MIT World Peace University

Object Oriented Programming with Java and C++
Second Year B. Tech, Semester 1

# Sparse Matrix Operations

Practical Report

Prepared By

Krishnaraj Thadesar
Cyber Security and Forensics
Batch A2, PA 20

October 9, 2022

# Contents

# 1 Objectives

1. To Study the concept of sparse matrix, how it is stored and displayed.
2. To understand the implementation of sparse matrix operations - Simple Transpose, Fast Transpose.

# 2 Problem Statements

Write a C program for sparse matrix realization and operations on it

- Simple Transpose
- Fast Transpose

# 3 Theory

## 3.1 Sparse Matrix

A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a *Sparse Matrix*

## 3.2 Need for Converstion of Sparse Matrix to its Compact Form

There are several Advantages of using a Sparse Matrix instead of a Normal one.

1. Storage: There are lesser non-zero elements than zeros and thus lesser memory can be used to store only those elements.
2. Computing time: Computing time can be saved by logically designing a data structure traversing only non-zero elements.

## 3.3 Advantage of Fast Transpose over Simple Transpose

1. Fast Transpose Uses a smaller amount of memory than Simple Transpose.
2. Fast Transpose Sorts the Elements while inserting them into the transposed matrix as opposed to simple Transpose, which first inserts the elements into another matrix, and then sorts them, resulting in slower transposing speeds.

# 4 Platform

**Operating System**: Arch Linux x86-64
**IDEs or Text Editors Used**: Visual Studio Code
**Compilers** : gcc on linux for C

# 5 Input

- Normal Matrix
- Sparse Matrix
- Selection of whether to do Simple or Fast Transpose

# 6 Output

- Menu to choose what to do
- Converted Compact Matrix
- Fast and Simple Transposed Matrix

# 7 Code

## 7.1 Pseudo Code

### Conversion to Compact Form Pseudo Code

```
Algorithm Compact(A, m, n, B)
{
  // m and n are total number of rows
  // columns of original matrix
  B(0, 0) = m;
  B(0, 1) = n;
  k = 1;
  for i= 0 to m
  {
    for j = 0 to n
    {
      if(A(i, j) != 0)
      {
        B(k, 0) = i;
        B(k, 1) = j;
        B(k, 2) = A(i, j);
        k++;
      }
    }
  }
  B(0, 2) = k - 1;
}
```

### Simple Transpose Pseudo Code

```
Algorithm Transpose(A, B)
{
  // A is the Sparse Matrix
  // B is the Transpose Matrix

  (m, n, t) = (A(0, 0), A(0, 1), A(0, 2))
  B[0, 0] B[0, 1], B[0, 2] = (n, m, t)
```

```
  if t <= 0:
  {
    return 0;
  }
  q = 1;
  for(int i = 0; i < n; i++)
  {
    for(int j = 1; j <= t; j++)
    {
      if(A[j, i]] == i)
      {
        B[q, 0], B[q, 1], B[q, 2] = A[p, 1], A[p, 0], A[p, 2]
        q++;
      }
    }
  }
}
```

**Fast Transpose Pseudo Code**

```
algorithm Fast_transpose(a, b)
{
  num_rows =  a[0][0]
  num_cols = a[0][1]
  num_terms = a[0][2]
  for(int i = 0; i <= num_cols; i++)
  {
    s[i] = 0;
  }
  for(int i = 1; i <= num_terms; i++)
  {
    s[a[i][1]]++;
  }

  T[0] = 1;
  for(i = 1; i <num_cols; i++)
  {
    i = T[a[i][1]];
    b[T[j][0]] = a[i][1];
    b[T[j][1]] = a[i][0];
    b[T[j][2]] = a[i][2];
    T[a[i][1]]++;
  }
}
```

