CS 426: Introduction to Blockchains
Assignment 1

Assignment Details

- **Assignment Given On:** 29th Jan 2025
- **Due Date:** 5th Feb 2025, on Google Classroom(10% penalty per 24-hour period after the due date).
- Programming Language to be used for this assignment is $C++$
- **Submission Format:**
  - Create a directory named `<Your_Roll_Number>`.
  - Copy all program files (source code) and output files into this folder.
  - Include a `Makefile` or script file for compilation and execution.
  - Copy output results as comments at the end of the source code, or include screenshots in the directory.
  - Zip the directory and submit on Google Classroom .
- Late submissions through email will not be accepted.
- Test your code thoroughly and follow a consistent coding style. Provide comments for clarity.
- **DO NOT USE CHATGPT OR COPY FROM ANYWHERE IN THE INTERNET OR COPY FROM ANOTHER STUDENT.** 100% **penalty if the submitted source code is found to be copied.**

Part 1: Block and Blockchain Class Implementation

Task Description

In this assignment, you will implement the class of a Block and the class of a Blockchain.

- Implement a **Block** class similar to that in Bitcoin. Each block must include:
  - **Parent Hash:** A reference to the hash of the previous block (256-bit hash).
  - **Nonce:** A random integer for proof-of-work mining (will be used in the later assignments).

- **Difficulty:** A threshold value for mining.
- **Timestamp:** Time of block creation.
- **Merkle Root:** For this assignment, Calculate merkle root using all transactions in the block concatenated and generating its hash output using SHA-256. Complete implementation will be done in upcoming assignment.
- **Transactions:** A list of dummy transactions (two per block for testing).
- **Hash:** The unique identifier for the block, calculated from its header fields. You may use SHA-256 function to generate the hash value.
- **Functions:** You may implement 2 functions for calculating hash of a block and calculating the merkle root hash value of the transactions.

- Implement a **Blockchain** class that:
  - contains all the blocks, implemented using a hash map.
  - Initializes the blockchain with a **Genesis Block** containing fixed values.
  - Provides an `addBlock()` function to add new blocks to the chain.
  - Provides an `displayBlock()` function to display block details.
  - Maintains the **Tip** using a function of the blockchain which returns the latest block hash value in the longest chain.
  - Implements a `displayBlockchainHashes()` function to return all block hashes from the genesis block to the tip.

Main Function:

- Initializes the blockchain with a genesis block.
- Inserts 50 additional blocks with dummy transactions.
- Displays:
  - The current tip of the blockchain.
  - The hashes of all blocks in the longest chain.

Input and Output Format

Input: No external input is required. The program generates dummy transactions internally.

Output: The program should print the following:

- Confirmation of the genesis block creation.
- Information about each block (including the genesis block) added to the blockchain, including:
  - Parent hash
  - Nonce
  - Difficulty

  - Timestamp
  - Merkle root
  - Hash
- The current tip of the blockchain after all blocks are inserted.
- The hashes of all blocks in the longest chain.

Part 2: Concurrent Mining with Thread Pool

In this part of the assignment, you implement threading on the classes that you have built in Part 1. This refers to concurrent miners trying to add new blocks to the blockchain.

Task Description

- Implement a thread pool to simulate 5 miners concurrently mining blocks.
- Each miner adds 10 blocks to the blockchain.
- Each miner is represented as an independent thread.
- A random delay is introduced between block insertions using `std::this_thread::sleep_for`.
- All miners share the blockchain object.
- The order of block insertion is randomized to mimic real-world mining.

Main Function:

- Initializes the blockchain with a genesis block.
- Launches 5 miner threads.
- Each thread independently adds 10 blocks to the blockchain.
- Displays:
  - The current tip of the blockchain.
  - The hashes of all blocks in the longest chain.

Synchronization

- No explicit synchronization (e.g., locks) is implemented, as thread safety is not required in this part of the assignment.
- Each thread independently accesses the blockchain.

FAQ

- **Can the fields of Header/Content of the blockchain be made public?** Yes, they can be made public. Alternatively, you can define getter functions.
- **What values should the fields in the genesis block have?** The fields (e.g., nonce, difficulty, timestamp, parent) should be fixed and not random. Use values such as:
  - Parent: `0x00...00`
  - Nonce: `0`
  - Difficulty: `0xff...ff`
  - Timestamp: `0`

Grading Criteria

- **Part 1:**
  - Script file (Makefile or equivalent): 10%
  - Block Class: 25%
  - Blockchain Class: 25%
  - Live demonstration of the working program: 10%
- **Part 2:**
  - Thread Implementation: 20%
  - Live demonstration of the working program: 10%

References

- Blockchain Explorer