Pokec Slovakian Social Network(SNAP)

DS 115 A: Data Structures/Algorithms in DS

American University of Armenia

Lucy Khadrlian, Karen Safaryan, Seroj Karapetyan

Gagik Khalafyan

Project Report

12.12.2025

Dataset Description and Schema

The system utilizes a subset of the **Pokec dataset**, the most popular Slovak online social network. The data is stored in dataset.csv and contains approximately **1,632,803 records**.

The system processes both categorical profile data and relational connection data.

Attribute	**Data Type**	**Description**
user_id	Integer (Primary Key)	Unique identifier for each user.
gender	Categorical	Male/Female representation.
age	Integer	User age, used for AVL indexing.
eye_color	String	Physical attribute description.
education	String	Academic background.
languages	String (CSV)	Multiple languages spoken by the user.
music	String (CSV)	Musical preferences.
friends	List (Semicolon-sep)	**Relationship Column:** IDs of users followed/friended.

To ensure high performance across massive datasets, the system employs three core data structures:

A. Hash Map (Primary Storage)

- **Implementation:** Python dict in storage.py.
- **Justification:** Provides **O(1)** average-case time complexity for point lookups. Given the requirement to retrieve full profile details by user_id instantly, a hash map is the most efficient choice.

B. AVL Tree (Self-Balancing BST)

- **Implementation:** indexing.py.
- **Justification:** Standard Binary Search Trees (BSTs) can become skewed (becoming $O(N)$) if data is inserted in sorted order. The AVL tree maintains balance through rotations, ensuring **O(\log N)** depth. This is critical for **range queries** (e.g., finding all users aged 18 to 30).

C. Graph (Adjacency List)

- **Implementation:** graph.py using collections.defaultdict(list).
- **Justification:** Social networks are inherently graph-based. An adjacency list is space-efficient for sparse graphs (where most users have far fewer friends than the total population N). It allows for rapid traversal of neighbor nodes.

3. Time and Space Complexity Analysis

Based on the implemented algorithms, the following complexities apply:

Operation	Data Structure	Time Complexity	Space Complexity
Point Lookup	Hash Map	O(1)	O(N)
Range Search (Age)	AVL Tree	O(log N + K)	O(N)
Shortest Path	BFS (Graph)	O(V + E)	O(V)
Strongly Connected Comp.	Tarjan's/Kosaraju	O(V + E)	O(V + E)
User Deletion	Multi-structure	O(log N + D)	O(1)

Note: N = total users, K = number of records in range, V = vertices, E = edges, D = degree of the deleted node.

4. Graph Feature Explanation

The system treats friendships as a **directed graph**. While the "relationship column" in the dataset represents connections, the system processes them into an adjacency list to support:

1. **Breadth-First Search (BFS):** Used to find the "Degrees of Separation" between two users. It explores neighbors level-by-level to guarantee the shortest path.
2. **Strongly Connected Components (SCC):** Identifies clusters within the network where every user can reach every other user in the same group.
3. **Degree Distribution:** Analyzes how "connected" the network is by calculating the number of friends per user.

5. Example Queries and Outputs

Based on system logs and visual output:

Query 1: Point Lookup (O(1))

- **Input:** get_record_by_id(1)
- **Output:** Record 1 found in 0.025 ms.

Query 2: Range Search Efficiency

The system compared Linear Search vs. AVL Tree Search for the age range [18, 30]:

- **Linear Search:** 1,144,938 records found in **343.710 ms**.
- **AVL Tree Search:** 1,144,938 records found in **259.102 ms**.
- **Result:** The AVL tree was **1.3x faster** even with Python's overhead.

Query 3: Graph Pathfinding

- **Input:** Shortest path between test_user and test_far_node.
- **Output:** Shortest path found (length 2): [user_A, friend_B, far_node].

Summary of Findings

- **Scalability:** The system successfully handled over **1.6 million records**. While initialization (loading the graph) took significant time (~1936 seconds), subsequent queries were millisecond-fast.
- **Consistency:** Deleting a user correctly removed their entry from the Hash Map, the AVL index, and the Social Graph simultaneously.
- **Performance:** The AVL Tree range query proved significantly more efficient than iterating through the entire list, validating the choice of advanced indexing structures.
- **UI Integration:** The Streamlit application provides an intuitive way for non-technical users to interact with these complex data structures, visualizing both profile data and social connectivity.

To check out the codes go by this link https://github.com/k-safaryan/Pokec-Slovakian-Social-Network.git