## 7.2 C Implementation of Problem Statement

```
#include <stdio.h>
#include <stdlib.h>

// Converts a given variable length simple matrix, into a given sparse matrix.
void convert_to_sparse(int *simple_mat, int rows_of_simple, int cols_of_simple,
    int sparse_mat[][3])
{
    sparse_mat[0][0] = rows_of_simple;
    sparse_mat[0][1] = cols_of_simple;
```

```c
    int sparse_rows = 1;
    for (int i = 0; i < rows_of_simple; i++)
    {
        for (int j = 0; j < cols_of_simple; j++)
        {
            if (simple_mat[i * rows_of_simple + j] != 0)
            {
                sparse_mat[sparse_rows][0] = i;
                sparse_mat[sparse_rows][1] = j;
                sparse_mat[sparse_rows][2] = simple_mat[i * rows_of_simple + j];
                sparse_rows++;
            }
        }
    }
    sparse_mat[0][2] = sparse_rows - 1;
}

// Simple Transposes the given sparse matrix, and stores the value in the given transpose matrix.
int basic_transpose(int sparse_mat[][3], int transposed_mat[][3])
{
    // Assigning some basic values
    int rows_simple_mat = sparse_mat[0][0];
    int cols_simple_mat = sparse_mat[0][1];
    int no_of_vals = sparse_mat[0][2];

    // Assigning them to the transposed matrix
    transposed_mat[0][0] = cols_simple_mat;
    transposed_mat[0][1] = rows_simple_mat;
    transposed_mat[0][2] = no_of_vals;

    // Error check
    if (no_of_vals == 0)
    {
        printf("Cannot transpose as there are no elements in the matrix\n");
        return 0;
    }

    // counter variable starting from 1 coz 0 is header
    int row_count_t_matrix = 1;

    // transposing
    for (int i = 0; i < cols_simple_mat; i++)
    {
        for (int j = 1; j <= no_of_vals; j++)
        {
            if (sparse_mat[j][1] == i)
            {
                transposed_mat[row_count_t_matrix][0] = sparse_mat[j][1];
                transposed_mat[row_count_t_matrix][1] = sparse_mat[j][0];
                transposed_mat[row_count_t_matrix][2] = sparse_mat[j][2];
                row_count_t_matrix++;
            }
        }
    }
}

// Fast Transposes the given sparse matrix, and stores the value in the given transpose matrix.
```

```c
void fast_transpose(int sparse_mat[][3], int transponsed_mat[][3])
{
    // Assigning some basic values
    int rows_simple_mat = sparse_mat[0][0];
    int cols_simple_mat = sparse_mat[0][1];
    int no_of_vals = sparse_mat[0][2];

    int transposed_mat[20][3];

    // Assigning them to the transposed matrix
    transposed_mat[0][0] = cols_simple_mat;
    transposed_mat[0][1] = rows_simple_mat;
    transposed_mat[0][2] = no_of_vals;

    // counter variable starting from 1 coz 0 is header
    int row_count_t_matrix = 1;

    for (int i = 0; i <= cols_simple_mat; i++)
    {
        s[i] = 0;
    }
    for (int i = 1; i <= no_of_vals; i++)
    {
        s[a[i][1]]++;
    }

    T[0] = 1;
    for (i = 1; i < cols_simple_mat; i++)
    {
        i = T[a[i][1]];
        b[T[j][0]] = a[i][1];
        b[T[j][1]] = a[i][0];
        b[T[j][2]] = a[i][2];
        T[a[i][1]]++;
    }
}

// Accepts a Variable length 2 Dimensional Matrix
void accept_mat(int *matrix, int rows, int cols)
{
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            scanf("%d", &matrix[i * cols + j]);
        }
    }
}

// Displays a variable length 2 Dimensional Matrix
void display_mat(int *matrix, int rows, int cols)
{
    printf("\n");
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < cols; j++)
        {
            printf("%d ", matrix[i * cols + j]);
        }
```

```
        printf("\n");
    }
    printf("\n");
}
int main()
{
    int rows_simple = 5, cols_simple = 5, choice = 0;
    int sparse_mat_1_rows = 8;
    int sparse_mat_2_rows = 9;
    int sparse_mat[40][3], transposed_mat[40][3], result_sparse[40][3];

    // defining a simple matrix
    int simple_mat[5][5] = {
        {1, 0, 0, 0, 1},
        {4, 2, 0, 0, 3},
        {0, 0, 0, 0, 4},
        {3, 3, 3, 0, 0},
        {3, 4, 1, 0, 0}};

    // Defining a sparse matrix so we can add it.
    int sparse_mat_1[8][3] = {
        {5, 5, 7},
        {0, 0, 1},
        {0, 0, 4},
        {1, 0, 4},
        {2, 4, 4},
        {3, 2, 3},
        {4, 0, 3},
        {4, 1, 4},
    };

    // Defining another sparse matrix, so we can add it to the previous one.
    int sparse_mat_2[9][3] = {{5, 5, 7},
                              {0, 0, 1},
                              {0, 0, 4},
                              {1, 1, 4},
                              {2, 4, 4},
                              {3, 2, 3},
                              {4, 0, 3},
                              {4, 1, 4},
                              {4, 4, 2}};

    printf("What do you wanna do with the matrices?"
           " \n1. Simple Transpose\n2. Fast Transpose\n\n");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:

        printf("Enter the rows of the Matrix: (Max 5) \n");
        scanf("%d", &rows_simple);
        printf("Enter the columns of the Matrix: (Max 5) \n");
        scanf("%d", &cols_simple);
        pritnf("Enter the simple matrix: \n");
        accept_mat(&simple_mat[0][0], rows_simple, cols_simple);
        printf("The Matrix you entered is: \n\n");
        display_mat(&simple_mat[0][0], rows_simple, cols_simple);
        convert_to_sparse(&simple_mat[0][0], rows_simple, cols_simple, sparse_mat)
    ;
```

```
        printf("The Sparse Matrix is : \n\n");
        display_mat(&sparse_mat[0][0], sparse_mat[0][2] + 1, 3);
        basic_transpose(sparse_mat, transposed_mat);
        printf("The Transposed Matrix is: \n\n");
        display_mat(&transposed_mat[0][0], transposed_mat[0][2] + 1, 3);
        break;
    case 2:

        printf("Enter the rows of the Matrix: (Max 5) \n");
        scanf("%d", &rows_simple);
        printf("Enter the columns of the Matrix: (Max 5) \n");
        scanf("%d", &cols_simple);

        printf("The Matrix you entered is: \n\n");
        display_mat(&simple_mat[0][0], rows_simple, cols_simple);
        convert_to_sparse(&simple_mat[0][0], rows_simple, cols_simple, sparse_mat);
        printf("The Sparse Matrix is : \n\n");
        display_mat(&sparse_mat[0][0], sparse_mat[0][2] + 1, 3);
        fast_transpose(sparse_mat, transposed_mat);
        printf("The Transposed Matrix is: \n\n");
        display_mat(&transposed_mat[0][0], transposed_mat[0][2] + 1, 3);
        break;
    default:
        printf("Try again\n");
    }

    return 0;
}
```

