Sparse Matrices

Sparse matrices are matrices in which a significant number of elements are zero or have some default value. These matrices are common in various scientific and engineering applications, such as finite element analysis, network analysis, and image processing, where the data is inherently sparse. Storing sparse matrices efficiently is essential to save memory and improve computational performance. In C++, there are several ways to represent sparse matrices:

1. Compressed Sparse Row (CSR) Format:

In the CSR format, you store three arrays:

- **data**: An array containing non-zero values.
- **indices**: An array containing the column indices of the corresponding values.
- **Index pointers**: An array that points to the beginning of each row in the **data** and **indices** arrays.

This format is efficient for matrix-vector multiplication operations and is widely used in numerical computing libraries like Intel Math Kernel Library (MKL) and SciPy.

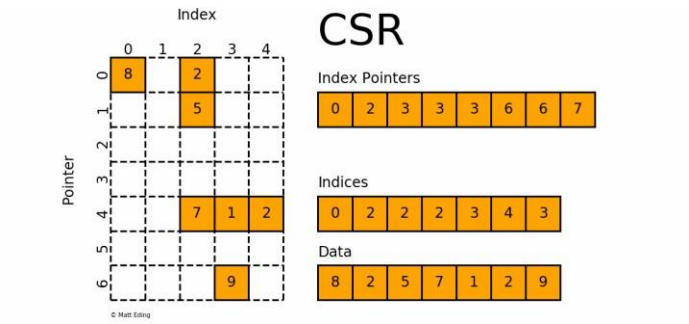


Fig.1 (Compressed Sparse Row)

2. Compressed Sparse Column (CSC) Format:

Similar to CSR, the CSC format stores non-zero values and column indices, but it uses a different indexing scheme. It is often more efficient for some operations like backslash (\) in MATLAB.

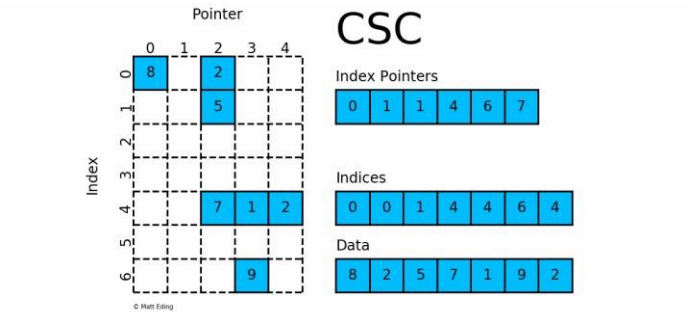


Fig.2 (Compressed Sparse Column)

3. List of Lists (LIL) Format:

In this format, you use a list of lists to represent the sparse matrix. Each list corresponds to a row and contains the non-zero elements along with their column indices. This format is straightforward but not very memory-efficient.

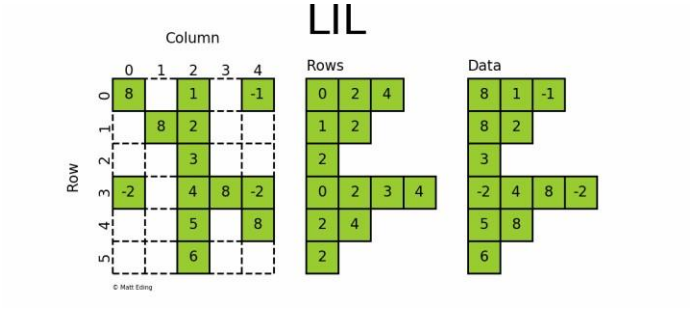


Fig.3 (List of List)

4. Coordinate List (COO) Format:

The COO format stores the (row, column, value) triplets of non-zero elements in a list. This format is simple but can be inefficient for certain operations.

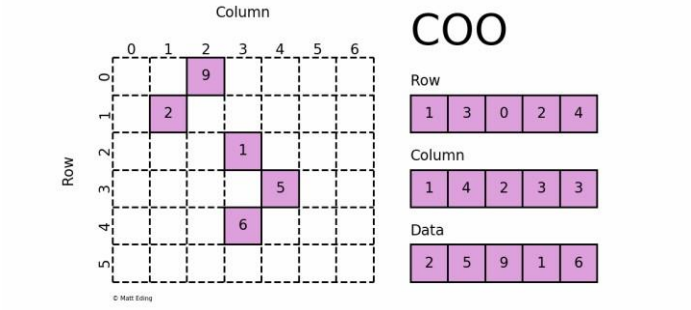


Fig.4 (Coordinate List)

5. Triangular Matrix Formats (CSR, CSC, etc.):

If your sparse matrix is triangular (upper or lower), you can use the CSR or CSC format and ignore the zero elements below or above the main diagonal. In a Lower triangular sparse matrix, all elements above the main diagonal have a zero value. This type of sparse matrix is also known as a lower triangular matrix. In a lower triangular matrix, Array [i],[j] = 0 where i<j.

Similarly, in an upper triangular matrix, Array [i],[j] = 0 where i>j. Similarly, a tri-diagonal matrix is also another type of a sparse matrix, where elements with a non-zero value appear only on the diagonal or immediately below or above the diagonal. In a tridiagonal matrix, Array [i][j] = 0, where |i – j| > 1.

