**Assignment 1**

**Weather Data Storage System**

Department: CSE  
Session: 2025-26  
Programme: BTech   
Semester: 3rd  
Course Code: ENCS205   
Course: Data Structures  
Assignment No: 01

**Submitted by: Sonam Sangwan**

**Submitted to: Ms. Swati**

# 1. Introduction

This assignment focuses on the implementation of a Weather Data Storage System using Abstract Data Types (ADTs) and 2D arrays in Java. The system is designed to store, retrieve, and manage weather-related data such as temperature for multiple cities over multiple years. It also demonstrates handling of sparse datasets, and comparison of row-major and column-major access.

# 2. Problem Statement

The task is to develop a software solution in Java that organizes and retrieves temperature data for different cities and years using structured data formats like 2D arrays and ADTs.

# 3. Weather Record ADT Design

Attributes:

a. Date: Custom class (day/month/year)  
b. City: String (name of the city)  
c. Temperature: double (temperature value)

Methods:

• insert(year, city, temp): Insert a new weather record  
• remove(year, city): Remove a weather record by city and year  
• retrieve(year, city): Retrieve temperature data for a specific city and year  
• rowMajorAccess(): Print all records row by row (year-wise)  
• columnMajorAccess(): Print all records column by column (city-wise)

# 4. Implementation (Java Code)

The following Java program demonstrates the implementation of the Weather Data Storage System with 2D arrays, row-major and column-major access, and sparse data handling.

import java.util.\*;  
  
class DateRecord {  
 int day, month, year;  
 DateRecord(int d, int m, int y) {  
 day = d; month = m; year = y;  
 }  
}  
  
class WeatherRecord {  
 DateRecord date;  
 String city;  
 double temperature;  
 WeatherRecord(int d, int m, int y, String c, double t) {  
 date = new DateRecord(d, m, y);  
 city = c;  
 temperature = t;  
 }  
}  
  
class WeatherDataSystem {  
 private List<String> cities;  
 private List<Integer> years;  
 private double[][] tempData;  
 private final double SENTINEL = -9999;  
  
 WeatherDataSystem(List<Integer> yrs, List<String> cts) {  
 years = yrs; cities = cts;  
 tempData = new double[years.size()][cities.size()];  
 for (int i = 0; i < years.size(); i++) {  
 Arrays.fill(tempData[i], SENTINEL);  
 }  
 }  
  
 void insert(int year, String city, double temp) {  
 int r = years.indexOf(year);  
 int c = cities.indexOf(city);  
 if (r != -1 && c != -1) tempData[r][c] = temp;  
 }  
  
 void remove(int year, String city) {  
 int r = years.indexOf(year);  
 int c = cities.indexOf(city);  
 if (r != -1 && c != -1) tempData[r][c] = SENTINEL;  
 }  
  
 double retrieve(int year, String city) {  
 int r = years.indexOf(year);  
 int c = cities.indexOf(city);  
 if (r != -1 && c != -1) return tempData[r][c];  
 return SENTINEL;  
 }  
  
 void rowMajorAccess() {  
 for (int i = 0; i < years.size(); i++) {  
 for (int j = 0; j < cities.size(); j++) {  
 System.out.println(years.get(i) + " - " + cities.get(j) + ": " + tempData[i][j]);  
 }  
 }  
 }  
  
 void columnMajorAccess() {  
 for (int j = 0; j < cities.size(); j++) {  
 for (int i = 0; i < years.size(); i++) {  
 System.out.println(years.get(i) + " - " + cities.get(j) + ": " + tempData[i][j]);  
 }  
 }  
 }  
}  
  
public class Weather\_Data\_Storage {  
 public static void main(String[] args) {  
 List<Integer> years = Arrays.asList(2023, 2024, 2025);  
 List<String> cities = Arrays.asList("Delhi", "Mumbai", "Chennai");  
  
 WeatherDataSystem wds = new WeatherDataSystem(years, cities);  
 wds.insert(2023, "Delhi", 32.5);  
 wds.insert(2024, "Mumbai", 29.0);  
  
 System.out.println("Delhi 2023 Temp: " + wds.retrieve(2023, "Delhi"));  
  
 System.out.println("\nRow Major Access:");  
 wds.rowMajorAccess();  
  
 System.out.println("\nColumn Major Access:");  
 wds.columnMajorAccess();  
 }  
}

# 5. Row-Major vs Column-Major Access

Row-Major Access: Data is accessed row by row (efficient in Java as memory is stored row-wise).  
Column-Major Access: Data is accessed column by column (less cache-efficient in row-major systems).

# 6. Sparse Data Handling

Sparse datasets are handled using SENTINEL values (-9999). Alternative methods include maps or linked structures to reduce space when most data is missing.

# 7. Complexity Analysis

Insert/Delete/Retrieve: O(1)  
Space Complexity: O(n × m) for full 2D array, optimized with sparse representation.

# 8. Output Screenshots

Sample Output:  
  
Delhi 2023 Temp: 32.5  
  
Row Major Access:  
2023 - Delhi: 32.5  
2023 - Mumbai: -9999.0  
2023 - Chennai: -9999.0  
2024 - Delhi: -9999.0  
2024 - Mumbai: 29.0  
2024 - Chennai: -9999.0  
2025 - Delhi: -9999.0  
2025 - Mumbai: -9999.0  
2025 - Chennai: -9999.0  
  
Column Major Access:  
2023 - Delhi: 32.5  
2024 - Delhi: -9999.0  
2025 - Delhi: -9999.0  
2023 - Mumbai: -9999.0  
2024 - Mumbai: 29.0  
2025 - Mumbai: -9999.0  
2023 - Chennai: -9999.0  
2024 - Chennai: -9999.0  
2025 - Chennai: -9999.0

# 9. Conclusion

This assignment demonstrates the design and implementation of a Weather Data Storage System using ADTs and 2D arrays in Java. It highlights efficient data retrieval, comparison of memory access patterns, and sparse data handling techniques, fulfilling the objectives of the assignment.