Group Project: Data Exploration

Overview

In this assignment, you will work with real-world data to implement efficient query mechanisms using appropriate data structures. You will select a large dataset, design and implement an interface for querying this dataset, and thoroughly test your implementation. This project will allow you to apply data structure concepts to solve practical data retrieval problems.

You are allowed to use your favorite LLM when completing the assignment

Learning Objectives

By completing this assignment, you will:

- Gain experience working with large, real-world datasets
- Apply data structure concepts to solve practical problems
- Design and implement query interfaces for efficient data retrieval
- Develop skills in unit testing and verification
- Collaborate effectively in a small team environment

Assignment Details

Team Structure

- Work in teams of 3 students
- Form your team by 04/14

Requirements

1. Dataset Selection (Week 1 of the project)

Select a large dataset (minimum 10,000 records) from Kaggle, UCI Machine Learning Repository, or another approved source. The dataset should:

- Contain multiple attributes/columns (at least 5)
- Include a mix of numerical and categorical data (not mandatory)
- Be complex enough to demonstrate the value of efficient query mechanisms

2. Interface Design (Week 1)

Design a query interface for your dataset that includes:

- **A method for exact match queries** (e.g., find all records where attribute X equals value Y)
- **A method for range queries** (e.g., find all records where attribute X is between values Y and Z or find all records where attribute X is less than value Y)
- **A method for computing a statistic** (e.g., find the average/median/…/ value of attribute X)
- Use appropriate parameter types and return values

Your interface should have **three (3) methods** (one per member), perform exact match and range queries, and be defined as follows:

You will use appropriate parameter types and return values in your implementation.

```
public interface DatasetQuery {  // Give a meaningful name to your
ADT
      /**
      * Returns all records that exactly match the specified
criteria.
      * @param attribute The attribute/field to query on
      * @param value The exact value to match
      * @return A collection of records matching the criteria
      */
      List<T> exactMatchQuery(String attribute, Object value);


      /**
      * Returns all records where the specified attribute falls
within the given range.
      * @param attribute The attribute/field to query on
      * @param lowerBound The lower bound of the range (inclusive)
      * @param upperBound The upper bound of the range (inclusive)
      * @return A collection of records matching the criteria
      */
      List<T> rangeQuery(String attribute, Comparable lowerBound,
Comparable upperBound);
 // Or
List<T> rangeQuery(String attribute, Comparable upperBound);

      /**
      * Returns the average value of the specified attribute  during
a given time frame
       * @param attribute The attribute/field
      * @param startTime The start time of the period (inclusive)
      * @param endTime The end time of the period (inclusive)
      * @return The statistic calculated   */
      List<T> averageQuery(String attribute, timeStamp startTime,
timeStamp endTime);


}
```

3. Implementation (Weeks 2-3)

Implement the interface for your dataset using appropriate data structures. Your implementation should:

- Use the most efficient data structures
- Include appropriate documentation
- Consider time and space complexity in your implementation choices
- Handle edge cases (e.g., empty results, invalid queries)

4. Unit Testing (Weeks 3-4)

Develop comprehensive unit tests for your implementation that:

- Test both exact match and range query functionality
- Include edge cases and boundary values
- Verify the correctness of the results
- Measure and compare the performance of your methods

5. Analysis and Documentation (Week 4)

Provide a concise written report that includes:

- Justification for your data structure choices
- Analysis of time and space complexity
- Performance comparison between methods or all the implementations considered
- Discussion of limitations and potential improvements

Deliverables

1. Source code for interface and implementations
2. Unit tests and test results
3. Documentation

Scaffolding Examples

Example 1: Dataset Selection and Description

```
# Dataset Selection Report

## Dataset: NYC Yellow Taxi Trip Data (2019)

- Source:
https://www.kaggle.com/datasets/microize/newyork-yellow-taxi-trip-dat
a

- Size: 7.5 million records

- Attributes:

  - pickup_datetime (timestamp)

  - dropoff_datetime (timestamp)

  - passenger_count (integer)

  - trip_distance (float)

  - fare_amount (float)

  - payment_type (categorical)

  - pickup_location_id (integer)

  - dropoff_location_id (integer)

## Rationale for Selection

This dataset offers multiple attributes suitable for exact match
queries (e.g., payment_type) and range queries (e.g., trip_distance,
fare_amount). The large volume of data will allow us to demonstrate
the efficiency benefits of our data structure implementations.
```

