CS39202 : Database Management Systems Laboratory

GRAPHITY
Graph Processing Database System

Group Members

Shashwat Kumar	*21CS30047*
Sukhomay Patra	*21CS30066*
Swapnil Ghosh	*21CS10086*
Soumojit Chatterjee	*21CS30062*
Raj Parikh	*21CS30039*

Contents

1. Introduction

Graphity is a Graph Processing Database System. This provides a web-based interface to query, manipulate and retrieve different salient aspects of a graph and perform operations on it. This makes working with graphs easier in a database management context, as traditional query languages like SQL are not particularly well-suited for working with graphs. In our project, we leverage a combination of graph algorithms to emulate a file system within a graph database. The idea is to store the graph in several text files of fixed size and then access them efficiently using required indices using efficient data structures.

GitHub respository to access the source code:
https://github.com/Sukhomay/G-R-A-P-H-I-T-Y-Graph-Processing-Database-System

1.1 Use cases

Graphity has many prospective use cases, as it works with a variety of graphs. Some of the use cases are as follows:

- ***Social Networks Analysis***: Graph databases excel in modeling social networks where nodes represent users, and edges represent relationships.
- ***Fraud Detection:*** Graph databases can detect fraudulent activities by analyzing patterns of connections between entities such as accounts, transactions, and IP addresses.
- ***Knowledge Graphs:*** Organizations utilize graph databases to build knowledge graphs that represent structured information and relationships between concepts, entities, and attributes.
- ***Biological and Medical Research:*** Graph databases aid in biological and medical research by modeling complex biological networks such as protein-protein interactions, genetic pathways, and disease associations.
- ***Geospatial Analysis:*** Graph databases are used to represent and analyze geospatial data such as transportation networks, supply chains, and location-based services.
- ***Semantic Web:*** Graph databases play a crucial role in the Semantic Web by representing linked data and ontologies.

1.2 Functions [Objectives]

Some basic operations supported by Graphity are as follows:

- Metadata of the graph
    - Count of Nodes, Edges
    - Count of Weakly Connected Components (WCC)
    - Count of Strongly Connected Components (SCC)
    - Size of Largest Strongly Connected Component (SCC)
    - Size of Largest Weakly Connected Component (WCC)
    - Count of simple cycles
    - Time required and Blocks (files) used for pre-processing
- Indegree of queried node u
- Outdegree of queried node u
- PageRank value and the rank of a queried node u
- Shortest Distance between two queried nodes u and v
- K-nearest neighbours of queried node u
- Rank List of nodes for the queried range [l, r]
- Whether queried nodes u and v belong to the same WCC, SCC

1.3 Salient Features

- Emulation of block access from disk is done through binary files of fixed size (512B) to properly monitor number of blocks accessed to gauge the performace of the database.
- Primary clustering ordered index is implemented for quick access of variable-sized adjacency lists are variable sized.
- Global information regarding graph is stored in a single file Metadata.txt.
- Node information is stored in fixed size chunks of 64 B which significantly improves performance on node-specific queries.
- Implemented static hashing for storage of pagerank-List to quickly handle range queries for a queried range [l, r]
- Customised adjacency list storage to optimise memory and support both weighted and unweighted graphs.
- Web-based interface for graph manipulation provides accessibility from any device, intuitive usability, and seamless integration with other web services, enhancing collaboration and scalability while reducing deployment complexity.

2. Methodology

2.1 Data structures for efficient storage and indexing

```
class Node:

def __init__(self, node_id, adj_start_idx, in_deg, out_deg, scc_id, pg_rank,
wcc_id, rank, adj_start_idx_in):

        self.node_id = node_id
        self.adj_start_idx = adj_start_idx
        self.in_deg = in_deg
        self.out_deg = out_deg
        self.scc_id = scc_id
        self.pg_rank = pg_rank
        self.wcc_id = wcc_id
        self.rank = rank
        self.adj_start_idx_in = adj_start_idx_in
```

