Graph Traversal Using Apache TinkerPop: Gremlin

Anirban Dey
Debajyoti Maity

July 3, 2023

1 Introduction

Graph databases have gained significant popularity in recent years due to their ability to store and analyze highly connected data. They provide a powerful and flexible way to represent and query complex relationships between data entities. Gremlin is a query language and traversal framework specifically designed for graph databases.

In this project documentation, we explore the use of Gremlin in the context of graph databases. We will discuss the installation process, the Gremlin query language, the traversal API, and provide some examples of Gremlin traversal on real-life data.

1.1 Graph Databases

Graph databases are a type of NoSQL database that uses graph structures to represent and store data. They consist of nodes (vertices) that represent entities and edges that represent relationships between those entities. Graph databases excel in handling highly connected data and are suitable for various use cases, including social networks, recommendation engines, fraud detection, and knowledge graphs.

1.2 Gremlin and its Use

Gremlin is a graph traversal language that allows users to perform complex queries and traversals on graph databases. It provides a concise and expressive syntax to navigate, filter, and manipulate the data stored in the graph. Gremlin is supported by several graph databases, including Apache TinkerPop, Amazon Neptune, JanusGraph, and Cosmos DB.

Gremlin is widely used in various domains and software applications. Some notable examples include:

- **Social Networks**: Gremlin is used by social media platforms to analyze and recommend connections between users based on their interests, relationships, and activities.

- **Recommendation Engines**: E-commerce platforms leverage Gremlin to generate personalized product recommendations by analyzing the purchase history, browsing patterns, and preferences of users.
- **Fraud Detection**: Gremlin is employed in fraud detection systems to identify suspicious patterns and connections in financial transactions or online activities.
- **Knowledge Graphs**: Gremlin is utilized to query and navigate large knowledge graphs that store structured information about various domains such as science, medicine, and literature.

This documentation aims to provide a comprehensive guide to using Gremlin in your project and harnessing the power of graph databases. Here, we will use Apache TinkerPop for our purpose.

2 Background

Graph databases have emerged as a powerful solution for handling highly connected data and complex relationships between entities. Unlike traditional relational databases, which store data in tables, graph databases represent data as nodes (vertices) and relationships as edges. This data model allows for efficient traversal and querying of the graph structure, enabling powerful insights and analysis.

One of the key challenges in working with graph databases is effectively navigating and querying the graph structure. This is where Apache TinkerPop and its query language, Gremlin, come into play. Apache TinkerPop is an open-source graph computing framework that provides a standardized way to work with graph databases. Gremlin, the query language used in TinkerPop, offers a powerful and expressive syntax for traversing and manipulating graphs.

The development of TinkerPop and Gremlin was motivated by the need for a unified and vendor-agnostic approach to graph computing. Prior to TinkerPop, there were several graph databases, each with its own query language and APIs. This fragmentation made it difficult for developers and users to switch between different graph databases or combine multiple graph databases in a single application.

TinkerPop addresses this challenge by providing a common interface and abstraction layer for interacting with graph databases. It defines a set of APIs and interfaces that enable developers to write graph-based applications without being tightly coupled to any specific database implementation. This means that you can write your application using the TinkerPop APIs and query the graph using Gremlin, and then easily switch between different graph databases that are compatible with TinkerPop.

Apache TinkerPop has gained significant adoption in the graph database ecosystem and is supported by several popular graph databases, including JanusGraph, Amazon Neptune,

and Microsoft Azure Cosmos DB. TinkerPop's vendor-agnostic approach and Gremlin's expressive query language make it a versatile choice for developing graph-based applications.

The goal of this project documentation is to provide a comprehensive guide to using Apache TinkerPop and Gremlin for graph traversal. Additionally, we will showcase some real-life examples of graph traversal using Gremlin on Apache TinkerPop.

References:
1. Apache TinkerPop. (n.d.). Retrieved from
`https://tinkerpop.apache.org/`
2. Gremlin: The Graph Traversal Language. (n.d.). Retrieved from
`https://tinkerpop.apache.org/gremlin.html`

3 Installation

This section provides step-by-step instructions on how to install Gremlin Console on an Ubuntu system. Gremlin Console is a command-line tool that allows you to interactively execute Gremlin queries and commands.

