Nth level Search of connections for a User in Social Networking using breadth first search and Neo4j

Akshay Raj Singh Anguwala  
Student CSE Department

University of North Texas

E-mail: rajsinghakshay28@gmail.com

Sri Vaeshnavi Nagula

Student CSE Department

University of North Texas

E-mail: vaeshnavinagula@gmail.com

**ABSTRACT**

In this project we are working on knowing the nth level friends or the connections in a network using a breadth first search and the Ne04j.

**Project Summary**

Through utilizing the breadth first search and the neo4j Graph Database, one may discover the nth level relationships for a user within a social network.

To implement the vast amount of Data in this case, we are employing the Graph Database.

# INTRODUCTION

The Proceedings are to know the nth level connections between two nodes in a Graph such that a user can know a particular nth level connection for a friend in a network.

For example, two users of a social networking site might be represented by nodes, and the relationship between them could be indicated by an edge between them.

Undirected edges can show that two people are buddies or that they both follow the same person, but directed edges show that just one of them follows the other in the edge's direction.

We could implement the scenario using the nth level search which contemplates the BFS or DFS algorithm. For instance, two users of a social media platform might be symbolized by nodes, and an edge connecting them might indicate their relationship.

Undirected edges can indicate that two individuals are friends or that they both follow that very same person, whereas directed edges indicate that only one individual is following the other in the direction of the edge.

As this traversal keeps track of all the nodes at each level, finding buddies at level n could also be accomplished using BFS considerably more quickly.

## Research Problem and Background

When analyzing and working on the research of the problem there was an roadblock to inhibit the data from a lot of people in a network when there is a huge neural network it would have been very disproportionated to evaluate the data and hence there is a use of some database which could handle such a huge amount of data we have done a thorough background verification to use a some type of database to compile the required output in less time and hence we understood different database and found a rather than using a relational database the using of graph database would be better hence to take a fast-forward approach we have used a neo4j graph database.

## Understanding the nth level finding

Let here be a graph with edges and vertices connected to one another.

Here we could see that the graph has connections to each other at different levels hence here we would want to know the 2nd level friends for the Node 1.

Chart

Description automatically generated

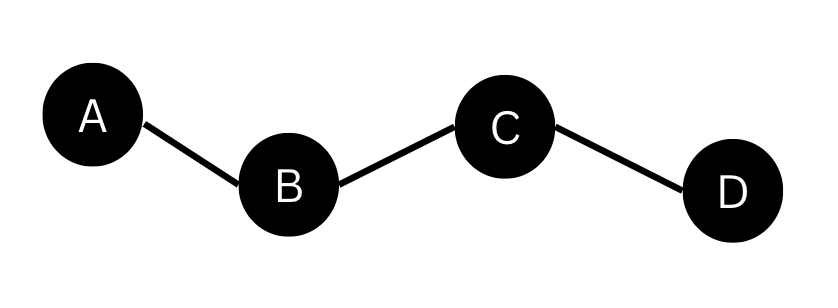
Then the second level friends for the node 1 could be the 1st level friends of the direct connected friends of Node 1.

A picture containing chart

Description automatically generated

Here as we could observe the second level friends would be 4,6 for node 1.

An Example instance Let us understand the whole level to level scenario for the above-mentioned nodes:



|  |  |  |  |
| --- | --- | --- | --- |
| Nodes | Level 1 | Level 2 | Level 3 |
| A | B | C | D |
| B | C, A | D | - |
| C | D, B | A | - |
| D | C | B | A |

# Related Work and Implementation

We have look through the various articles and IEEE papers to work upon this problem and we have resolute the problem in sub parts such that the implementation could become much simpler.

There is a related work by Aleska and Jonas who worked on a query to find out the nth level finding using a query which would make the working much faster in comparison to the code which could not take much of the data through itself.

Given below is the analysis for their work on the speed of the data execution using a simple Cypher query.

Application

Description automatically generated with medium confidence

[imagesource:https://neo4j.com/news/how-much-faster-is-a-graph-database-really/]

When comes to our Implementation we have at first worked on a brute force algorithm to find the nth level node and then we have considered to take the input node at first and then the connection between the nodes. After there is a connection, we ask the user to input what relationship of the you want to know for the nth level finding from that node.

