DATA STRUCTURES (ITPC-203)

Sparce Matrices

Mrs. Sanga G. Chaki

Department of Information Technology

Dr. B. R. Ambedkar National Institute of Technology, Jalandhar

Contents

- Sparse Matrices

- Representations

Sparse Matrix

1. A sparse matrix is defined as the matrix of order MxN which has the **number of zero values strictly greater than the number of non-zero values**
2. Those matrices which contain more non-zero values than zero values, they are called dense matrices.
3. Number of zeros = 14
4. Number of non-zero values = 6

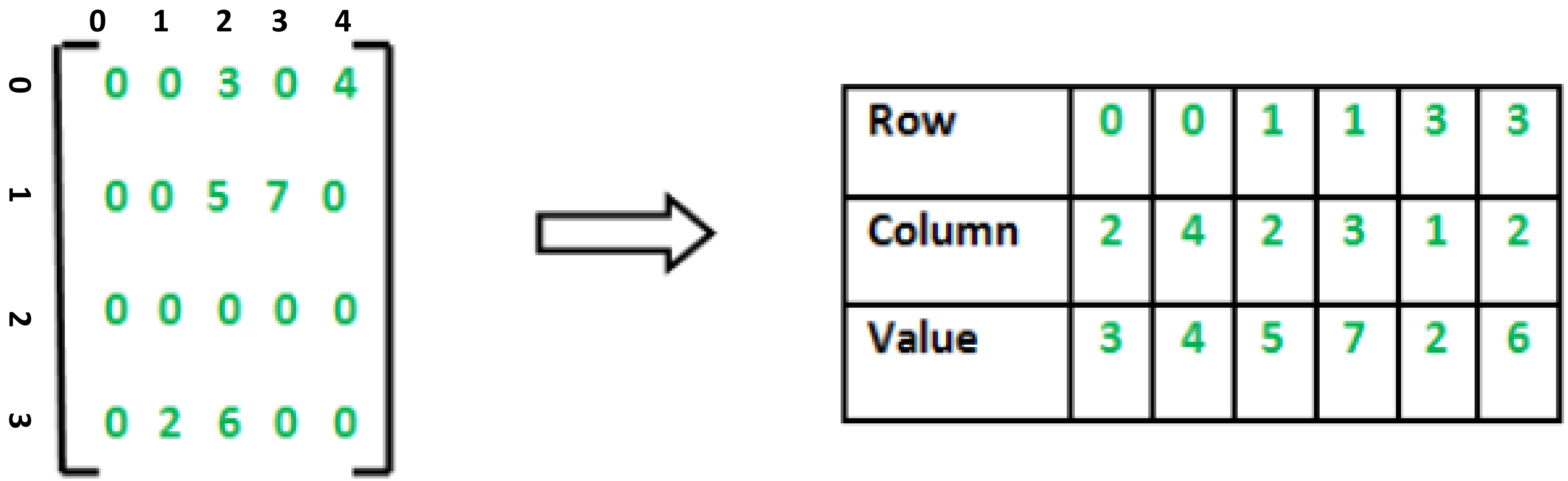
$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix}$$

Sparse Matrix

1. If the matrix is sparse, the normal row major or column major arrangement will waste a lot of memory.
2. This is because if majority of elements of the matrix are 0.
3. Need to have an alternative storage
4. Through which we can store only the non-zero elements and keep intact the functionality of the matrix.
5. This is the concept of a compact matrix of a sparse matrix

Sparse Matrix - Representation

1. Store non-zero elements as with triples- (Row, Column, value).
2. Here, number of columns in compact matrix must be equal to number of non - zero elements in the sparse matrix



$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 0 & 0 & 3 & 0 & 4 \\ 1 & 0 & 0 & 5 & 7 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 2 & 6 & 0 & 0 \end{array} \Longrightarrow$$

