Data Structures:

- ☑ Implement a Hash Table with a size of 127.
- ☑ Implement a Balanced AVL Tree as the data structure within each Hash Table bucket.

Data Management:

- ☑ Define a data structure to represent a parcel, including destination (string), weight (integer), and valuation (float).
- ☑ Allocate memory dynamically for the destination string within the parcel structure.

File I/O:

- ☑ Read data from a file named "couriers.txt".
- ☑ Each line in the file should contain three comma-separated values: destination (string), weight (integer), and valuation (float).
- ☑ Ensure data format adheres to the specified limitations:
    - ☑ Destination: Maximum 20 characters
    - ☑ Weight: Range 100 grams to 50,000 grams
    - ☑ Valuation: Range $10 to $2,000

Hash Function:

- ☑ Implement the "djb2 function" to generate a unique hash value for each destination string.
- ☑ Use the hash value to determine the appropriate bucket index in the Hash Table for storing the parcel data.

AVL Tree Operations:

- ☑ Implement functions for inserting, searching, and traversing the AVL Trees within each Hash Table bucket.
- ☑ Ensure AVL Tree operations maintain balance after modifications (insertion/deletion) to guarantee efficient searching.

User Interface (Basic):

- ☑ Develop a basic user interface to allow interaction with the data.
    - ☑ Options should include:
        - ☑ Reading data from the "couriers.txt" file.
        - ☑ Searching for parcels based on destination (country name).
        - ☑ Displaying various outputs based on user selection (e.g., total weight or valuation for a specific destination or range of destinations).

Coding Practices:

- ☑ Adhere to best practices for code readability, maintainability, and efficiency.
- ☑ Include comments to explain the purpose of different code sections.
- ☑ Use meaningful variable names.
- ☑ Follow proper indentation and formatting conventions.

Testing:

- ☑ Implement unit tests to verify the functionality of individual components (Hash Table operations, AVL Tree operations, data loading, etc.).
- ☑ Conduct integration testing to ensure all components work together seamlessly.
- ☑ Test your program with various scenarios and edge cases (e.g., empty file, invalid data format, large datasets).

Additional Considerations:

- ☑ Explore error handling mechanisms to gracefully handle potential issues during file reading or data processing.

Submission:

- ☑ Ensure your code compiles and runs without errors.
- ☑ Submit all project files, including source code, documentation, and any test files you used.
- ☑ Follow the submission guidelines provided by your instructor.