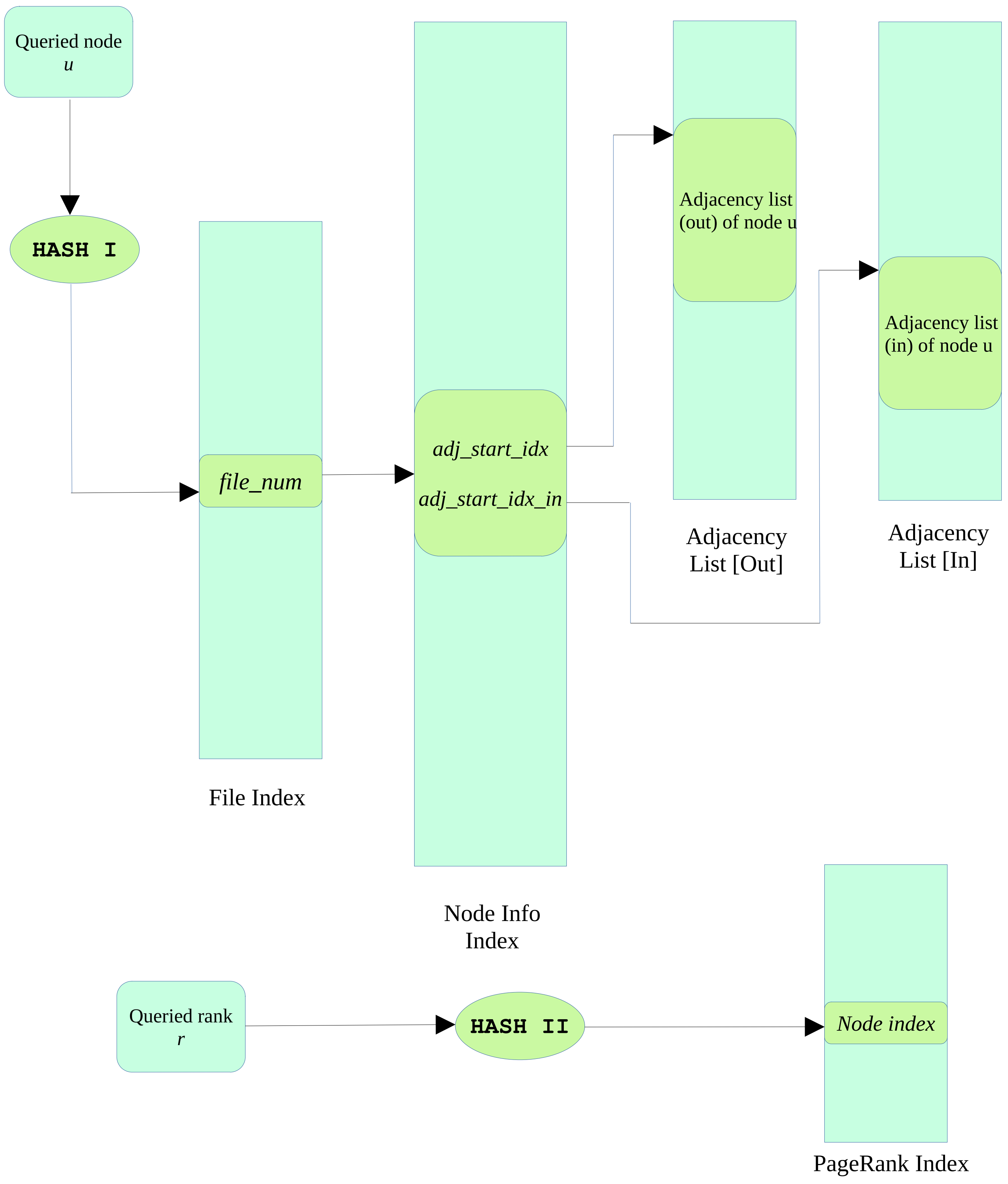
```
class Index:

def __init__(self, file_num):

        self.file_num = file_num
```

Note: *The index structure is 4 bytes, and the record structure is 64 bytes, both of which are factors of the block size (512B). It is created that way so that the disk blocks are used efficiently, without the need to split a record or an index into two blocks.*

2.2 File structure mechanism for efficient block access



2.3 Adjacency List storage

The adjacency lists of the nodes are of variable length, and so they are allowed to cross the block boundaries, because otherwise the blocks would have been used very inefficiently, with as much as 50% space being wasted in the worst case. In the implementation of ***Graphity***, the adjacency lists are stored as follows:

⟨adjacency list of node 0⟩ ⟨adjacency list of node 1⟩ ⟨adjacency list of node 2⟩...