Planned Query Types

1. Exact Match: Find all trips with a specific payment_type

2. Range Query: Find all trips with fare_amount between $X and $Y

3. Exact-match Query: Find the average trip distance in 2022, between 01/01/2022 and 12/31/2022

Example 2: Interface Definition & Implementation

```
/**
 * Interface for querying the NYC Yellow Taxi Trip dataset.
 * This interface provides methods for both exact match and range
 * queries on various attributes of taxi trip data.
 */
public interface NYCTaxiDataQuery {
       /**
     * Loads the NYC Taxi dataset from the specified file path.
     * @param filePath Path to the dataset file (CSV/JSON/XML format)
     * @return Number of records loaded
     * @throws IOException If there's an error reading the file
     */
    int loadDataset(String filePath) throws IOException;

    /**
     * Returns all taxi trips that exactly match the specified
     * criteria.
     * @param attribute The attribute/field to query on (e.g.,
     * "payment_type", "passenger_count")
     * @param value The exact value to match
     * @return A list of taxi trips matching the criteria
     * @throws IllegalArgumentException If the attribute is not
```

```
     * supported or the value type is incorrect
     */
    List<TaxiTrip> exactMatchQuery(String attribute, Object value);


    /**
     * Returns all taxi trips where the specified attribute falls
     * within the given range.
     *
     * @param attribute The attribute/field to query on (e.g.,
     * "fare_amount", "trip_distance")
     * @param lowerBound The lower bound of the range (inclusive)
     * @param upperBound The upper bound of the range (inclusive)
     * @return A list of taxi trips matching the criteria
     * @throws IllegalArgumentException If the attribute is not
     *  supported for range queries, or the bound types are incorrect
     */
    List<TaxiTrip> rangeQuery(String attribute, Comparable lowerBound,
Comparable upperBound);


    /**
     * Returns all taxi trips that occurred within the specified
     * date-time range.
     *
     * @param attribute The date-time attribute to query on
     * ("pickup_datetime" or "dropoff_datetime")
     * @param startDateTime The start date-time (inclusive) in format
     * "YYYY-MM-DD HH:MM:SS"
     * @param endDateTime The end date-time (inclusive) in format
     * "YYYY-MM-DD HH:MM:SS"
     * @return A list of taxi trips matching the criteria
     * @throws IllegalArgumentException If the attribute is not a
     * date-time attribute or the date-time strings are not in the
     * correct format
     */
```

```
    List<TaxiTrip> dateTimeRangeQuery(String attribute, String
startDateTime, String endDateTime);



    /**
     * Returns all taxi trips that occurred between the specified
     * pickup and dropoff locations.
     *
     * @param pickupLocationId The pickup location ID
     * @param dropoffLocationId The dropoff location ID
     * @return A list of taxi trips matching the criteria
     */
    List<TaxiTrip> locationQuery(int pickupLocationId, int
dropoffLocationId);


    /**
     * OPTIONAL Returns the number of records in the loaded dataset.
     *
     * @return The number of taxi trip records
     */
    int getRecordCount();

    /**
     * Returns statistics about the dataset, including:
     * - Average fare amount
     * - Average trip distance
     * - Average passenger count
     * - Number of trips by payment type
     *
     * @return A map of statistic names to their values
     */
    Map<String, Object> getDatasetStatistics();

}
```

Create a class to define your dataset (records) data type

```
/**
 * Class representing a NYC Yellow Taxi trip data type.
 */
public static class TaxiTrip {

    private String pickupDatetime;

    private String dropoffDatetime;

    private int passengerCount;

    private double tripDistance;

    private double fareAmount;

    private String paymentType;

    private int pickupLocationId;

    private int dropoffLocationId;

    private double tipAmount;

    private double totalAmount;


    // Constructor

    public TaxiTrip(String pickupDatetime, String dropoffDatetime,
                    int passengerCount, double tripDistance, double
                    fareAmount, String paymentType,
                    int pickupLocationId, int dropoffLocationId,
                    double tipAmount, double totalAmount) {
```

```
        this.pickupDatetime = pickupDatetime;
        this.dropoffDatetime = dropoffDatetime;
        this.passengerCount = passengerCount;
        this.tripDistance = tripDistance;
        this.fareAmount = fareAmount;
        this.paymentType = paymentType;
        this.pickupLocationId = pickupLocationId;
        this.dropoffLocationId = dropoffLocationId;
        this.tipAmount = tipAmount;
        this.totalAmount = totalAmount;
    }

    // Getters
    public String getPickupDatetime() { return pickupDatetime; }
    public String getDropoffDatetime() { return dropoffDatetime; }
    public int getPassengerCount() { return passengerCount; }
    public double getTripDistance() { return tripDistance; }
    public double getFareAmount() { return fareAmount; }
    public String getPaymentType() { return paymentType; }
    public int getPickupLocationId() { return pickupLocationId; }
    public int getDropoffLocationId() { return dropoffLocationId; }
    public double getTipAmount() { return tipAmount; }
    public double getTotalAmount() { return totalAmount; }

    @Override
```

```
        public String toString() {
            return "TaxiTrip{" +
                    "pickupDatetime='" + pickupDatetime + '\'' +
                    ", dropoffDatetime='" + dropoffDatetime + '\'' +
                    ", passengerCount=" + passengerCount +
                    ", tripDistance=" + tripDistance +
                    ", fareAmount=" + fareAmount +
                    ", paymentType='" + paymentType + '\'' +
                    ", pickupLocationId=" + pickupLocationId +
                    ", dropoffLocationId=" + dropoffLocationId +
                    ", tipAmount=" + tipAmount +
                    ", totalAmount=" + totalAmount +
                    '}';
        }
    }
```

```
public class NYCTaxiDataQueryImpl implements NYCTaxiDataQuery {

    //Implement the interface.  This is the main class of the project

}

}
```

