JAYPEE INSTITUTE OF INFORMATION TECHNOLOGY, NOIDA

B. Tech ODD SEM 2025

15B17CI371

DATA STRUCTURES PROJECT

TITLE OF THE PROJECT:

Smart Music Player

SUBMISSION BY:

NAME	ENROLLMENT NUMBER
Pari Jindal	2401030147
Kopal Kashyap	2401030160
Malya Khajuria	2401030158

SUBMISSION TO:

Ms. Anupama Padha

Mr. Varun Srivastava

Ms. Dhanlakshmi G.

TABLE OF CONTENTS

SUMMARY

Brief Overview

The **Smart Music Player** is a console-based application developed in C++ that demonstrates how modern music player features can be implemented using **Data Structures**. The project focuses on the internal working of a music player rather than audio playback. It allows users to add songs, create and explore playlists, search for songs, and navigate through them easily.

Along with the basic features, the system also includes advanced functionalities such as maintaining a play history, handling an upcoming song queue, showing recommendations based on most-played songs, displaying top-rated songs using a custom heap, and representing song similarity using a graph. The goal of the project is to show how different data structures work together to create an efficient and interactive system.

Objectives

- **To implement real-world music player features using appropriate data structures**, ensuring efficient storage, search, navigation, and recommendation of songs.

- **To demonstrate the practical use of major data structures** such as Linked Lists (for playlist navigation), Stacks (for play history), Queues (for upcoming songs), HashMaps (for fast lookup), Priority Queues (for recommendations), Graphs (for similar songs), and Heaps (for top-rated songs).

- **To build a modular, interactive, and user-friendly software application** that shows how multiple components can work together in a single system.

- **To enhance problem-solving and design skills** by applying DSA concepts to create a complete project that reflects real-world application behaviour.

INTRODUCTION

Background and Context

Music applications (e.g., Spotify, Wynk, YouTube Music) internally rely heavily on data structures to store songs, recommend tracks, manage playlists, and maintain playback flow.
This project replicates these mechanisms using academic DSA concepts.

Problem Statement

Traditional playlist systems are simple and do not support advanced features like history tracking, upcoming song management, similarity-based recommendations, or maintaining top-rated songs.
The challenge was to design a system that can *efficiently* handle multiple functionalities using the correct data structures.

Scope of the Project

The system supports:

- Adding, searching, and viewing songs
- Playlist management
- Song navigation (next/previous/start/end)
- Recommendation system
- History manager
- Upcoming queue
- Similar songs and DFS traversal
  It is focused on DSA application and does not include audio playback.

DESIGN AND IMPLEMENTATION

Class Diagrams (Description)

1. **Song Class**
    - Stores metadata (ID, title, artist, rating, duration, play count).
    - Provides getters, setters, and display functions.
2. **Playlist (Doubly Linked List)**
    - Each node has pointers to previous and next song.
    - Supports add, remove, navigate next/prev, display.
    - Maintains a *current pointer* to track currently playing song.
3. **Music Library (HashMap)**
    - Unordered map storing songs by ID and title.
    - Enables fast O(1) lookup.
4. **History Manager (Stack)**
    - Maintains list of previously played songs.
    - LIFO: recently played songs come first.
5. **Upcoming Manager (Queue)**
    - Stores “Play Next” songs in FIFO order.
6. **Recommendation Engine (Priority Queue)**
    - Max-heap based on play count.
    - Frequently played songs are recommended first.
7. **Song Graph (Adjacency List)**
    - Represents song similarity relationships.

- Supports DFS traversal.

8. **Rating Max Heap (Custom Heap)**
    - Stores songs according to highest rating.
    - Gives top-rated songs without using STL priority_queue.

Data Structures Implemented

Feature	Data Structure Used	Reason
Playist	Doubly Linked List	Bidirectional navigation
Play History	Stack	LIFO for last played items
Upcoming “Play Next”	Queue	FIFO ordering
Music Library	Hash Map	Instant O(1) search
Recomendations	Priority Queue	Higher Play Count -> Higher Priority
Song Similarity	Graph (Adjacency List)	Efficient Connections
Top Rated Songs	Custom Max Heap	Manual Heap Implementation

Algorithms Used (Conceptual)

1. Searching

- **ID Search:** O(1) average using HashMap
- **Title Search:** O(1) mapping title → vector of songs

2. Sorting / Priority Logic

- Playback Count → Priority Queue (max-heap)
- Ratings → Custom Max Heap based on:

- Higher rating
- In tie → lower ID

3. Graph Traversal

- DFS is used to explore similar songs connected in the graph.