Structure for adjcency list of node i:

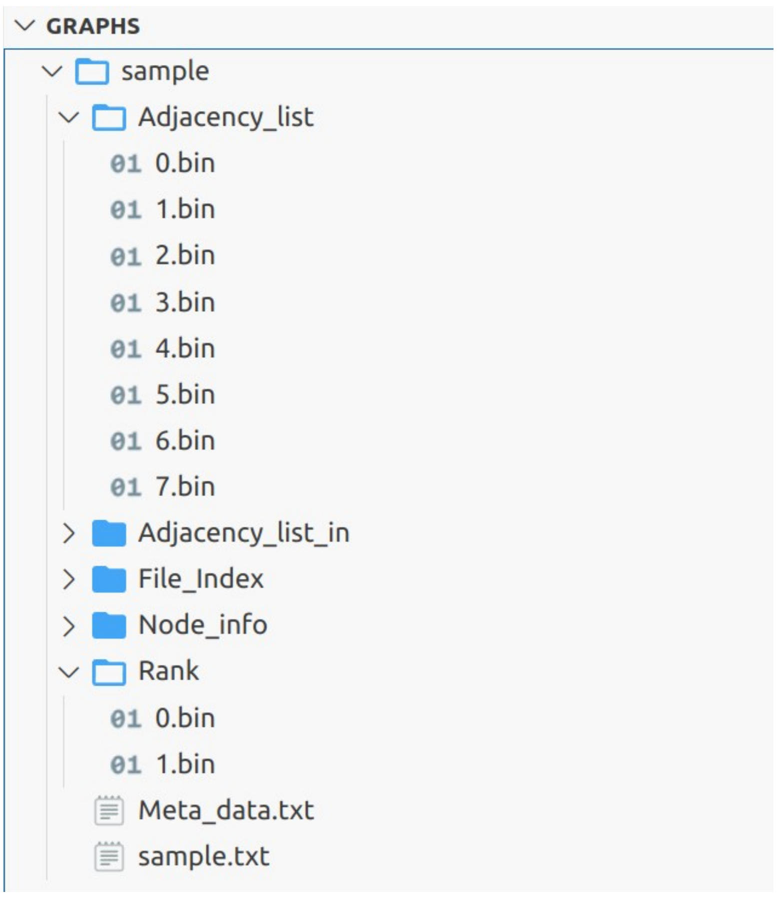
...[adjacency list of node i-1] [n_1 n_2 n_3 n_4 ... n_k] [adjacency list of node i+1]...

n_j = j-th neighbour of node i

Note: *The node-id and the size of the adjacency list of a node is not stored to avoid redundancy as the information can be extracted from the Node-info block.*

3. Architecture of the System

- The architecture of our database consists of a frontend built using HTML and CSS that sends queries and uploaded files via HttpRequest and invokes the Database system built using Python.
- The Database system stores the graph in the following directory structure:



