[Project 1] [Phase 1]

CSE 511: Data Processing at Scale - Spring 2025

Available: 03/05/2025 **Due Date**: 03/30/2025 11:59pm **Points: 100**

Introduction

Graph-based data is ubiquitous and has various applications, such as examining the structure of the internet, bioinformatics data representation, chemistry, and distributed computation. Academia and industry collaborations have developed their own graph processing systems to handle massive graphs. Graph-processing tasks involve using specific algorithms to extract various properties from graphs, such as computing important vertices with PageRank or identifying large communities using different algorithms. These algorithms are typically executed in parallel when resources are available. When implementing such algorithms, developers must consider challenges such as parallelism and the use of underlying frameworks. In this project, we will explore how to implement graph processing algorithms in both non-distributed and distributed deployment environments.

Project Logistics

This is an **individual** project. The project is divided into two phases, Phase 1 and Phase 2. Each phase carries equal weightage i.e. **10.0 %** of the total overall grades.

Problem Statement

In Phase 1 of this project, you will be learning to create Docker containers that install neo4j and will be implementing **two** Graph algorithms in neo4j. For this project, we will use the **NYC Yellow Cab Trip dataset.** To make it easy to follow, we have divided this assignment into 4 broad steps that will help you understand the topics and complete the project.

Step 0: Exploring the world of Graph Databases (0 pts)

Graph databases use specialized data structures and algorithms to efficiently traverse and query these graph structures. Graph databases are well-suited for highly interconnected data, such as social networks, recommendation engines, fraud detection, and knowledge management

systems. For those new to this domain, we highly advise you to follow the steps below to get a general idea of how to use graph databases and what they are capable of:

- Create a free neo4j account to start exploring! They provide a variety of tools for developers and in this part of the project, we are going to explore one of those ones: Neo4j Aura.
- Create a new AuraDB instance (only the first instance is free). Go ahead and create that instance and load the provided stack overflow database. **Do remember to save or remember the password!**
- It will take a couple of minutes to create the instance, after which you can login to the instance using the password from earlier. In this instance, you will see a handy toolbar on the right which will prompt you to go through various operations that can be performed in the stack overflow dataset! These steps are very well written and elaborated, simply going through them slowly will help gain a very good understanding of the topic.

In the scope of this project, we will use the TLC Trip Record Data. Specifically, the dataset of March 2022. The schema of the NYC Trip dataset is available here. You can also go through this short blog to understand how such datasets are making cities smarter.

Step 1: Setting up the Dockerfile and Loading the data (35 pts)

This step of the project will explore loading data into neo4j and what a Dockerfile looks like. By the end of this step, you should be able to explore your neo4j environment with the loaded data and have a basic grasp of creating the Docker containers we have been using for the last few assignments.

To get you started off, we have already provided you with a template `Dockerfile` which performs various preliminary tasks like setting up the operating system and the target build platforms (so that your container can run on both ARM and x86). Additionally, we have also provided you with a template `data_loader.py` file here.

In the `data_loader.py` file, you need to write a query to load the data into the file. The schema you should follow has been described below. Ensure that you follow this!

- ***1 node for each pickup/dropoff locationID**. The node should have the label "**Location**". The node should have the properly "**name**" as the locationID (formatted as integer). The location ID corresponds to the columns PULocationID and DOLocationID*

- ***1 relationship for each trip**. The relationship should have the type "**TRIP**". This relationship should have the following properties: the **distance** (formatted as float), **fare** (formatted as float), **pickup_dt** (formatted as datetime), and **dropoff_dt** (formatted as datetime) as its properties.*

***Note**: Make sure you name the relationships, nodes, and properties named properly as it may cause tests to fail! For the datetime, you may choose if you want to include the timezone or not. The grading script is agnostic to it.*

Hints:

These are some high-level steps to implement Step 1. However, in consideration of the breadth of the topic, we highly encourage you to explore these topics by yourself as well

1. We first need to download 2 files here, the dataset, and the `data_loader.py` file from your personal GitHub repository (uploaded by yourself after implementing the query) into the Docker Container.
    a. Find out how you would download a file in bash. Then write a command that downloads the taxicab dataset in the correct location.
    b. Explore how you can perform a **git clone** inside a Docker container for private organization repositories. **DO NOT give us your GitHub password**. Instead, you might want to check what **GitHub Tokens** are.
2. Now that you have downloaded the files, you need to make sure that the necessary Python libraries are installed. Python uses `pip` (pip stands for => pip installs packages).
    a. You should not need to install any packages except `neo4j`, `pandas`, `pyarrow`. Make sure you **upgrade** `pip` before trying to install the `neo4j` package.
3. Setup the `neo4j` password (the password should be **project1phase1**) and also setup how the server should broadcast
    a. You can refer this document to get started with how to configure `neo4j` for machines that are not local (for remote access). Docker in some ways behaves like a remote machine for this particular scenario, you can consider that and try those changes.
    b. Explore how you can edit files in Linux without actually opening them. That is an important tool for this bit.
4. The `data_loader.py` file that you have cloned from GitHub can now be run. This part is handed in the RUN block that has already been provided by us, you can refer to that to see how you are to write your code.

a. For the `data_loader.py` file, most information has been provided in the file itself along with some data cleaning that has been done for you. You will easily be able to interface with `neo4j` and your responsibility lies with loading the data according to the schema provided.