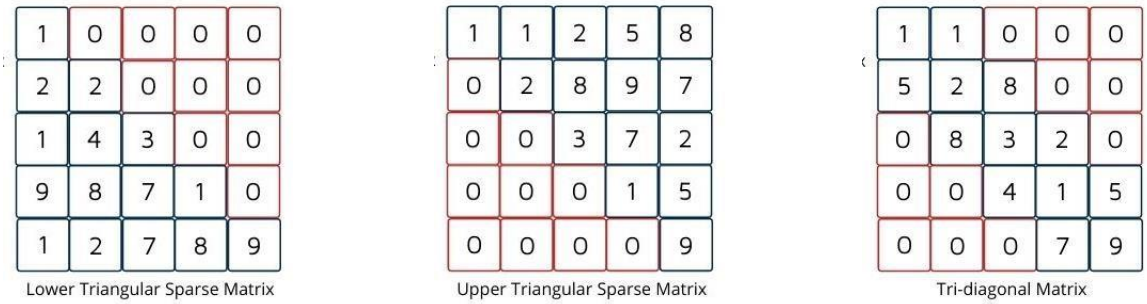


Fig. 5 (Triangular Matrix)

6. Dictionary of Keys (DOK) Format:

DOK format uses a dictionary (associative container) to store (row, column) keys and their corresponding values. This format is useful for constructing a sparse matrix incrementally but may not be as efficient for matrix operations.

7. Eigen Sparse Matrix:

If you're using the Eigen library in C++, it provides a built-in Sparse Matrix class that can efficiently represent and operate on sparse matrices. It supports various storage schemes, including compressed formats.

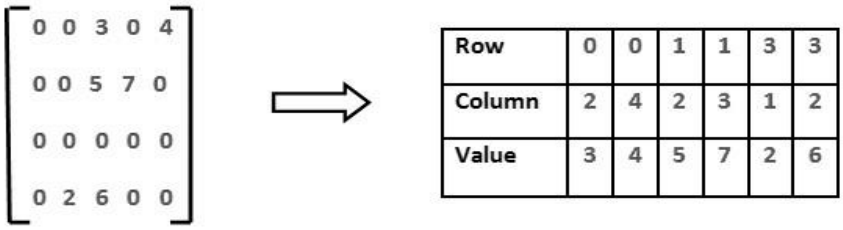
How to Select a Sparse Matrix Representation:

Choosing the right representation depends on your specific application and the operations you need to perform on the sparse matrix. The choice of format can significantly impact memory usage and computational performance. You may also need to consider libraries or frameworks like Eigen or Boost uBLAS that provide efficient implementations of sparse matrix operations.

In this Lab we will be using 2-D array-based representation of Sparse Matrix.

Each element of the matrix is uniquely characterized by its row and column positions. So a triple (i, j, value) can easily represent the non-zero elements of the matrix. In the sparse representation of a matrix, there are three columns. In the first row, we always specify the number of rows, columns, and non-zero elements (No_Of_Non ZeroValues) in columns 1, 2, and 3, respectively. From the second row onwards, we store each non-zero element by its triple (i, j, value).

So in a sparse matrix, there are three columns and (No_Of_NonZeroValues + 1) rows. In general, for space reliability, 3 × (No_Of_NonZeroValues + 1) should always be less than or equal to m x n where m = number of rows and n = number of columns. No_Of_NonZeroValues = Number of non-zero elements. In brief, for the alternate representation, we should have 3 × (No_Of_NonZeroValues + 1) <= m ×n



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

Fig.6 (Sparse Matrices) In left picture the Matrix is normal, in the right figure the matrix is sparse.

Task 01: Sparse Matrix

Create a 2D matrix class with well-defined data members and member functions to represent the matrix. You have been provided with a template for two classes, **which you can adjust as needed.** Consider how to track the count of non-zero elements in the matrix and contemplate the concept of composition. Should the Sparse Matrix class serve as a composite, with the Matrix class being composed within it?

```
class Matrix
{
private:
    int** data;
    int rows;
    int cols;

public:
    Matrix(int numRows=0, int numCols=0);
    ~Matrix();

    void setValue(int row, int col, int value);
    int getValue(int row, int col) const;
    void display() const;
};
```

Fig.7 (Matrix)

Also, overload [][] operator for your matrix Class.

Define another class named SparseMatrix.

```
class SparseMatrix
{
private:
    int non_zero;
    const int colms { 3 };
    int** S_Mat;
public:
    SparseMatrix(int non_zero=0); // the constructor
    void Read_SparseMatrix();

    SparseMatrix  AddSparseMatrix(SparseMatrix B);

    void display() const;
};
```

Fig.8 (Sparse Matrix)

Instantiate two Matrix class objects and initialize them with random values obtained from a file. For instance, if you're creating a 10x10 matrix, it should consist of 100 elements. Out of these 100 elements, ensure that at least 85 percent of them are zeros.

The data of matrices should be read from a text file. Format of file should be like this for a 3x3 matrix.

```
0 3 0
0 0 0
1 0 0
```

Fig.9 (File Format)

Every line represents a row and number of elements in each line tell about elements that should be inserted in matrix in each row. This matrix will be formed from above file data.

0 3 0
0 0 0
1 0 0

You are tasked with populating two matrices using data from separate files. After successfully populating these matrices, convert them into valid Sparse Matrix objects (e.g., named sp_A, etc.). Once the data is read into these two matrices, each object of the Sparse Matrix class should now effectively represent a Sparse Matrix.

Following this, invoke a method within the Sparse Matrix class that performs the addition of these two Sparse Matrices and returns the resulting matrix. To conclude, display the initial two original matrices, the corresponding Sparse Matrices, and the resultant matrix obtained from the addition operation.