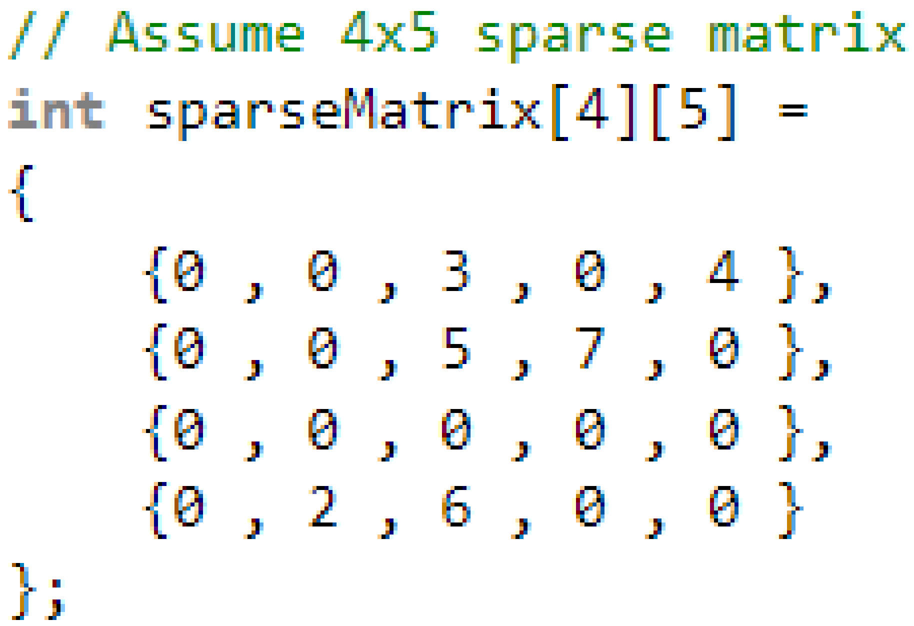
Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6

Sparse Matrix

1. Implementation –
2. Arrays
3. Linked Lists

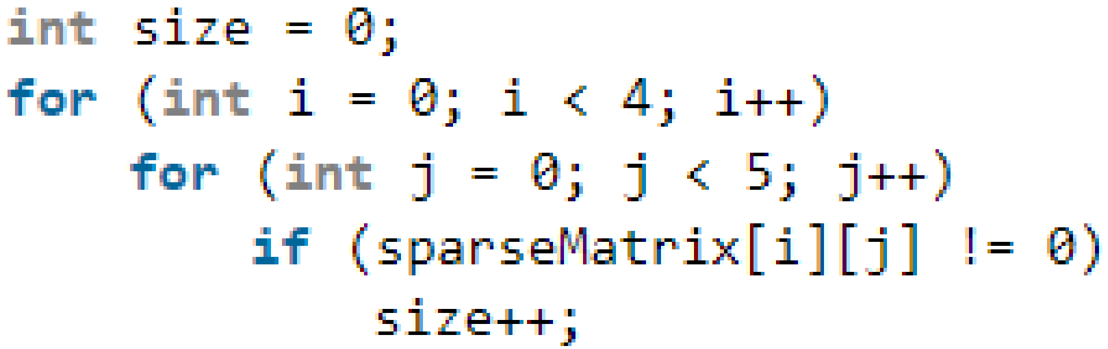
Sparse Matrix – Array Implementation

1. The sparse matrix:



```
// Assume 4x5 sparse matrix
int sparseMatrix[4][5] =
{
    {0 , 0 , 3 , 0 , 4 },
    {0 , 0 , 5 , 7 , 0 },
    {0 , 0 , 0 , 0 , 0 },
    {0 , 2 , 6 , 0 , 0 }
};
```

