Name: Irtaza Jawad, Ali Abdullah, Syed Rafay Hassan.

Roll No: 21i-1376, 22i-1857,22i-1955,.

Approach:

The project focuses on implementing a concurrent MapReduce framework to efficiently process textual data in parallel. This framework splits input into smaller chunks, processes each chunk in parallel threads during the Map Phase, and aggregates results in the Reduce Phase. Mutex locks ensure thread-safe operations, maintaining data integrity throughout the workflow.

Components:

The solution is based on the MapReduce paradigm, which involves:

1. Splitting Input: Breaking down the input string into manageable chunks.
2. Mapping: Processing each chunk to generate intermediate results (word counts).
3. Reducing: Aggregating the intermediate results to compute the final counts for each word.

Each step is designed to utilize multithreading for parallel processing, improving efficiency and scalability.

Custom Data Structures

- wordCountPair: Represents intermediate key-value pairs, containing a key (word) and its count.
- reducedResult: Represents the final aggregated results, containing a key (word) and its total count.

Phases of MapReduce Framework:

1. Map Phase

- Function: executeMapPhase
- Processes a chunk of text to identify and count unique words.
- For each word:
  - If the word exists in the processing vector, increment its count.
  - Otherwise, create a new wordCountPair for it.
- Concurrency:
  - Multiple threads independently process different chunks.
  - A mutex ensures thread safety while modifying the shared processing vector.

2. Shuffle Phase

- This is implicitly handled by the processing vector during the Map Phase, which groups all key-value pairs by key.

3. Reduce Phase

- Function: executeReducePhase
- Aggregates counts for each unique word across all key-value pairs.
- For each unique key:
  - Find all occurrences of the key in the processing vector.

    - Sum the associated counts and store the result in the finalResults vector.
- Concurrency:
    - Multiple threads process different keys in parallel.
    - A mutex ensures thread safety while modifying the shared finalResults vector.

Main Function Workflow:

1. **Input Splitting:**
    - Divides the input string into chunks using the divideIntoChunks function.
    - Chunk size is determined dynamically based on the input size and phase requirements.

2. **Map Phase Execution:**
    - Creates threads for each chunk.
    - Processes chunks in parallel using executeMapPhase.

3. **Reduce Phase Execution:**
    - Creates threads for each unique key.
    - Aggregates results in parallel using executeReducePhase.

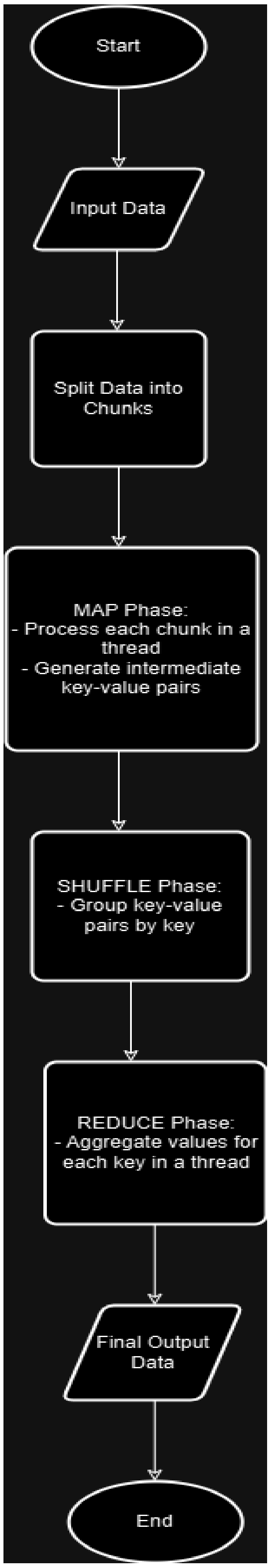
4. **Displaying Results:**
    - Outputs the aggregated word counts in the form (word, count).

Test Cases and User Interaction:
- The program offers predefined test cases (easy, medium, hard) and the option for custom input.
- Chunk sizes are adjusted to match the complexity of the test case.

Thread-Safe Execution:
- mutex locks ensure synchronized access to shared resources (processing and finalResults vectors), preventing race conditions.

Key Advantages
- Scalability: The multithreaded approach handles large inputs efficiently.
- Flexibility: Users can select predefined cases or provide custom inputs.
- Thread Safety: Ensures data integrity during parallel processing.

Flow Diagram:


Start
Input Data
Split Data into Chunks
MAP Phase:
- Process each chunk in a thread
- Generate intermediate key-value pairs
SHUFFLE Phase:
- Group key-value pairs by key
REDUCE Phase:
- Aggregate values for each key in a thread
Final Output Data
End