Course: Data Structure

(Course Code : ENCS205)

UNIT-1: Foundations of Data Structures

School of Engineering & Technology
K.R. Mangalam University, Gurugram (Haryana)

SESSION 13:

Introduction to Sparse Matrices

Session 4: Learning Objectives

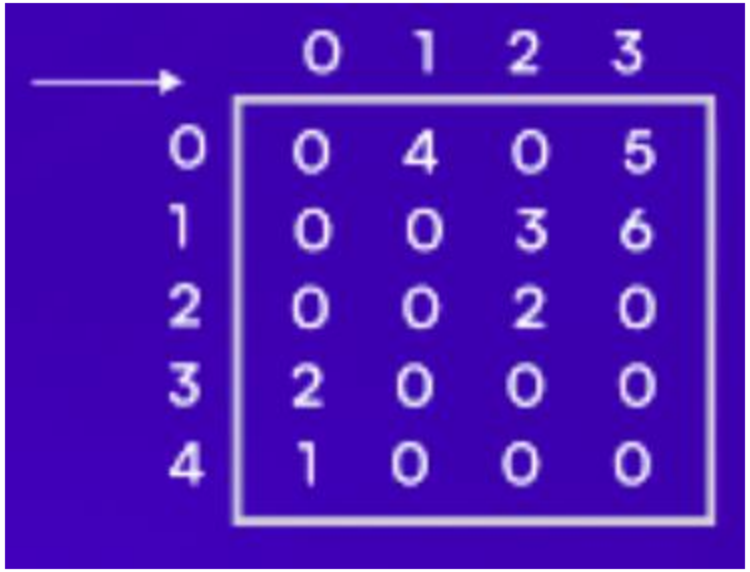
By the end of this session, will be able to:

- Understand what a sparse matrix is and why it is used
- Identify different types of sparse matrices (lower, upper, tri-diagonal)
- Explain the need for efficient representation of sparse matrices
- Describe and compare array-based and linked list-based representations
- Analyze and Compare the time and space complexities of each method.

What Is a Sparse Matrix?

- A matrix where **most elements are zero.**

- Common in large-scale, real-world datasets—e.g., in machine learning, data science, and graph theory



Why Use Special Representations?

- **Problems with Normal Storage:**
  - Wastes memory (stores zeros unnecessarily)
  - More processing time for traversals/operations
- **Advantages of Sparse Storage:**
  - Saves memory
  - Reduces time complexity for operations involving only non-zero elements
- **Example:**
  - A 1000×1000 matrix = **1 million elements**
  - If only 1000 are non-zero, normal storage wastes **99.9% memory**

Types of Sparse Matrices

- **Lower Triangular**: all entries above the main diagonal are zero
- **Upper Triangular**: all entries below the main diagonal are zero
- **Tri-diagonal**: non-zero values only on main, sub-, and super-diagonals

Lower Triangular Matrix

Representation of Lower Triangular Matrix A[5,5]

$$
\begin{bmatrix}
A_{11} & 0 & 0 & 0 & 0 \\
A_{21} & A_{22} & 0 & 0 & 0 \\
A_{31} & A_{32} & A_{33} & 0 & 0 \\
A_{41} & A_{42} & A_{43} & A_{44} & 0 \\
A_{51} \longrightarrow & A_{52} \longrightarrow & A_{53} \longrightarrow & A_{54} \longrightarrow & A_{55}
\end{bmatrix}
$$

Upper Triangular Matrix

Representation of Upper Triangular Matrix
A[5,5]

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} \\ 0 & A_{22} & A_{23} & A_{24} & A_{25} \\ 0 & 0 & A_{33} & A_{34} & A_{35} \\ 0 & 0 & 0 & A_{44} & A_{45} \\ 0 & 0 & 0 & 0 & A_{55} \end{bmatrix}$$

Tri-Diagonal Sparse Matrix

Representation of Tri-Diagonal Sparse Matrix
A[5,5]

$$
\begin{bmatrix}
A_{11} & A_{11} & 0 & 0 & 0 \\
A_{21} & A_{22} & A_{22} & 0 & 0 \\
0 & A_{32} & A_{33} & A_{34} & 0 \\
0 & 0 & A_{43} & A_{44} & A_{45} \\
0 & 0 & 0 & A_{54} & A_{55}
\end{bmatrix}
$$

