K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Assignment 1
Weather Data Storage System

Department: CSE(DS)

Session: 2025-26

Programme: BTech

Semester: 3rd

Course Code: ENCS205

Course: Data Structures

Assignment No: 01

Submitted by: Shashwat Tripathi

Roll no. 2401420025

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 Python. 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 Python 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: String (day/month/year)
b. City: String (name of the city)
c. Temperature: float (temperature value)

Methods:
- insert(year, city, temp): Insert a new weather record
- delete(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 (Python Code)

```
SENTINEL = -9999.0

class WeatherRecord:
  def __init__(self, date, city, temperature):
    self.date = date
    self.city = city
    self.temperature = temperature

class WeatherDataStorage:
  def __init__(self, years, cities):
    self.years = years
```

```
        self.cities = cities
        self.temperature_data = [[SENTINEL for _ in range(len(cities))] for _ in
range(len(years))]

    def insert(self, city, year, temperature):
        if city in self.cities and year in self.years:
            r = self.years.index(year)
            c = self.cities.index(city)
            self.temperature_data[r][c] = temperature

    def delete(self, city, year):
        if city in self.cities and year in self.years:
            r = self.years.index(year)
            c = self.cities.index(city)
            self.temperature_data[r][c] = SENTINEL

    def retrieve(self, city, year):
        if city in self.cities and year in self.years:
            r = self.years.index(year)
            c = self.cities.index(city)
            return self.temperature_data[r][c]
        return SENTINEL

    def row_major_access(self):
        for i in range(len(self.years)):
            for j in range(len(self.cities)):
                print(self.years[i], self.cities[j], self.temperature_data[i][j])

    def column_major_access(self):
        for j in range(len(self.cities)):
            for i in range(len(self.years)):
                print(self.years[i], self.cities[j], self.temperature_data[i][j])
```

5. Row-Major vs Column-Major Access

Row-Major Access: Data is accessed row by row (efficient in Python as lists are 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 dictionaries or linked structures to reduce space when most data is missing.

7. Complexity Analysis

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

8. Sample Outputs

```
Retrieve Delhi 2023: 32.5

Row Major Access:
2023 - Delhi: 32.5
2023 - Mumbai: 29.0
2023 - Chennai: -9999.0
2024 - Delhi: -9999.0
2024 - Mumbai: -9999.0
2024 - Chennai: 34.0
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: 29.0
2024 - Mumbai: -9999.0
2025 - Mumbai: -9999.0
2023 - Chennai: -9999.0
2024 - Chennai: 34.0
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 Python. It highlights efficient data retrieval, comparison of memory access patterns, and sparse data handling techniques, fulfilling the objectives of the assignment.