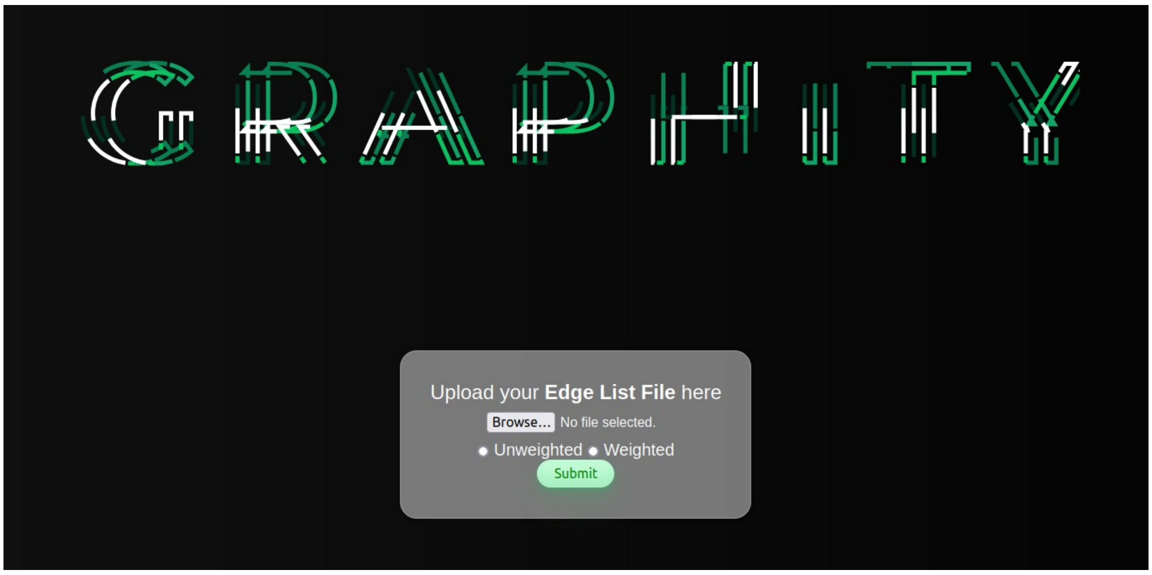
```
GRAPHS
  sample
    Adjacency_list
      0.bin
      1.bin
      2.bin
      3.bin
      4.bin
      5.bin
      6.bin
      7.bin
    Adjacency_list_in
    File_Index
    Node_info
    Rank
      0.bin
      1.bin
    Meta_data.txt
    sample.txt
```

- The Database system pre-processes the uploaded text file and efficiently stores the adjacency list in fixed-size binary files. It also store the Node information, PageRank information and other index structures in the same process .

