Movies Data Manager Using Data Structures (Fall 2025)

Abdul Ghafoor
Department of AI
FAST-NUCES
Islamabad, Pakistan
ID: 24I-0118

Abdul Azeem
Department of AI
FAST-NUCES
Islamabad, Pakistan
ID: 24I-2013

***Abstract*—This project presents the design and implementation of a Movies Data Manager system aimed at managing a large real-world dataset of movies. The system allows users to search for movies, retrieve details, discover relationships between films, and receive recommendations. A core constraint of this project was the strict prohibition of the C++ Standard Template Library (STL), requiring the manual implementation of fundamental data structures. We utilized AVL Trees for efficient record management, Hash Tables for fast attribute indexing, and Graphs for modeling complex relationships between entities. The system was tested on the IMDb 5000 Movie Dataset, demonstrating the practical application of data structures in processing and analyzing structured data.**

***Index Terms*—Data Structures, C++, AVL Tree, Hash Table, Graph Theory, IMDb Dataset.**

I. Introduction

The organization and retrieval of large-scale data are fundamental challenges in computer science. This project, "Movies Data Manager," was developed as part of the Data Structures course (Fall 2025) at FAST-NUCES. The objective was to build a console-based application capable of parsing, storing, and analyzing movie data without relying on pre-built libraries (STL).

The system processes the IMDb 5000 Movie Dataset [1], providing functionalities such as:

- Efficient searching of movies by title.
- Filtering movies by actors and genres.
- Generating movie recommendations based on similarity.
- Finding the "shortest path" (degrees of separation) between two movies.

II. Dataset and Parsing

The system utilizes the `movie_metadata.csv` file, which contains approximately 5000 records. Key attributes extracted include `movie_title`, `director_name`, `actor_1_name`, `genres`, `imdb_score`, and `title_year`.

A custom CSV parser was implemented to handle data ingestion. The parser specifically manages edge cases such as fields enclosed in quotes containing commas (e.g., "Action, Adventure"). It reads the file line-by-line, tokenizes the string, and populates a `MovieNode` object for insertion into the data structures.

III. System Architecture and Class Design

The architecture is modular, with distinct classes handling specific data management tasks.

A. Basic Containers

1) LinkedList (Template): A generic Singly Linked List was implemented to serve as the building block for other structures (Stacks, Queues, Hash Table chaining, and Adjacency Lists). It supports insertion, deletion, and traversal.

2) Stack and Queue: Both structures were implemented using the Linked List. The `Queue` is primarily used for the Breadth-First Search (BFS) algorithm in the graph traversal features.

B. Core Data Structures

1) MovieNode: This struct acts as the primary data carrier. It stores all movie metadata (title, year, rating) and contains Linked Lists for actors and genres.

2) AVL Tree (Movie Repository): To ensure $O(\log n)$ time complexity for searching, insertion, and deletion of movies by title, an AVL Tree was implemented.

- **Balancing:** The tree automatically balances itself using rotations (LL, RR, LR, RL) based on the balance factor of nodes.
- **Key:** The movie title serves as the key.
- **Usage:** This is the primary storage for all movie records.

3) Hash Table (Indexing): To enable fast lookup ($O(1)$ average case) for non-unique keys like **Actors** and **Genres**, a Hash Table was utilized.

- **Hash Function:** A polynomial rolling hash or simple summation of ASCII values modulo the table size.
- **Collision Resolution:** Separate Chaining using Linked Lists.
- **Usage:** The system maintains indices where an actor's name maps to a list of pointers to `MovieNodes` featuring that actor.

4) Graph (Relationships): A Graph data structure models the relationships between movies.

- **Vertices:** Movies, Actors, and Genres are treated as nodes in the graph context.

- **Edges:** Relationships are weighted or unweighted connections. For example, if two movies share the same director or actor, an edge is created between them.
- **Representation:** Adjacency List.

IV. Algorithms and Functionality

A. Search and Retrieval

Searching for a movie utilizes the AVL Tree traversal. The user input is formatted (trimmed and lowercased) and compared against the tree nodes. Searching by Actor or Genre queries the Hash Table, returning a linked list of results instantly.

B. Recommendations

The recommendation engine is based on Graph connectivity. When a user requests recommendations for a specific movie:

1) The system locates the movie node in the graph.
2) It traverses the immediate neighbors (adjacent nodes).
3) Movies connected via strong attributes (e.g., same director, shared high-billing actors) are returned as suggestions.

C. Shortest Path (Degrees of Separation)

To find the connection between two movies (e.g., Movie A $\rightarrow$ Actor X $\rightarrow$ Movie B), the system implements the **Breadth-First Search (BFS)** algorithm.

- BFS guarantees the shortest path in an unweighted graph.
- The algorithm utilizes the custom `Queue` class to explore layer by layer until the target movie is found.
- To reconstruct the path, a `parent` array or map tracks the predecessor of each visited node.

V. Conclusion

The Movies Data Manager successfully meets the requirements of the term project by implementing a robust system without STL. The use of AVL Trees ensures the system remains performant even as the dataset grows, while the Hash Table allows for instant filtering. The Graph implementation effectively models real-world relationships, enabling advanced features like recommendations and pathfinding. This project reinforced the importance of selecting appropriate data structures based on access patterns and performance requirements.

References

[1] IMDb 5000 Movie Dataset, https://www.kaggle.com/datasets/carolzhangdc/imdb-5000-movie-dataset.