**Flow Distribution Algorithm - Documentation**

**Overview**

The **Flow Distribution Algorithm** is designed to allocate users to astrologers in a fair and balanced manner, while also allowing for increased flexibility in assigning top astrologers. The goal is to distribute users evenly among all astrologers and provide a way to prioritize specific astrologers based on business rules or user preferences. The system is built using **Node.js** and **Express.js**, with an in-memory data structure for simplicity.

**Design Principles**

1. **Fair Distribution**: Users are allocated to astrologers based on the number of current connections, ensuring that astrologers with fewer connections are prioritized. This results in an even distribution of users.
2. **Priority Handling for Top Astrologers**: A mechanism is built to flag "top astrologers" who are given higher priority by allowing them to handle more users. Top astrologers can receive 2 users for every user assigned to a regular astrologer.
3. **Scalability**: The algorithm is designed to handle up to 2000-3000 users daily with a pool of around 500 astrologers.
4. **Efficiency**: The algorithm efficiently assigns users in a time complexity of **O(n log n)** due to sorting the astrologers by the number of current connections.

**Data Structures**

* **Astrologer**:
  + id: Unique identifier for the astrologer.
  + connections: The current number of users assigned to the astrologer.
  + isTop: Boolean value indicating whether the astrologer is marked as a top astrologer.
  + maxConnections: Maximum number of connections the astrologer can handle.
* **User**:
  + id: Unique identifier for the user.
  + preference: User's preference, e.g., general, which could be used for future enhancements to assign astrologers based on preference.

**Flow Distribution Algorithm**

The algorithm sorts the astrologers based on their current number of connections and assigns users to the astrologer with the least number of connections. If the astrologer is marked as isTop, they receive two connections for every one user assigned.

**Algorithm Logic**:

1. Sort astrologers by connections in ascending order.
2. Find the astrologer with the least connections.
3. Check if the astrologer can take more users (i.e., connections < maxConnections).
4. If the astrologer is a top astrologer (isTop: true), increment their connections by 2.
5. If the astrologer is a regular astrologer, increment their connections by 1.
6. Assign the user to the astrologer.

**API Endpoints**

**1. Add Astrologer**

* **Endpoint**: POST /add-astrologer
* **Description**: Adds an astrologer to the pool.
* **Request Body**:

json

Copy code

{

"isTop": true,

"maxConnections": 15

}

* **Response**:

json

Copy code

{

"id": 1,

"connections": 0,

"isTop": true,

"maxConnections": 15

}

**2. Assign User**

* **Endpoint**: POST /assign-user
* **Description**: Assigns a user to an astrologer based on the flow distribution algorithm.
* **Request Body**:

json

Copy code

{

"preference": "general"

}

* **Response**:

json

Copy code

{

"assignedAstrologer": {

"id": 1,

"connections": 2,

"isTop": true,

"maxConnections": 15

}

}

**3. Toggle Top Astrologer Status**

* **Endpoint**: POST /toggle-top-astrologer/:id
* **Description**: Toggles the isTop status of a given astrologer.
* **Parameters**:
  + id: The ID of the astrologer to toggle.
* **Response**:

json

Copy code

{

"message": "Astrologer 1 top status toggled."

}

**4. Get All Astrologers**

* **Endpoint**: GET /astrologers
* **Description**: Retrieves all astrologers and their current statuses.
* **Response**:

json

Copy code

[

{

"id": 1,

"connections": 2,

"isTop": true,

"maxConnections": 15

}

]

**How to Run the Application**

1. **Install Dependencies**:

bash

Copy code

npm install

1. **Run the Application**:

bash

Copy code

node index.js

1. **Test the Endpoints**:
   * Use Postman or cURL to test the API endpoints.
   * Example for adding an astrologer:

bash

Copy code

curl -X POST http://localhost:3000/add-astrologer -H "Content-Type: application/json" -d '{"isTop":true, "maxConnections": 15}'

**Testing**

Unit tests are written using **Mocha** and **Chai**. The tests validate the core functionality of the flow distribution algorithm.

To run the tests:

1. Install Mocha and Chai (if not installed):

bash

npm install --save-dev mocha chai

1. Run the tests:

bash

npx mocha test.js

Example test cases cover:

* Fair distribution of users among astrologers.
* Correct handling of isTop astrologers.
* Boundary cases for when no astrologers are available.

**Conclusion**

This flow distribution algorithm ensures a balanced distribution of users to astrologers, allowing for both fairness and flexibility when prioritizing top astrologers. It is easily extendable to handle future requirements, such as preference-based assignments or more complex load management strategies.

Top of Form