Graph Data Processing and Analytics Using MongoDB

NoSQL Final Project

Self Introduction

- Aditya Vardhan
- Archit Sangal
- Gagan Agarwal

Group 6

Problem Statement

- Provide an easy and quick way to load, process, and analyse graphs utilising the aggregation pipeline made available by MongoDB.
- The project can adjust the analysis to the demands of the user by filtering, grouping, and transforming the input graphs through a series of stages.
- The overall objective of the project is to create a reusable set of components that may be used for various graph transformation scenarios, facilitating and accelerating the analysis of big graphs.

Motivation

- Large graph processing and analysis, however, can be difficult and time-consuming.
- The project seeks to address this issue by utilising the capabilities of MongoDB's aggregation pipeline.
- We were given 5 categories and the graph data processing & analytics category seemed very innovative and interesting to me.
- Want to try something new.

Approach

- Making a synthetic dataset
- Making components.
- Pipeline can also be treated as components and it gets that same reusability.
- Analytics is performed using JSNetworkx

Dataset

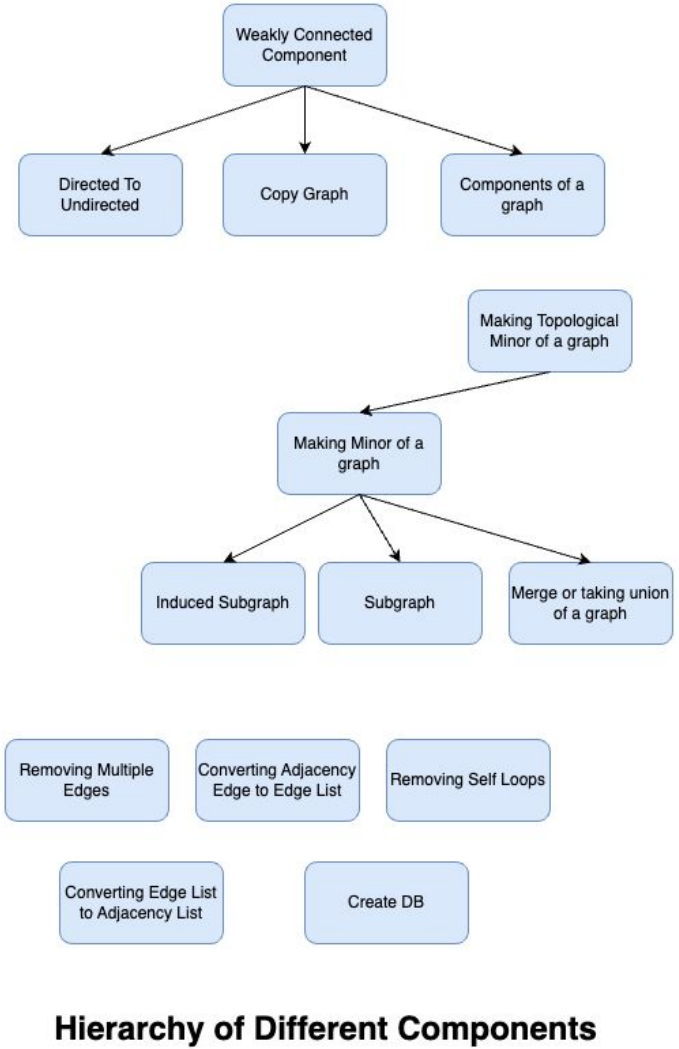
- We required directed and undirected graphs of varied node sizes from small to large.
- We created a script to generate random directed and undirected graph with a specified number of nodes.
- We used JsNetworkx library to generate these graphs.
- An edge between every pair of nodes is created with a specified probability.
- Once the graph is generated we convert it to edgeList format, and export it to a json file.
- Finally we use mongoimport to import the graph to a collection in our database.

Implementation and System Demo

- Jsnetworkx
- Jsnx

Implementation and System Demo

- After importing graph, we can perform the task using scripts.



Hierarchy of Different Components

Implementation and System Demo

- Jsnetworkx

- http-server

Evaluation

	Time taken for 10 vertices	Time taken for 100 vertices	Time taken for 1000 vertices
Importing Graph	0.1	1.9	11.0
Copying Graph	1.7	1.75	3.2
Conversion to Adj. List	1.7	1.77	3.58
Conversion to Edge List	1.71	1.773	4.42
Induced Subgraph	1.83	2.284	1800+
Subgraph	2.154	2.644	1800+
Topological Minor	1.889	2.073	1800+
Minor	2.02	2.517	1800+
Remove Self Loop	1.789	2.073	4.32
Remove Mutli-edges	1.717	2.2	4.57
Components Of graph	1.815	2.605	4.57
WCC Of graph	10.635	1206.4	1800+

Analytics

- Once our components pipeline generates an output graph, we export it to a json file using mongoexport.
- We then read this json file in our analysis script file, that creates a graph using the JSNetworkX library.
- We then execute functions such as number of nodes, edges, maximum clique size, all pairs shortest path algorithm.
- Apart from displaying this information in the terminal, we also use d3.js to visualize the output graph in the browser.

Conclusion

- MongoDB is indeed capable of handling graphs.
- We were able to make reusable scripts which were very user-friendly interface.
- Evaluation Results were also quite good.

Lessons Learned

- We did found out that join operations are very expensive. We had aggregation pipeline for finding connected components that was taking a lot of time. We changed the way of doing the aggregation, and we were able to reduce the time taken significantly.
- $graphLookUp is not that intuitive
- Wanted to try something out of the box. Graphs with MongoDB (Hands On)

Thank You