Projects: Implementation and Experimental Evaluation of Multidimensional Data Structures

Professors: S. Sioutas, G. Vonitsanos (Postdoc Researcher@CEID)

Goal: The major task is the implementation and experimental evaluation of a variety of multi-dimensional data structures in a programming language of your preference (we suggest Python, C++ or Java). Use real datasets to evaluate Build, Insert, Delete, Update and Searching (range, similarity, kNN) queries. You can download the real datasets from the following URLs:

Amazon Fine Food Reviews (CSV): https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

NYC Taxi Trip Duration: https://www.kaggle.com/competitions/nyc-taxi-trip-duration/data

Project-1: Multi-dimensional Data Indexing and Similarity Query Processing: Develop multidimensional access methods based on k-d trees, quad trees, Range Trees and R-trees to support k-dimensional queries, where $k \leq 4$ (indexing at most four numeric attributes). After the first indexing phase, perform textual similarity queries over review text using MinHash and Locality-Sensitive Hashing (LSH). Compare the four hybrid schemes: k-d Tree + LSH, 4D Quad Tree + LSH, Range Tree + LSH, and R-Tree + LSH. A 4D Quad Tree requires a 16-way subdivision (or a clearly documented equivalent generalization).

Use-Case: Amazon Fine Food Reviews Dataset (CSV)

https://www.kaggle.com/datasets/snap/amazon-fine-food-reviews

The Amazon Fine Food Reviews dataset contains approximately 568,454 product reviews. The Reviews.csv file includes product and reviewer identifiers, rating scores, timestamps, helpfulness votes, short summaries, and full review text. It supports both numerical multidimensional filtering and textual similarity search. The four indexing dimensions are: d1 = Score, d2 = Time, d3 = HelpfulnessNumerator, and d4 = HelpfulnessDenominator. The Text attribute is used separately for MinHash/LSH.

Dataset Columns Description

Column	Description
Id	Unique review identifier
ProductId	Unique product identifier
UserId	Reviewer identifier
ProfileName	Reviewer profile name
HelpfulnessNumerator	Number of users who found the review helpful (dimension d3)
HelpfulnessDenominator	Number of users who voted on review helpfulness (dimension d4)
Score	Product rating, from 1 to 5 (dimension d1)
Time	Review timestamp, Unix time (dimension d2)
Summary	Short review summary
Text	Full review text used for MinHash/LSH

Dataset file: Reviews.csv (CSV format). Use Id as the unique review key. Convert Time (Unix timestamp) to a consistent date/time representation for queries. Check missing values and duplicate records. For textual

similarity, preprocess Text and build shingles/MinHash signatures; define the similarity metric and LSH parameters explicitly.

Example: Consider the four-dimensional review points (Score, Time, HelpfulnessNumerator, HelpfulnessDenominator). Find the top N most similar reviews (according to their Text attribute) published between 2008 and 2012, having Score between 4 and 5, HelpfulnessNumerator ≥ 10, and HelpfulnessDenominator ≤ 30. Parameter N is user-defined (e.g., N = 5). First apply the multidimensional filters, then rank the eligible reviews using textual similarity.

Project-2: Develop the following geometric data structures. We suggest the following real dataset:

New York City Taxi Trip Duration (Kaggle): https://www.kaggle.com/competitions/nyc-taxi-trip-duration/data

The dataset is a cleaned sample of 2016 NYC Yellow Cab trip records, originally published by the NYC Taxi and Limousine Commission. The training file contains 1,458,644 trips and the test file contains 625,134 trips. Use train.csv for tasks requiring trip_duration or dropoff_datetime, since the test file does not provide all target fields.

Dataset Columns Description

Column	Description
id	Unique trip identifier
vendor_id	Provider code
pickup_datetime	Pickup timestamp
dropoff_datetime	Dropoff timestamp (training data)
passenger_count	Number of passengers
pickup_longitude	Longitude of pickup point
pickup_latitude	Latitude of pickup point
dropoff_longitude	Longitude of dropoff point
dropoff_latitude	Latitude of dropoff point
store_and_fwd_flag	Store-and-forward flag
trip_duration	Trip duration in seconds (training data)

1. 3D R-trees for Spatio-Temporal Query Processing:

Implement 3D R-trees to index pickup events as (longitude, latitude, pickup_time), optionally indexing dropoff events separately. Evaluate 3D orthogonal range queries, e.g. “Find the number of pickups inside a specified Manhattan rectangle between 12:00 and 14:00 on a given day.” The dataset does NOT contain intermediate GPS trajectory points: do not claim that a taxi passed through a region between pickup and dropoff unless additional trajectory data is used.

2. Interval Trees and Segment Trees:

Represent each trip as the time interval [pickup_datetime, dropoff_datetime]. Implement Interval Trees for interval (overlap) queries and Segment Trees for a clearly specified temporal aggregation / stabbing-query workload. Compare build time, memory, and query time; relate empirical trends to theoretical complexity.

Multi-Dimensional Data Structures: Choose one of the following two (2) projects

3. Convex Hull and Skyline Operator:

Consider the two-dimensional point set P1 = {(trip_duration, passenger_count)}. Compute CH(P1). For a meaningful preference query, construct P2={(trip_duration, trip_distance_proxy)}, where trip_distance_proxy is the great-circle pickup-to-dropoff distance computed from coordinates, and find the Skyline for MIN trip_duration and MAX trip_distance_proxy. *Explicitly distinguish the derived straight-line distance from the actual driven distance. Develop the SKYLINE OPERATOR that is implemented as a PREDICATE in SQL as follows:*

```
SELECT ... FROM ... WHERE ...

GROUP BY ... HAVING ...

SKYLINE OF [DISTINCT] d1 [MIN | MAX | DIFF],……, dm [MIN | MAX | DIFF]

ORDER BY ...
```

Evaluate the time and space performance of your proposed methods and prove experimentally that the response time follows the theoretical complexities.

4. Line Segment Intersection:

Model each taxi trip as a straight line segment connecting its pickup and dropoff coordinates in a projected planar coordinate system. Develop a sweep-line intersection algorithm and evaluate time/space performance. These segments are geometric approximations, NOT recorded road trajectories. Compare against a brute-force baseline on manageable subsets.

Evaluate the time and space performance of the proposed methods and investigate whether the observed scaling is consistent with their theoretical complexities. Include appropriate correctness checks and clearly document preprocessing assumptions.

Background Knowledge: Data Structures, Algorithms and Complexity, Databases, Object-Oriented Programming (C++, Java), Functional Programming (Python, Scala).

References:

A. K. Tsakalidis, Advanced Data Structures

https://en.wikipedia.org/wiki/Range_tree

https://en.wikipedia.org/wiki/K-d_tree

https://en.wikipedia.org/wiki/Quadtree

https://en.wikipedia.org/wiki/Interval_tree

https://en.wikipedia.org/wiki/Segment_tree

https://en.wikipedia.org/wiki/MinHash

Multi-Dimensional Data Structures: Choose one of the following two (2) projects

https://en.wikipedia.org/wiki/Locality-sensitive_hashing

https://en.wikipedia.org/wiki/R-tree

https://en.wikipedia.org/wiki/Convex_hull

https://en.wikipedia.org/wiki/Sweep_line_algorithm

https://en.wikipedia.org/wiki/Line_segment_intersection

(***) Deliverables: ZIP or RAR file containing executable source code, dataset preparation scripts, a README with reproduction instructions, and a brief report with experimental plots, correctness validation, and complexity discussion. Deadline: To be announced.