3.1 Prerequisites

Before installing Gremlin Console, ensure that you have the following prerequisites:

- Ubuntu system with administrative privileges.
- Java Development Kit (JDK) installed. Gremlin Console requires Java to run. You can install OpenJDK using the package manager:

```
sudo apt update
sudo apt install openjdk-8-jdk
```

3.2 Installing Gremlin Console

To install Gremlin Console on your Ubuntu system, follow these steps:

1. Open a terminal window.
2. Download the latest version of Apache TinkerPop from the official website using the `wget` command:

```
wget https://downloads.apache.org/tinkerpop/<version>/apache-tinkerpop-<version>-bin
```

Replace `<version>` with the desired version of Apache TinkerPop. For example, if you want to install version 3.5.0, the command will be:

```
wget https://downloads.apache.org/tinkerpop/3.5.0/apache-tinkerpop-3.5.0-bin.zip
```

3. Unzip the downloaded file using the `unzip` command:

   ```
   unzip apache-tinkerpop-<version>-bin.zip
   ```

   This will create a directory named `apache-tinkerpop-<version>` in the current directory.

4. Move the extracted directory to a desired location. For example, you can move it to the `/opt` directory:

   ```
   sudo mv apache-tinkerpop-<version> /opt/tinkerpop
   ```

5. Add the `bin` directory of the TinkerPop installation to the system's `PATH` environment variable by editing the `.bashrc` file:

   ```
   nano ~/.bashrc
   ```

   Add the following line at the end of the file:

   ```
   export PATH="/opt/tinkerpop/bin:$PATH"
   ```

   Save the file and exit the text editor.

6. Load the updated `.bashrc` file to apply the changes:

   ```
   source ~/.bashrc
   ```

7. Verify that Gremlin Console is installed correctly by running the following command:

   ```
   cd /opt/tinkerpop/bin/
   ./gremlin.sh
   ```

8. If this does not work please do the following:

```
chmod +x gremlin.sh
./gremlin.sh
```

   You should see the Gremlin Console prompt, indicating that the installation was successful.

Congratulations! You have successfully installed Gremlin Console on your Ubuntu system.

Note: The above instructions assume that you are installing Apache TinkerPop version 3.5.0. Make sure to adjust the version numbers accordingly based on your requirements and the latest available version of TinkerPop.

4 Getting Started With Some Basic Queries

To get started with Gremlin in your project, follow these steps:

1. Install and set up Gremlin Console as described in the previous section.
2. Launch Gremlin Console by running the following command in the terminal:

   ```
   ./gremlin.sh
   ```

   You will see the Gremlin Console prompt.
3. Start by connecting to a graph database. Gremlin Console provides a preconfigured `graph` object that represents the current graph database connection. You can connect to a graph database by executing the appropriate command based on the database you are using. For example, to connect to a TinkerGraph, which is an in-memory graph database provided by TinkerPop, use the following command:

   ```
   graph = TinkerFactory.createModern().traversal()
   ```

   This command connects to the TinkerGraph and assigns the traversal instance to the `graph` variable.
4. Now you can start executing Gremlin queries on the connected graph database. Gremlin queries are written in a concise and expressive syntax that allows you to navigate, filter, and manipulate the data stored in the graph. For example, to retrieve all the vertices in the graph, you can use the following query:

```
graph.V()
```

This query returns all the vertices in the graph.

This is just a basic example to get you started. Gremlin provides a rich set of traversal steps and methods that enable you to perform complex queries and manipulations on the graph. The Gremlin documentation and resources available on the Apache TinkerPop website provide comprehensive information and examples to help you explore and utilize the full power of Gremlin in your project.
For reference you can see :
`https://tinkerpop.apache.org/docs/current/reference/`

5 Gremlin Query Language

The Gremlin query language provides a powerful and expressive syntax for traversing and manipulating graphs. Gremlin queries consist of a sequence of steps that are applied to a graph traversal. Each step performs a specific operation on the traversal, such as navigating to adjacent vertices, filtering vertices based on properties, or aggregating results.

