Concurrent Block Processor Assignment (Public)

Context

A core component of the Flow Blockchain is the consensus algorithm, which processes blocks and votes. Once a block receives enough votes, it is considered accepted and is incorporated into the finalized blockchain data structure.

A block is a suggested extension to the blockchain. The `id` field is a globally unique identifier, represented as a hex string. The `view` field is a positive integer which defines an ordering for the set of all blocks (views start with `1`, the block with view `n` is before the block with view `n+1`, and no views are skipped). Here is a sample JSON representation of a block, as we will use in this problem.

```
{
    "id": "a65e9803bb37256c4a663a5c1b",
    "view": 1234,
}
```

A vote is an affirmation from one of the consensus participants that it accepts a particular block as a valid extension. Here is a sample JSON representation of a vote, as we will use in this problem.

```
{
    "block_id": "a65e9803bb37256c4a663a5c1b",
}
```

In this problem, we will build a simplified version of the sub-component which processes blocks and votes.

Objective

Develop a Block Processor program that processes blocks and votes through an HTTP API. When a block is **accepted**, the Block Processor should print the block's ID and view to STDOUT. **Accepted** blocks should be printed exactly once and in ascending view order.

A block `B` is considered **accepted** when:

- we have observed a vote for the block (i.e. a vote `V` such that `V.block_id == B.id`)
- all blocks with views smaller than `B.view` have been **accepted**

Requirements

- Implement an HTTP API that accepts blocks and votes in JSON format.
- The API should correctly handle multiple concurrent requests, with messages arriving out of order.
- The sequence of accepted blocks must have view numbers that are strictly monotonically increasing in increments of 1 (without gPraps).
- Once a block is accepted, print its ID and view to STDOUT.

Submission Guidelines

- While we prefer GoLang, we recommend using the programming language you are most comfortable and productive with.
- Your code should be clean, well-documented, and follow best practices for the chosen language.
- Consider and handle potential edge cases in your implementation.
- 🚨 Please package your submission as follows:
  - Enclose a README with clear instructions on
    - Program environment setup, dependencies etc
    - How to start and/or interact with the program
    - Any assumptions your implementation makes about the problem
  - If you implemented tests, please include instructions how to run them.

- Please share your submission as a private Github repo; invite to be sent to @franklywatson

This assignment should hopefully take not more than an afternoon. If you have any questions or need clarification we are happy to help you over email!