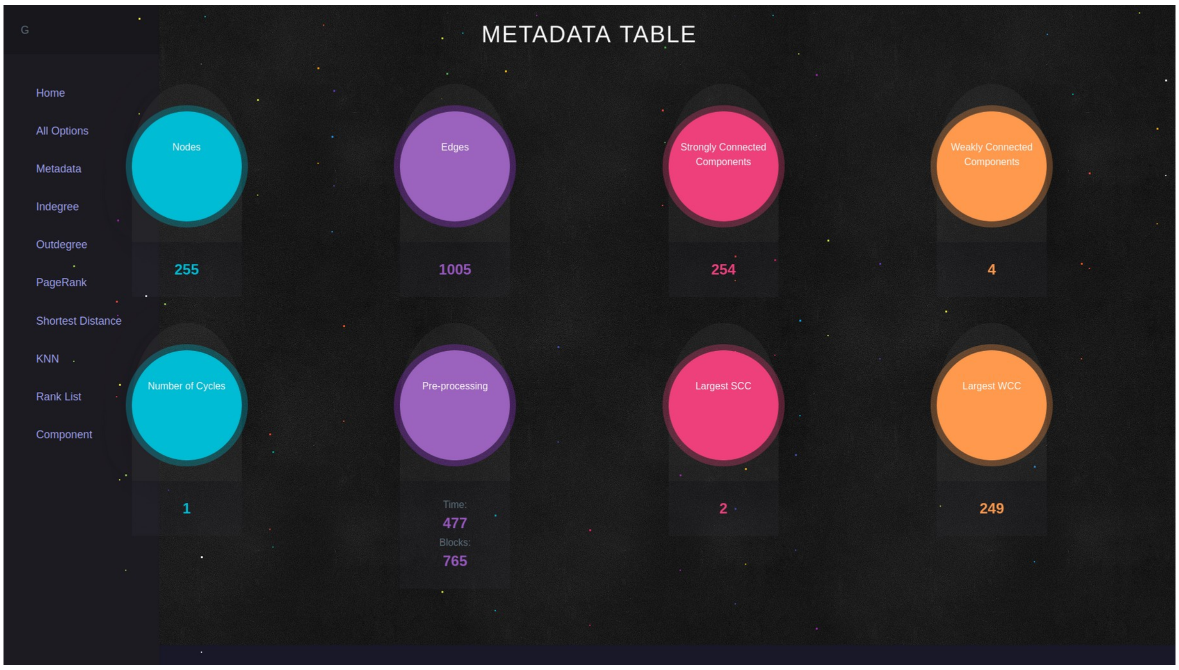
4. Result

4.1 Frontend screenshots

- *Home Page*


GRAPHITY
Upload your **Edge List File** here
Browse... No file selected.
Unweighted Weighted
Submit

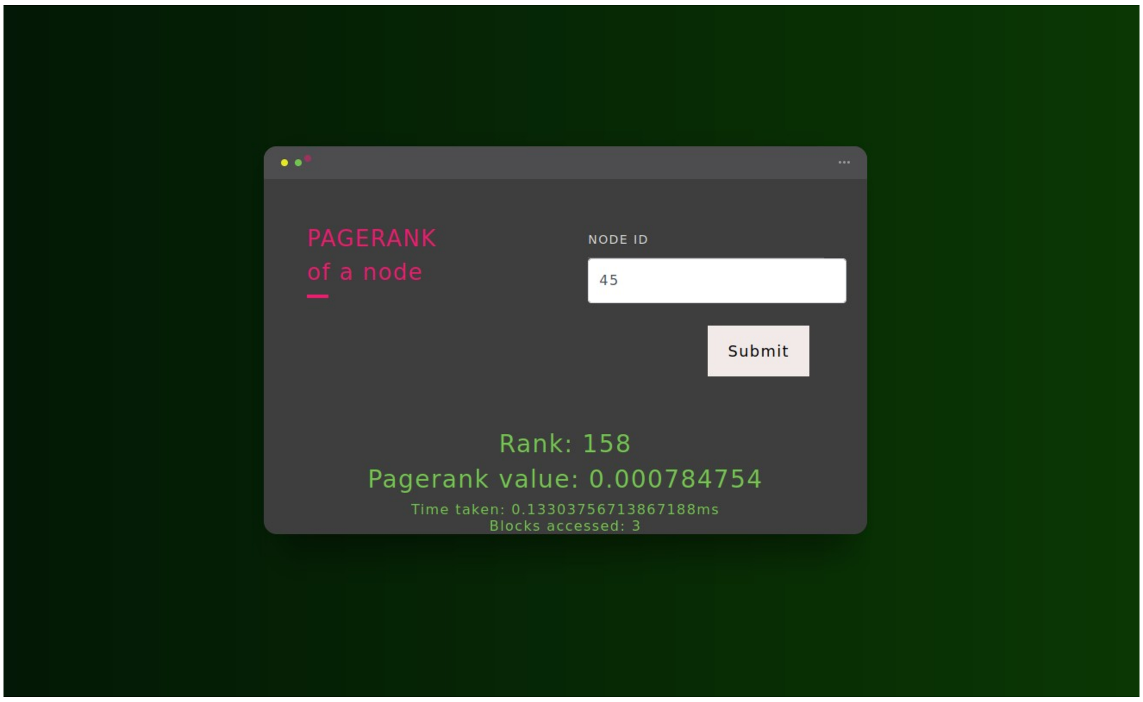
- *Metadata Page*



METADATA TABLE

Nodes	Edges	Strongly Connected Components	Weakly Connected Components
255	1005	254	4
Number of Cycles	**Pre-processing**	**Largest SCC**	**Largest WCC**
1	Time: 477 Blocks: 765	2	249

