Long Assignment 1:
Time-Travelling File System

1 Introduction

In this assignment, you will implement a simplified, in-memory version control system inspired by Git. Your system will manage versioned files with support for branching and historical inspection. The primary goal is to apply your understanding of Trees, HashMaps, and Heaps to a complex, practical application.

2 System Architecture

Your file system will manage a collection of files, each with its own version history represented as a tree. The system must utilize the following core data structures:

- **Tree:** To maintain the version history of each file. A node in the tree represents a specific version.
- **HashMap:** To provide fast, $O(1)$ average-time lookups of versions by their unique ID.
- **Heaps:** To efficiently track system-wide file metrics, such as the most recently or frequently edited files.

Note: You must implement the above data structures (along with the operations needed for this project) yourself from scratch, i.e., you are **not** allowed to use C++ Libraries which already implement these data structures.

File and Version Data Models

Each file object in your system will contain the following members:

```
// File Structure
TreeNode* root; // Your implementation of the tree
TreeNode* active_version;
map<int, TreeNode*> version_map; // Your implementation of the HashMap
int total_versions;
```

Each version (a node in the tree) must store the following information:

```
// Version (TreeNode) Structure
int version_id;
string content;
string message; // Empty if not a snapshot
time_t created_timestamp;
time_t snapshot_timestamp; // Null if not a snapshot
TreeNode* parent;
vector<TreeNode*> children;
```

3 Command Reference

Your program must read and execute a series of commands from `stdin`.

3.1 Core File Operations

`CREATE <filename>` Creates a file with a root version (ID 0), empty content, and an initial snapshot message. Note that the root is marked as a snapshot.

`READ <filename>` Displays the content of the file's currently active version.

`INSERT <filename> <content>` Appends content to the file. This creates a new version if the active version is already a snapshot; otherwise, it modifies the active version in place.

`UPDATE <filename> <content>` Replaces the file's content. Follows the same versioning logic as `INSERT`.

`SNAPSHOT <filename> <message>` Marks the active version as a snapshot, making its content immutable. It stores the provided message and the current time.

`ROLLBACK <filename> [versionID]` Sets the active version pointer to the specified `versionID`. If no ID is provided, it rolls back to the parent of the current active version.

`HISTORY <filename>` Lists all snapshotted versions of the file chronologically, which lie on the path from active node to the root in the file tree, showing their ID, timestamp, and message.

3.2 System-Wide Analytics

`RECENT_FILES [num]` Lists files in descending order of their last modification time restricted to the first `num` entries.

`BIGGEST_TREES [num]` Lists files in descending order of their total version count restricted to the first `num` entries.

4 Key Semantics

- **Immutability:** Only snapshotted versions are immutable. Non-snapshotted versions can be edited in place.
- **Versioning:** Version IDs are unique per file and assigned sequentially, starting from 0.

You must handle the cases of incorrect / inconsistent input as you deem appropriate (how you are handling must be mentioned in the README).

5 Submission

This is an individual assignment. You must submit a compressed file (.zip/.rar) containing your project code (.cpp, .hpp files, if any) along with a working shell script to compile your code. You must also add a README containing the instructions on how to run your code and use different commands. **Note** that the user must be able to input commands at runtime, i.e., from `stdin` and the commands must follow the given syntax.

The deadline for the submission is September 11, 23:59 IST (Thursday). The submission will be on moodlenew.

6 Evaluation

The evaluation for the project will be based on a Viva (dates to be announced later) which will involve (but not limited to) questions regarding your code, checking output on some specific sequence of commands, quality of the code etc.