Representation Methods

1. **Using Array (Compact Storage)**
   - Example C implementation stores non-zero entries as a compact 3×N array (row, col, value)
2. **Using Linked List**
   - Use a list of nodes; each node stores value, row index, column index, and a pointer to the next node

Array (Triplet) Representation

- **Idea:** Store only non-zero elements and their positions
- **Format:** Three arrays or a 2D array with 3 columns
  - Row index
  - Column index
  - Value

- **Example:**

Original Matrix (5×4):

Sparse Matrix →

$$
\begin{array}{c|cccc|}
 & 0 & 1 & 2 & 3 \\
\hline
0 & 0 & 4 & 0 & 5 \\
1 & 0 & 0 & 3 & 6 \\
2 & 0 & 0 & 2 & 0 \\
3 & 2 & 0 & 0 & 0 \\
4 & 1 & 0 & 0 & 0 \\
\hline
\end{array}
$$

Table Structure

Row	Column	Value
0	1	4
0	3	5
1	2	3
1	3	6
2	2	2
3	0	2
4	0	1
5	4	7

Array Representation — Implementation

Pseudocode:

```
function sparse_matrix_to_triplet(matrix, rows, cols):
1.      triplet = []
2.      for i in range(rows):
3.          for j in range(cols):
4.              if matrix[i][j] != 0:
5.                  triplet.append([i, j, matrix[i][j]])
6.      return triplet
```

- **Space Complexity:** O(k) where k = number of non-zero elements
- **Time Complexity:** O(N × M) for conversion

Linked List Representation

- **Idea:** Store non-zero elements as nodes in a linked list
- **Advantages over array:**
  - Dynamic size
  - Easier to insert/delete non-zero elements
- **Example:**
- The same matrix as before becomes:

  (0,1,4) → (0,3,5) → (1,2,3) → (1,3,6) → (2,2,2) → (3,0,6) → (4,0,1)

Linked List — Implementation

Pseudocode:

```
function sparse_matrix_to_linked_list(matrix, rows,cols):
    head = None
    for i in range(rows):
        for j in range(cols):
            if matrix[i][j] != 0:
                newNode = Node(i, j, matrix[i][j])
                insert_at_end(head, newNode)
    return head
```

Complexity:

- **Space:** O(k)
- **Conversion Time:** O(N × M)

Complexity Comparison

Representation	Time Complexity (Conversion)	Space Complexity	Pros	Cons
Array	$O(N \times M)$	$O(k)$	Simple, fast lookup	Fixed size, harder insert/delete
Linked List	$O(N \times M)$	$O(k)$ + pointer mem	Dynamic, easy modifications	Slower lookup

Summary of Session 14

- Sparse matrices save space and time when dealing with mostly zero elements.
- **Array Representation:** Good for quick access, compact storage.
- **Linked List Representation:** Flexible, better for frequent updates.
- Choosing the right representation depends on **access pattern** and **update frequency**.

Exercise

1. You have a 1000×1000 matrix with only 1500 non-zero elements.
   a. Calculate the space complexity for storing it in normal 2D array form and in array (triplet) representation, assuming each integer takes 4 bytes.
   b. Which saves more memory and by how much?

2. Consider the following triplet representation:

```
Row | Col | Value
-----------------
0   | 2   | 5
0   | 3   | 8
1   | 0   | 3
2   | 1   | 6
```

Reconstruct the 3×4 matrix from this triplet form.

Q3. For a matrix with n*n rows, m*m columns, and k*k non-zero elements:

a. What is the time complexity of converting it to array (triplet) form?
b. What is the space complexity of the triplet form?

Q4. In linked list representation of a sparse matrix:

a. If you need to find the value at position (i, j), what is the worst-case time complexity? Why?
b. Suggest one modification to improve lookup time.

Answer Key

Q1: Normal: 1000×1000×4=4,000,000 bytes (≈ 4 MB)

Triplet: 1500×3×4=18,000 bytes (≈ 18 KB)

Saved: ≈ 3.982 MB

Q2:

```
0 0 5 8
3 0 0 0
0 6 0 0
```

Q3: a) O(n×m) b) O(k)

Q4: Worst case: O(k) — must traverse the list.

Improvement: Use a hash map with (row, col) as key → O(1) lookup.

Thank You

K.R. MANGALAM UNIVERSITY

THE COMPLETE WORLD OF EDUCATION

Recognised under the section 2 (f) of the UGC Act 1956

Empowering the Youth; Empowering the Nation