Listing 1: Main.Cpp

### 7.3 C Output

```
What do you wanna do with the matrices?
1. Simple Transpose
2. Fast Transpose
3. Add 2 Sparse Matrices

1
The Matrix you entered is:


1 0 0 0 1
4 2 0 0 3
0 0 0 0 4
3 3 3 0 0
3 4 1 0 0

The Sparse Matrix is :


5 5 12
0 0 1
0 4 1
1 0 4
1 1 2
1 4 3
2 4 4
```

```
3 0 3
3 1 3
3 2 3
4 0 3
4 1 4
4 2 1

The Transposed Matrix is:


5 5 12
0 0 1
0 1 4
0 3 3
0 4 3
1 1 2
1 3 3
1 4 4
2 3 3
2 4 1
4 0 1
4 1 3
4 2 4
```

Listing 2: Output

## 8 Time Complexity

Time Complexity for Simple Transpose : O(N*T)
Time Complexity for Fast Transpose : O(N+T)

## 9 Conclusion

Thus, implemented sparse matrix Operations assignment. This System is able to perform different operations on sparse matrices such as simple and fast transpose and their time complexities.

## 10 FAQs

1. **What is a sparse matrix? List the applications?**
   A matrix is a two-dimensional data object made of m rows and n columns, therefore having total m x n values. If most of the elements of the matrix have 0 value, then it is called a *Sparse Matrix*

   It has several Applications in various fields, mostly involving Mathematics.

   (a) Sparse matrices can be useful for computing large-scale applications that dense matrices cannot handle. One such application involves solving partial differential equations by using the finite element method. The coefficient matrix is mostly sparse. Also, the size of the coefficient matrix is large in order to get an accurate approximation to the solution of PDEs. Therefore, practical finite element method applications always rely on sparse matrices and sparse matrix operations.

   (b) Sparse matrices are at the heart of Linear Algebraic Systems. Needless to say everything of any significance happening in a sufficiently complex computer system will require lots of Linear Algebraic operations. You really cannot represent very large high dimensional matrices (when most of them have zeroes) in memory and do manipulations on them.

   (c) Computer Graphics, Recommendtion Algorithms used by Search Engines, Machine Learning, Neural Networks, and Information Retrieval all rely on large matrices, filled mostly with null or 0 values.

2. **Represent sparse matrices with suitable data structures? Explain with an example simple and fast transpose?**

   (a) Using Arrays: 2D array is used to represent a sparse matrix in which there are three rows named as

   Row: Index of row, where non-zero element is located Column: Index of column, where non-zero element is located Value: Value of the non zero element located at index - (row,column)

   (b) Using Linked Lists In linked list, each node has four fields. These four fields are defined as:

   Row: Index of row, where non-zero element is located Column: Index of column, where non-zero element is located Value: Value of the non zero element located at index – (row,column) Next node: Address of the next node

   (c) As a Dictionary : where row and column numbers are used as keys and values are matrix entries. This method saves space but sequential access of items is costly.

3. **Find out the addition of two sparse matrices in triplet form and also find Simple and Fast transpose**

Matrix 1:

| **4** | **5** | **6** |
|---|---|---|
| 0 | 3 | 5 |
| 1 | 3 | 8 |
| 1 | 4 | 45 |
| 2 | 3 | 4 |
| 3 | 2 | 45 |
| 4 | 1 | 2 |

Matrix 2:

| **4** | **5** | **6** |
|---|---|---|
| 0 | 3 | 7 |
| 0 | 4 | 6 |
| 1 | 4 | 4 |
| 2 | 1 | 8 |
| 3 | 2 | 45 |
| 4 | 4 | 21 |

The Addition of these Matrices would be:

| **4** | **5** | **9** |
|---|---|---|
| 0 | 3 | 12 |
| 0 | 4 | 6 |
| 1 | 3 | 8 |
| 1 | 4 | 49 |
| 2 | 1 | 8 |
| 2 | 3 | 4 |
| 3 | 2 | 90 |
| 4 | 1 | 2 |
| 4 | 4 | 21 |

The Transpose of This Matrix would be:

| **4** | **5** | **9** |
|---|---|---|
| 1 | 2 | 8 |
| 1 | 4 | 2 |
| 2 | 3 | 40 |
| 3 | 0 | 12 |
| 3 | 1 | 8 |
| 3 | 2 | 4 |
| 4 | 0 | 6 |
| 4 | 1 | 49 |
| 4 | 4 | 21 |