For a code which inhibits the max of 2^(31)-1 could not with hold the huge amount of data with respect to compile it would take a lot of time Hence we connect the neo4j database with the python driver such that we could simply integrate them such that there would be less time and accurate results.

We Have also used the Google cloud to compute our code.

The Source code we have written in python is as follows:

The Code was compiled under the linux virtual machine in

The google cloud environment and it was prescribed to work as expected.

Text

Description automatically generated

Text

Description automatically generated

.py Code used to find the nth level friends’ network.

Here we inducted the neo4j with the python driver to implement the nth level finding of the nodes efficiently.

## Neo4j with Python

Here we have installed the ne04j into. The system and in the ne04j we could either import the database or we could also create one database.

Here we have created the database for a friend system using the queries:

Graphical user interface, text, application

Description automatically generated

//Create the user in the Database with the keys user id and name the examples are mentioned for the same.

create (UserName1:User {id : 1,name:'UserName1'})

create (UserName2:User {id : 2,name:' UserName2'})

//Make the Entitled relationship between the users to make a =n edge between the node the examples are mentioned for the same.

create (UserName1)-[:knows]->( UserName2),( UserName1)-[:knows]->( UserName3),

(UserName1)-[:knows]->( UserName2),

(UserName3)-[:knows]->( UserName4),

Some CQL queries used in Graph Database Neo4j:

CREATE CONSTRAINT ON (n:User) ASSERT (n.id) IS UNIQUE

CREATE INDEX FOR (m:User) ON (m.name)

 MATCH (UserName1:User {id: 1}) RETURN UserName1

 MATCH (n:User) RETURN n LIMIT 25

When we need some kth level value of friends for any user, we use match query as follows:

Match  (p-User)- [\*3]->(n:User) where p.name = 'UserName'  return distinct n

Resulted output would be the nth level friends of a user:

Graphical user interface, text, application

Description automatically generated

This is how the network looks when we try to retrieve the created Graph:

A picture containing necklet, screenshot

Description automatically generatedChart

Description automatically generated

After creating the database we will now get into the notebook and connect the created graph dB and compile our nth level search given below is the visual representation of our compilation where we used a for loop and session. Run() with some cypher queries which gave us the output.

Like what we expected:

Table

Description automatically generated with low confidence

[Note: However we could also attach the .py file for execution]

# Challenges and solutions, Evaluation

## Challenges

During the implementation we had a challenge with respect to converting a brute force thinking on how to contribute to the nth level search and we also had a crucial challenge on how to connect the ne04j with the python driver.

Understanding the Neo4j and trying to implement the network of nodes and edges into the Database.

We had a challenge on how to compile a code in the google cloud if we know how to compile how to grow the logic there save it and execute.

The challenge was to create an instance in the cloud and run our files in the Virtual machine by installing the befitting packages for the same.

establishing connections between the nodes, viewing their visual representation, and receiving the output in the desired format.

Using knowledge of and implementation of the Cypher query language,

Build the data.

To obtain the finished product for the social network dataset that was created, create relationships with that data (relations between nodes).

## Solutions

To convert out thinking of algorithm into the python code we have made a visit to various websites and implemented the code.

We have searched a blog and neo4j documentation to interlink them both and have successfully compiled it.

We were able to create a Linux virtual machine in the google cloud and were able to execute the same there.

Worked on knowing different approaches to be used to implement the problem statement for the project.

Reading different Blogs and relevant articles and visiting the websites with similar content on BFS and Graph Databases to get an idea on how the problems can be addressed.

Visited the Neo4j official website to learn the CQL queries and was able to implement them.

## Evaluations

The Evaluations were done based on the required outputs and were noted down into making the comparisons by what is to be expected and what did we get.

The Evaluation metrics were simple which gave us the exact output which covered:

Chart, bubble chart

Description automatically generated

# Comparison to related work

We have collected, analyzed, and coded from the IEEE and Nuclei website which helped us to work on the approach we were taking towards this project. We would majorly discuss on the related research efforts and why we focus on the nth level search in the network.

## Social Data Analysis

Novetta's SocialBee can make use of this largely untapped data source to perform more in-depth social network analysis based on actor behavior as well as to enrich the social network analysis with topic modelling, sentiment analysis, and trending over time by combining this metadata with data and communications content [3].