The syntax of a Gremlin query is as follows:

```
<graph traversal>.<step1>().<step2>().<step3>()...
```

Here, `<graph traversal>` represents the starting point of the traversal, and `<step1>, <step2>, <step3>, ...` represent the sequence of steps to be applied to the traversal. Each step returns a new traversal that can be further modified or evaluated.

Gremlin provides a large number of built-in steps that cover various traversal operations, including navigating the graph structure, filtering vertices and edges, transforming and aggregating data, and executing complex computations. Some commonly used steps include:

- `V()`: Select all vertices in the graph.
- `E()`: Select all edges in the graph.
- `has(label, value)`: Filter vertices or edges based on a property value.
- `out()`: Traverse outgoing edges from the current vertex.
- `in()`: Traverse incoming edges to the current vertex.
- `both()`: Traverse both outgoing and incoming edges from the current vertex.
- `limit(n)`: Limit the number of results to `n`.

- `count()`: Count the number of vertices or edges in the traversal.
- `valueMap()`: Retrieve the properties of vertices or edges.

This is just a small subset of the available steps. Gremlin provides many more steps that allow you to perform complex traversals and computations on the graph.

To learn more about the Gremlin query language and the available steps, refer to the official Apache TinkerPop documentation at `https://tinkerpop.apache.org/docs/current/reference/#gremlin-queries`.

6 Gremlin Traversal On Jan-2019-ontime.csv Data

1. Creating an empty graph object

```
graph = TinkerGraph.open()
```

2. Importing the data in the graph from a graphml file

```
graph.io(graphml()).readGraph('/home/system/your-graphml-file.graphml')
```

3. create a graph traversal source object for our loaded graph

```
g = graph.traversal()
```

4. Show the properties of first few vertices and edges

```
g.V().limit(10).valueMap()
g.E().limit(10).valueMap()
```

5. Listing all vertices 'b' (DEST) which are connected by 'a'(ORIGIN)

```
g.V().outE()
```

6. count of vertices that can be reached by following two consecutive outgoing edges from the starting vertices

```
g.V().out().out().count()
```

7. Finding the count of unique paths in the graph that satisfy:
   Vertex 'a' to 'b' to 'c', where 'a' is connected to 'b' and 'b' is connected to 'c', then filtering out paths where 'a', 'b', and 'c' are distinct and then again filtering out paths where there exists an incoming edge to 'c' from any vertex other than 'a':

```
g.V().as('a')
.out().as('b')
.out().as('c')
.where('a', P.neq('b'))
.where('a', P.neq('c'))
.where('b', P.neq('c'))
.where(.as('a').out().not(.in('c')))
.select('a', 'b', 'c')
.count()
```

8. python code csv2graphml

```
  import csv
import networkx as nx

# Read the CSV file
with open("D:\\CHROME DOWNLOAD\\Jan_2019_ontime.csv", 'r') as f:
    reader = csv.DictReader(f)
    data = [row for row in reader]

# Create a directed graph
G = nx.DiGraph()

# Add edges and vertices to the graph
for row in data:
    src = row['ORIGIN']
    dst = row['DEST']
    distance = float(row['DISTANCE'])
    src_name2 =row['ORIGIN_AIRPORT_ID']
    dst_name_2=row['DEST_AIRPORT_ID']
```

```
    # Add vertices with name and ID properties
    G.add_node(src, ID=src, ORI_AIR_ID=src_name2)
    G.add_node(dst, ID=dst, DST_AIR_ID=dst_name_2)

    # Check if an edge already exists between src and dst
    if G.has_edge(src, dst):
        # Keep only the edge with the largest distance
        if G[src][dst]['distance'] < distance:
            G[src][dst]['distance'] = distance
    else:
        G.add_edge(src, dst, distance=distance)

# Write the graph to a GraphML file
nx.write_graphml(G, "D:\\CHROME DOWNLOAD\\Final_graph3.graphml")
```