCESS_TOKEN_SECRET")
13
14 API = twitter.Api(
15     consumer_key=CONSUMER_KEY,
16     consumer_secret=CONSUMER_SECRET,
17     access_token_key=ACCESS_TOKEN_KEY,
18     access_token_secret=ACCESS_TOKEN_SECRET)
19
20
21 def check_rate_limit():
22     url_followers = "https://api.twitter.com/1.1/followers/list.json"
23     url_friends = "https://api.twitter.com/1.1/friends/list.json"
24
25     rr_followers = API.CheckRateLimit(url_followers)
26     rr_friends = API.CheckRateLimit(url_friends)
27     print(rr_followers, rr_friends)
28     if (rr_followers.remaining == 0):
29         print(
30             f"LIMIT on Followers: Try
    {arrow.get(rr_followers.reset).humanize()}"
31         )
```

```
        raise Exception("Rate Limit Exceeded")
    if (rr_friends.remaining == 0):
        print(
            f"LIMIT on Friends: Try {arrow.get(rr_friends.reset).humanize()}")
        raise Exception("Rate Limit Exceeded")


def get_next_mutuals(node: Node):
    followers = API.GetFollowers(node.id)
    friends = API.GetFriends(node.id)
    mutuals = list(set(followers) & set(friends))
    ans = [Node(mutual.id, mutual.name) for mutual in mutuals]
    return ans


def get_current_id_name():
    user = API.VerifyCredentials()
    user_id = user.id
    user_name = user.screen_name
    return user_id, user_name
```

Main Program

```
from graph import Node, bfs, dfs
from twitter_controller import check_rate_limit, get_current_id_name, get_next_mutuals
from grakn.client import GraknClient
from functools import partial


def grakn_write(grakn_session, raw_query, n_args, u, v):
    if (n_args == 1 and (u is None and v is None)):
        return
    if (n_args == 2 and (u is None or v is None)):
        return

    with grakn_session.transaction().write() as write_transaction:
        print(f"Inserting: {u} -> {v}", end="")
        write_transaction.query(raw_query.format(**locals()))
        write_transaction.commit()
        print(f"\tSUCCESS")


def grakn_save(root):
    with GraknClient(uri="localhost:48555") as client:
        with client.session(keyspace="twitter") as session:
            insert_person = partial(
                grakn_write, session,
                'insert $x isa person, has full-name "{v.name}", has uuid "{v.id}";',
                1)
            insert_friendship = partial(
                grakn_write, session,
                'match $p1 isa person, has uuid "{u.id}"; $p2 isa person, has uuid "{v.id}";insert $new_friendship (friend: $p1, friend: $p2) isa friendship;',
                2)

            print("INSERTING PEOPLE")
```

```
            dfs(root, lambda v: v.neighbors, None, insert_person)
            print("INSERTING FRIENDSHIPS")
            dfs(root, lambda v: v.neighbors, None, insert_friendship)


def main():
    check_rate_limit()

    user_id, user_name = get_current_id_name()
    root = Node(user_id, user_name)
    root = dfs(root, get_next_mutuals, 2)
    bfs(root, lambda v: v.neighbors, print, 3)
    check_rate_limit()

    grakn_save(root)


if __name__ == "__main__":
    main()
```