If you have followed all the steps correctly, **you will be able to create a docker image using your dockerfile**, and on running a container using this instance, your `neo4j` instance will be live.

Step 2: Setting up the required neo4j plugin (15 pts)

You are permitted to install and use any neo4j plugin you want to use for this phase of the project. In this step, you will perform the installation of the neo4j GDS plugin from the `Dockerfile`. The GDS (graph data science) plugin adds several basic and useful procedure calls to neo4j, and more specifically, it adds procedure calls for Step 3.

Hints:
Exploring a bit will lead you to the official installation documentation and steps which will guide you on how to perform this

1. You need to download the GDS plugin (version 2.3.1) and move the jar file to the plugin directory of neo4j. In our case, the plugin directory is `/var/lib/neo4j/plugins/`
2. Make sure the security procedures in neo4j.conf are set appropriately!

At the end of this step, you should have your entire environment set up and be ready to perform operations. We have exposed port 7474 from inside docker, this allows you to even browse `localhost:7474` on your host machine and interact with your neo4j browser directly! Some queries you might want to try running over there:

- Viewing the DB schema: `CALL db.schema.visualization();`
- View some of the data: `MATCH (n) RETURN n LIMIT 25;`

***Note**: Be patient after creating your container. **It takes 2-4 minutes** for the server to be available after starting.*

Step 3: Implementing Graph Algorithms (50 pts)

Now, when you have your Neo4j development setup ready on a container instance, the next step is to implement the **two** Graph Algorithms mentioned below. These algorithms need to be implemented in the **interface.py file** in their respective functions. The interface.py file does not

need to be inside the container and will connect to the docker container from outside it, i.e. using the 7687 port that we have exposed from inside docker.

1. **PageRank (25 pts)**

   The PageRank algorithm evaluates the significance of each node in a graph by considering the number of incoming relationships and the importance of the source nodes that contribute to it. The basic premise of this algorithm is that a page's importance is determined by the pages that link to it. Pages rank in original Google paper is defined as below

   $$PR(A) = (1-d) + d(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)})$$

   Where

   - we assume that page A has pages T_1 to T_n which point to it.
   - d is a damping factor which can be set between 0 (inclusive) and 1 (exclusive). It is usually set to 0.85.
   - C(A) is defined as the number of links going out of page A.

   This equation is used to iteratively update a candidate solution and arrive at an approximate solution to the same equation.

   What to implement?

   You need to implement the PageRank algorithm which will tell you how the locations (nodes) you have created in the graph fare against each other. Make sure that you take the **max_iter** and **weight_property** variables into consideration. A helpful guide here will be the official GDS documentation for PageRank. Once done, you need to return the nodes with the maximum and minimum PageRank.

2. **Breadth First Search (25 pts)**

   The Breadth First Search algorithm is a graph traversal algorithm that **given a start node visits nodes in order of increasing distance**. When the probability of finding a specific node decreases as the distance increases, this algorithm is useful for searching. It supports various termination conditions for the traversal, such as reaching one of several target nodes, reaching the maximum depth, exhausting a given budget of traversed relationship cost, or traversing the entire graph.

What to implement?

You need to implement the BFS algorithm from location A to multiple locations. The locations will be provided using 2 variables start_node and target_nodes. Make sure to use these variables.

***Note**: The neo4j GDS library has support for these algorithms, you are permitted to install the library and use the algorithms directly.*

Grading:

- We expect you to provide us with a `Docker` file. This docker file will be used to create an image and deploy your container to test your code.
- We have provided a **Docker file template**. Kindly follow the same to add your code. **DO NOT EDIT** any existing code in the template docker file.
- The Docker file **MUST** be able to perform the following steps **without any manual interference**. (Failure to do so will result in 0-grade points)
    - Download the dataset (NYC Trip: March 2022) and get your code from GitHub
    - Complete setup of the neo4j environment.
    - Load the data into the database

Submission Requirements & Guidelines

Submit the assignment following the below guidelines

1. This is an **individual** assignment.
2. **What to maintain on the GitHub private repository?**
    1. Maintain your code on the provided private GitHub repository in the https://github.com/SP-2025-CSE511-Data-Processing-at-Scale
3. **What to submit on Canvas?**
   Create a **zip** file containing the following:
    1. `Dockerfile`
    2. `interface.py`
    3. `data_loader.py`
4. **What is the Submission File nomenclature?**
    1. Name the **zip** as per your asuriteId; eg johnDoe.zip
    2. Your docker file should be named "**Dockerfile**"

3. Your Python file should be named "**interface.py**" and "**data_loader.py**"

Submission Policies

1. Late submissions will *absolutely not* be graded (unless you have verifiable proof of emergency). It is much better to submit partial work on time and get partial credit for your work than to submit late for no credit.
2. Every student needs to *work independently* on this exercise. We encourage high-level discussions among students to help each other understand the concepts and principles. However, a code-level discussion is prohibited, and plagiarism will directly lead to failure of this course. We will use anti-plagiarism tools to detect violations of this policy.