- *PageRank Query Page*

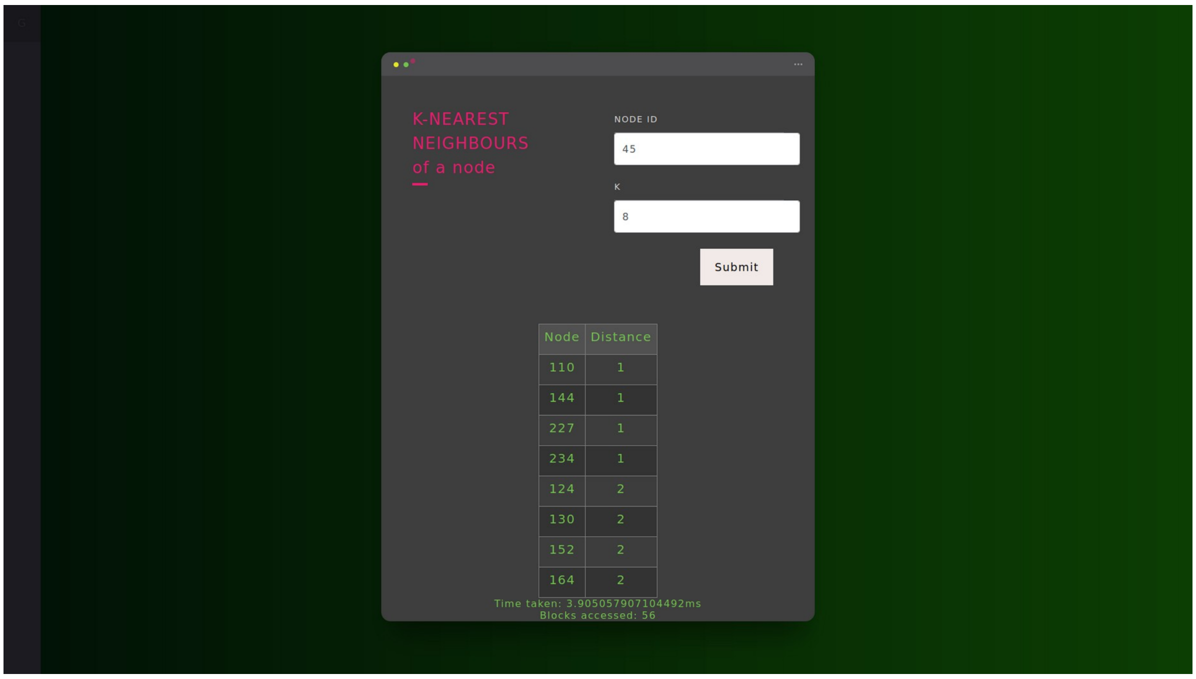

PAGERANK
of a node

NODE ID
45

Submit

Rank: 158
Pagerank value: 0.000784754
Time taken: 0.13303756713867188ms
Blocks accessed: 3

- *K-Nearest Neighbour List Query Page*


K-NEAREST
NEIGHBOURS
of a node

NODE ID
45

K
8

Submit

Node	Distance
110	1
144	1
227	1
234	1
124	2
130	2
152	2
164	2

Time taken: 3.905057907104492ms
Blocks accessed: 56

4.2 Efficiency and Performance

Following are the benchmarks measured against the "*Gnutella peer-to-peer network, August 4 2002*" Graph from the Stanford SNAP database in a AMD Ryzen 7 (5000H series) machine. These are just for some arbitrary queries, exact timings of shortest path and neighbour printing may differ based on the exact queries.

Operation	***Disk Block(s) accessed***	***Time taken(ms)***
Pre-processing and Metadata calculation	2157	21891
Indegree retrieval (n = 55)	2	0.09
Outdegree retrieval (n = 155)	2	0.08
PageRank retrieval (n = 66)	4	0.08
Rank List range query retrieval ([10, 1000])	9	15.34
Shortest Path [26 → 1196]	140	3.77
K-Nearest Neighbour (n = 10, k = 100) retrieval	12325	972
Component analysis (n1=26, n2=1196)	9	0.14

5. References

- Stanford Large Network Dataset Collection: **https://snap.stanford.edu/data/p2p-Gnutella04.html**
- Neo4j Graph Data Science (GDS) Documentation: https://neo4j.com/docs/graph-data-science/current/algorithms/
- PageRank Algorithm: https://neo4j.com/docs/graph-data-science/current/algorithms/
- DBMS Course slides on indexing and hashing: https://cse.iitkgp.ac.in/~ksrao/cou-dbms-2023spring.html