SocialBee has also proven successful in predicting hidden relationships that exist in an external dataset via various channels of communication through the extraction and analysis of topic-enriched links. However, SocialBee also examines the content of communications, enabling a richer analysis of the tone, topic, and sentiment of each interaction. This enables the clustering of communities based on behavior over time.

Our problem is that a traditional social network analysis frequently only examines the connections between actors rather than the content of their conversations. This means that even though we may be aware of a relationship between two people, we are unaware of its true nature.[3]

-PetraIsenberg

# Results

We have got some good results while we have given the required input either by a code or through a query,

At first let us see the result with the python input what we gave was there 6 nodes and gave the relationship between them and we got the required output as we can see from below Text

Description automatically generated

Text

Description automatically generated

And, when we do it with the huge data where the neo4j is connect with python it works as well please see the same below

Graphical user interface

Description automatically generated with low confidence

# Use of Google Cloud

We have created a Linux virtual machine in the cloud named as instance 1 which is and Debian linux which has 10gb of disk space

After creation of the VM we have checked the directory with pwd and then we have installed the miniconda to run the python files into the created nano file editor.

After creating a .py file we have executed the python file into the linux machine using the command ‘python example.py’

And gave the graph nodes and their relation as input and we have got the respected output.

Please see below the use of google cloud:

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

# Milestones Achieved and Learnt from project

* We were able to inter-link the Python/C++ programming language interface with Graph DB.
* We have completed and implemented the Algorithm of BFS to achieve the feat of searching nth level users in the network.
* We also created the Dataset Having the username and id with their specific links.
* We widely used the google cloud into the Problem statement.
* We implement the code in the Linux Virtual Debian machine, commit it.
* Documented the source code and README file.

We have learnt on how to implement the graph traversal have good understanding of the graph database and how to use the google cloud platform.

# Project Status

Chart, bar chart

Description automatically generated

Made 100% progress for the Implementation and Completion quarter which got the project to the conclusion.

# REFERENCES

[1] Yuede Ji., and H. Howie Huang. 2020.AQUILA: Adaptive Parallel Computation of Graph Connectivity Queries. HPDC ’20, June 23–26, 2020, Stockholm, Sweden.149-160

[2] Bharati, A. (2019, October 3). *Social Networking with BFSand Neo4j*. Nuclei. https://gonuclei.com/resources/social-networking-with-bfs-and-neo4j/

[3] pertha (no date) *Social Analysis*, *tutorial 6*. Available at: https://www.aviz.fr/TeachingVA2016/Tutorial6.

*[4] Understanding Neo4j’s data on disk - Knowledge Base*. (n.d.). Neo4j Graph Data Platform. <https://neo4j.com/developer/kb/understanding-data-on-disk/>

[5] L. Luo, M. Wong, and W.-m. Hwu, “An effective gpu implementation of breadth-first search,” in Proceedings of the 47th Design Automation Conference, ser. DAC ’10. New York, NY, USA: ACM, 2010, pp. 52–55. [Online]. Available: <http://doi.acm.org/10.1145/1837274.1837289>

[6] L. Munguia, D. Bader, and E. Ayguade, “Task-based parallel breadth-first search in heterogeneous environments,” in High Performance Computing (HiPC), 2012 19th International Conference on, Dec 2012, pp. 1–10.

[7] S. Hong, T. Oguntebi, and K. Olukotun, “Efficient parallel graph exploration on multi-core cpu and gpu,” in Parallel Architectures and Compilation Techniques (PACT), 2011 International Conference on, Oct 2011, pp. 78–88.

[8] S. Beamer, K. Asanovi´c, and D. Patterson, “Direction-optimizing breadth-first search,” in Proceedings of theInternational Conference on High Performance Computing, Networking, Storage and Analysis, ser. SC ’12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 12:1–12:10. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2388996.2389013>

[9] D. Merrill, M. Garland, and A. Grimshaw, “Scalable gpu graph traversal,” in Proceedings of the 17th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ser. PPoPP ’12. New York, NY, USA: ACM, 2012, pp. 117–128. [Online]. Available: http://doi.acm.org/10.1145/2145816.2145832

[10] J. Leskovec and A. Krevl, “SNAP Datasets: Stanford large network dataset collection,” http://snap. stanford.edu/data, Jun. 2014.

[11] Graph500, “http://www.graph500.org/.”

[12] J. Travers and S. Milgram, “An experimental study of the small world problem,” Sociometry, pp. 425–443, 1969.