PROJECT REPORT

OPTIMIZED MULTI ATTRIBUTE AVL TREE ADAPTER (O-M.A.A.T.A)

COURSE INSTRUCTOR

Sir Shoaib Rauf

SUBMITTED BY

Shahzaib Khan (19K-0273)

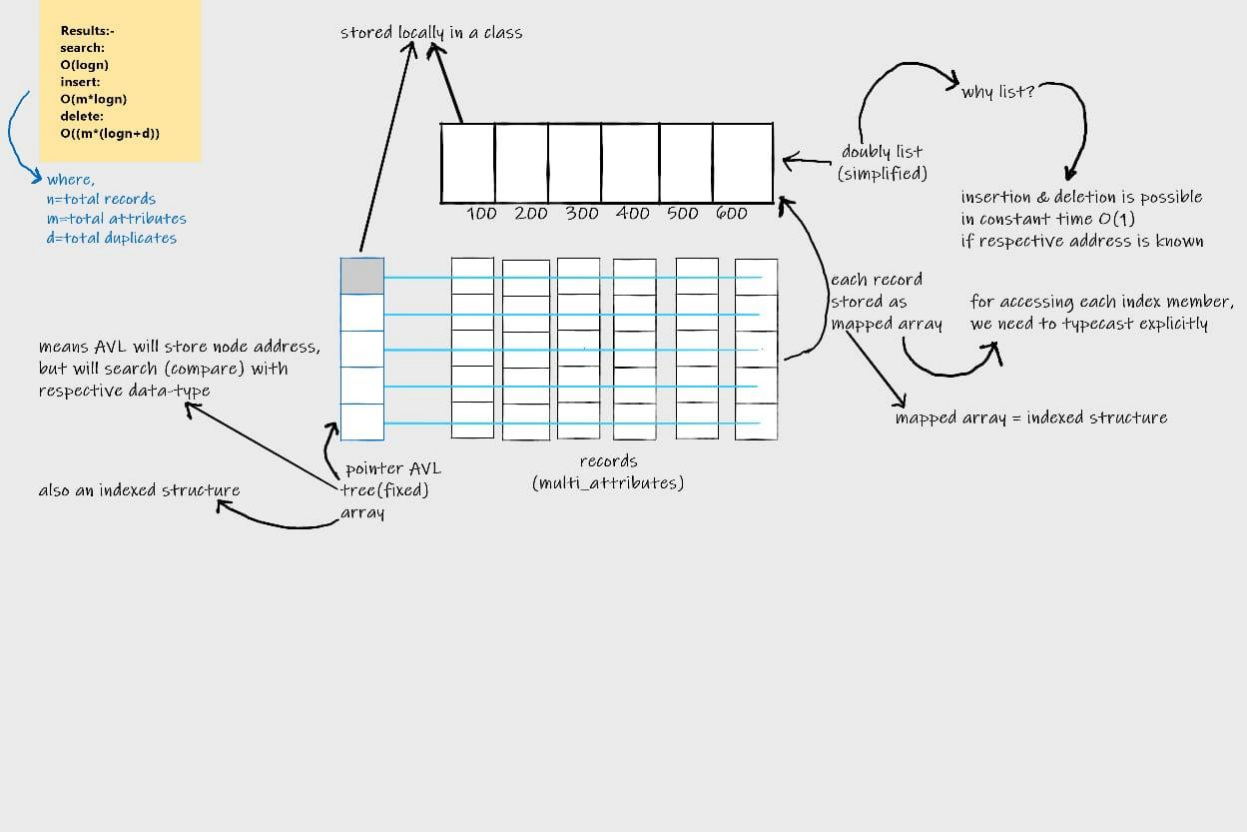
Introduction:

Our class objects can be treated as SQL Like Tables since they can be used for optimized insertion, deletion and searching with O(log n) time complexity. It can basically resolve most SQL queries on given data of almost any format on our demand. It can easily read data from excel sheets or any other files and can manipulate them very easily and efficiently.

Problem Statement:

In single (key) attributed trees (BST/AVL etc), we usually point data of one type at a time (i.e., we can use comparator for only one attribute), but the problem occurs when we need trees for storing structure of multiple attributes & require optimize results for all attributes. Means, for AVL, it can then only give optimized results for one attribute i.e. O(log n) while for rest its O(N) time-complexity.

Simplified Concept Diagram:



Applications:

1) Useful for data manipulation in .csv (a.k.a. Excel) files

2) Useful in Data Analytics for quick summary

Main Features:

1) Can be used as generic SQL like table

2) Can resolve most of SQL like queries, without join relations

3) Can handle duplicates very efficiently

4) Quick Indexing for row-wise record access for data manipulation

5) Can be easily modified w.r.t user requirements

6) Provides filter search, update & delete operations

7) Queries can also utilize Range filter in C.R.U.D operations.

8) each key can also have different comparator for filter.
(e.g., equal, not_equal, greater, lesser, is_in_range, isnt_in_range, etc)

Concepts & Base structures used:

1) Mapped object array (as record / row)

2) Singly linked list (as duplicate node list)

3) Doubly linked list (as main data handler & holder)

3) Node stacks & queues (in Garbage Collector version, and list operations)

4) Array stack and queues (as dynamic circular-array deque)

5) AVL trees (as attribute address book keeper)

Additional Indirect Learning Concepts:

1) Iterators

2) Move semantics

3) Rule of 5

4) Move Initializer List

5) Template Meta Programming

6) Object Oriented & Functional Programming

7) Vector concept understanding with P.O.Ds and Non-P.O.Ds

Future Work:

1) Conceptually expansible (i.e. making it possible for almost any future work)

2) Possible to use lambda functions as comparators for explicit filtering (functional programming)

3) Growth factor for indexer can be easily optimized by using segmented deque (combination of segmented vector (2D dynamic array) with deque (dynamic circular-array double-ended queue)

Note: after handling join relations with connector tables (acting as SQL junction table), our class objects will almost behave like SQL tables.

Tools & Technologies:

1) Dev C++ compiler

2) C++11 programming language

3) Operating System: Microsoft Windows 10/11