4. Linked List Operations

- Insert at start
- Insert at end
- Insert at position
- Delete by ID
- Move forward / backward

Output Screens (Described)

The program displays a complete **Main Menu** with options:

1. Add song
2. Show library
3. Show playlist
4. Search by ID
5. Search by title
6. Play current
7. Next song
8. Previous song
9. Go to start
10. Go to end

11. Add to upcoming
12. Show upcoming queue
13. Show history
14. Show recommendations
15. Add similarity
16. Show similar songs
17. DFS traversal
18. Top rated songs
19. Remove song
    0.Exit

Each action displays well-formatted descriptive output such as:

- Song details
- Playlist with current pointer highlighted
- History stack
- Queue list
- Recommendations by play count
- Graph connections
- Rated songs ranking

OUTPUTS

```
==========================================
        SMART MUSIC PLAYER - MAIN MENU
==========================================
 1. Add Song (Library + Playlist)
 2. Show Music Library
 3. Show Playlist
 4. Search Song by ID
 5. Search Song by Title
 6. Play Current Song
 7. Next Song
 8. Previous Song
 9. Go to Start of Playlist
10. Go to End of Playlist
11. Add Song to Upcoming Queue
12. Show Upcoming Queue
13. Show Play History
14. Show Recommendations (Most Played)
15. Add Song Similarity (Graph Edge)
16. Show Similar Songs for an ID
17. DFS Traversal in Song Graph
18. Show Top Rated Songs (Custom Heap)
19. Remove Song from Playlist
 0. Exit
==========================================
Enter your choice: 1

Enter song title: Shape of You
Enter artist name: Ed Sheeran
Enter duration in seconds: 230
Enter rating (1 to 5): 5
Song added with ID: 1

Press ENTER to continue...
```

```
Enter your choice: 2

===== MUSIC LIBRARY (3 songs) =====
[3] Believer - Imagine Dragons | Duration: 210s | Rating: 5 | Played: 0 time
s
[2] Perfect - Ed Sheeran | Duration: 240s | Rating: 4 | Played: 0 times
[1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 | Played: 0 times

Press ENTER to continue...
```

```
Enter your choice: 3

===== PLAYLIST (3 songs) =====
>> [1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 | Played: 0 ti
mes
   [2] Perfect - Ed Sheeran | Duration: 240s | Rating: 4 | Played: 0 times
   [3] Believer - Imagine Dragons | Duration: 210s | Rating: 5 | Played: 0 t
imes

Press ENTER to continue...
```

```
Enter your choice: 6

===== NOW PLAYING =====

-------------------------------------
 Song ID      : 1
 Title        : Shape of You
 Artist       : Ed Sheeran
 Duration (s) : 230
 Rating       : 5/5
 Play Count   : 1
-------------------------------------

Press ENTER to continue...
```

```
Enter your choice: 11

Enter song ID to add into upcoming queue: 3
Song added to upcoming queue.

Press ENTER to continue...
```

```
Enter your choice: 7

Playing from upcoming queue:

-------------------------------------
 Song ID      : 3
 Title        : Believer
 Artist       : Imagine Dragons
 Duration (s) : 210
 Rating       : 5/5
 Play Count   : 1
-------------------------------------

Press ENTER to continue...
```

```
Enter your choice: 13

===== PLAY HISTORY (Top is latest) =====
1. [3] Believer - Imagine Dragons | Duration: 210s | Rating: 5 | Played: 1 t
imes
2. [1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 | Played: 2 ti
mes
3. [2] Perfect - Ed Sheeran | Duration: 240s | Rating: 4 | Played: 1 times
4. [1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 | Played: 2 ti
mes

Press ENTER to continue...
```

```
Enter your choice: 14

===== TOP RECOMMENDED SONGS =====
Played 2 times: [1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 |
 Played: 2 times
Played 1 times: [1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 |
 Played: 2 times
Played 1 times: [3] Believer - Imagine Dragons | Duration: 210s | Rating: 5
| Played: 1 times
Played 1 times: [2] Perfect - Ed Sheeran | Duration: 240s | Rating: 4 | Play
ed: 1 times

Press ENTER to continue...
```

```
Enter your choice: 15

Enter Song ID 1: 1
Enter Song ID 2: 2
Similarity added between 1 and 2.

Press ENTER to continue...
```

```
Enter your choice: 16

Enter Song ID to view similar songs: 1

===== SIMILAR SONGS FOR ID 1 =====
Similar to -> Song ID: 2

Press ENTER to continue...
```

```
Enter your choice: 17

Enter start Song ID for DFS traversal: 1

DFS Traversal from Song ID 1: 1 2

Press ENTER to continue...
```

```
Enter your choice: 18

===== TOP RATED SONGS (Using Custom Max Heap) =====
Rating 5 -> [1] Shape of You - Ed Sheeran | Duration: 230s | Rating: 5 | Pla
yed: 2 times
Rating 5 -> [3] Believer - Imagine Dragons | Duration: 210s | Rating: 5 | Pl
ayed: 1 times
Rating 4 -> [2] Perfect - Ed Sheeran | Duration: 240s | Rating: 4 | Played:
1 times

Press ENTER to continue...
```

```
Enter your choice: 0
Exiting Music Player. Goodbye!

Process returned 0 (0x0)   execution time : 185.813 s
Press any key to continue.
```

CONCLUSION

Summary & Achievement of Objectives

The Smart Music Player successfully demonstrates the application of multiple core data structures in a single integrated system.
It meets all objectives — searching, playlist manipulation, traversal, recommendations, similarity graph, and stack/queue management.

The project shows how **DSA concepts can replicate real-world systems** and how choosing the right data structure can greatly improve efficiency.

Future Work & Recommendations

- Add file storage to save songs permanently.
- Add GUI (Qt, SFML, or Web UI).
- Implement audio playback using external libraries.
- Add shortest-path recommendations using graph algorithms.
- Add user accounts and personalized playlists.

REFERENCES

- “Data Structures and Algorithms in C++” – Adam Drozdek
- C++ STL Documentation – cppreference.com
- GeeksforGeeks – Data Structures Tutorials
- Lecture notes from Data Structures course