Example 3: Unit Testing

```
public class NYCTaxiDataQueryImplTest {

    private DatasetQuery<TaxiTrip> queryEngine;

    private List<TaxiTrip> testDataset;


    @Before

    public void setUp() {

        // Create a test dataset

        testDataset = new ArrayList<>();

        // Add sample taxi trips to the test dataset

        testDataset.add(new TaxiTrip("2019-01-01 12:00:00", "2019-01-01
12:30:00", 1, 5.2, 15.5, "credit", 1, 2));

        testDataset.add(new TaxiTrip("2019-01-01 13:00:00", "2019-01-01
13:45:00", 2, 7.8, 25.0, "cash", 3, 4));

        testDataset.add(new TaxiTrip("2019-01-01 14:00:00", "2019-01-01
14:20:00", 1, 3.5, 12.5, "credit", 5, 6));

        // Add more test data...


         // Initialize the query engine with the test dataset

        queryEngine = new NYCTaxiDataQueryImpl(testDataset);

    }
```

```
    @Test
    public void testExactMatchQuery() {
        // Test exact match for payment_type = "credit"
        List<TaxiTrip> results =
                    queryEngine.exactMatchQuery("payment_type", "credit");
        // Assert that the correct number of results is returned
        assertEquals(2, results.size());
        // Assert that all results have the expected payment_type
        for (TaxiTrip trip : results) {
            assertEquals("credit", trip.getPaymentType());
        }
    }

    @Test
    public void testRangeQuery() {
        // Test range query for 10.0 <= fare_amount <= 20.0
        List<TaxiTrip> results = queryEngine.rangeQuery("fare_amount",
10.0, 20.0);
        // Assert that the correct number of results is returned
        assertEquals(2, results.size());
        // Assert that all results are within the expected range
        for (TaxiTrip trip : results) {
            double fareAmount = trip.getFareAmount();
```

```
            assertTrue(fareAmount >= 10.0 && fareAmount <= 20.0);
        }
    }

    @Test
    public void testExactMatchQueryWithNoResults() {
        // Test exact match for payment_type = "unknown"
        List<TaxiTrip> results =
queryEngine.exactMatchQuery("payment_type", "unknown");
        // Assert that no results are returned
        assertTrue(results.isEmpty());
    }

    @Test(expected = IllegalArgumentException.class)
    public void testExactMatchQueryWithUnsupportedAttribute() {
        // Test exact match for an unsupported attribute
        queryEngine.exactMatchQuery("unsupported_attribute", "value");
    }

    // Additional tests...
}
```

Timeline and Milestones

Week 1

- Days 1-2: Form teams and select a dataset
- Days 3-4: Design the query interface and data model
- Days 5-7: Begin implementation
- **Checkpoint #1: 04/16**

Week 2

- Days 8-12: Work on implementation
- Days 13-14: Begin writing unit tests
- **Checkpoint #2: 04/23**

Week 3/4

- Days 15-19: Complete implementation, unit tests, and fix any identified issues
- Days 20-23: Conduct performance analysis and finalize documentation
- **May 1 @11:59pm: Project submission**

Grading Criteria

Component	Weight	Description
Dataset Selection	10%	Appropriateness and complexity of the chosen dataset
Interface Design	25%	Quality, clarity, and completeness of the interface
Implementation	40%	Correctness, efficiency, and documentation of code
Analysis Report	15%	Depth of analysis and quality of documentation

Teamwork	10%	Equal contribution and effective collaboration
Unit Testing (extra credit)		Comprehensiveness and quality of tests

Resources

- Kaggle Datasets
- UCI Machine Learning Repository
- JUnit Documentation
- Java Collections Framework Tutorial
- Big-O Cheat Sheet

Submission Instructions

Gradescope

Questions and Support

If you have questions about this assignment, please:

1. Post in Ed discussion
2. Attend office hours
3. Do not stay stuck, seek help