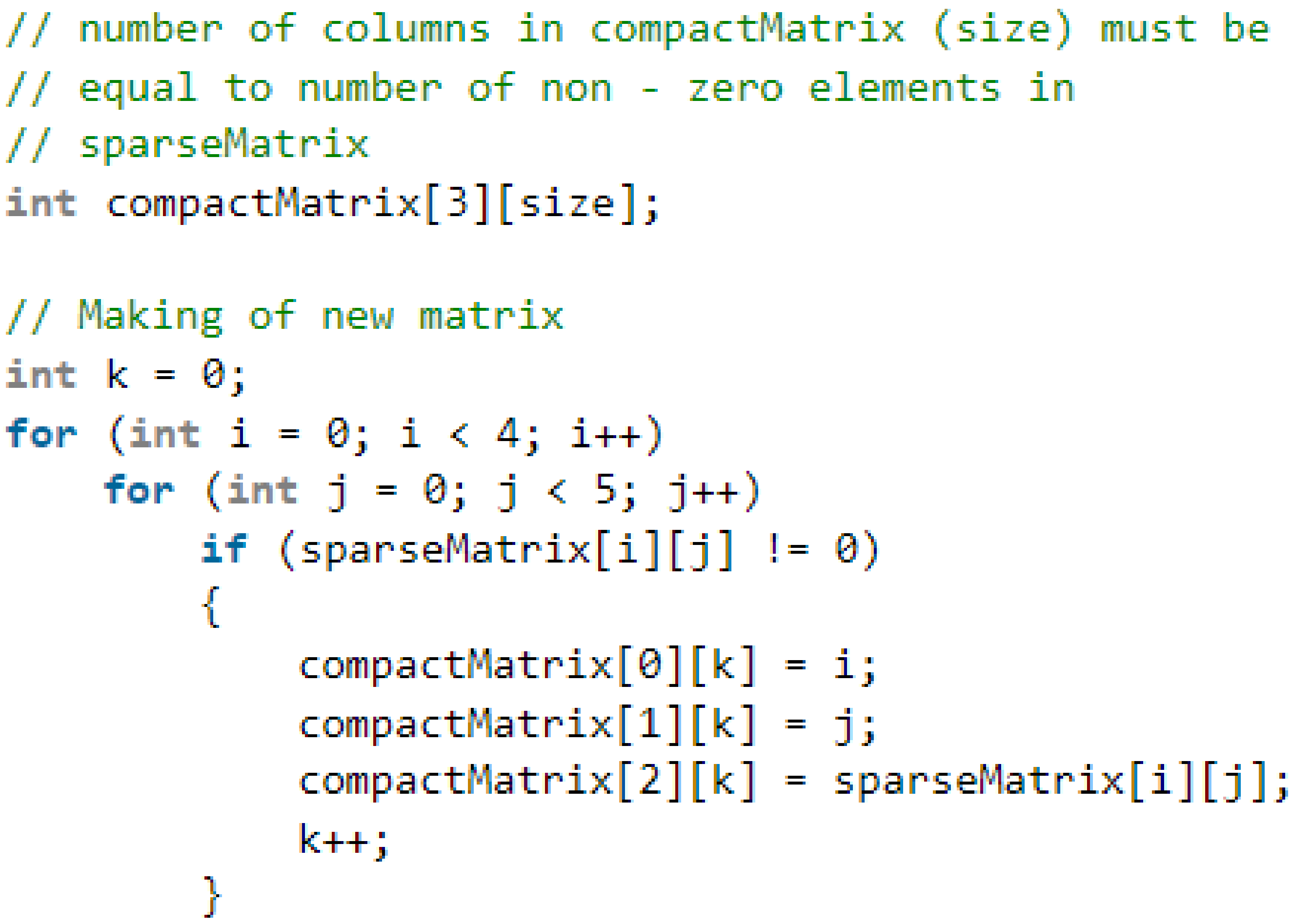
1. To get the number of non-zero elements:



```
int size = 0;
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 5; j++)
        if (sparseMatrix[i][j] != 0)
            size++;
```

Sparse Matrix – Array Implementation

1. Creating the compact matrix:

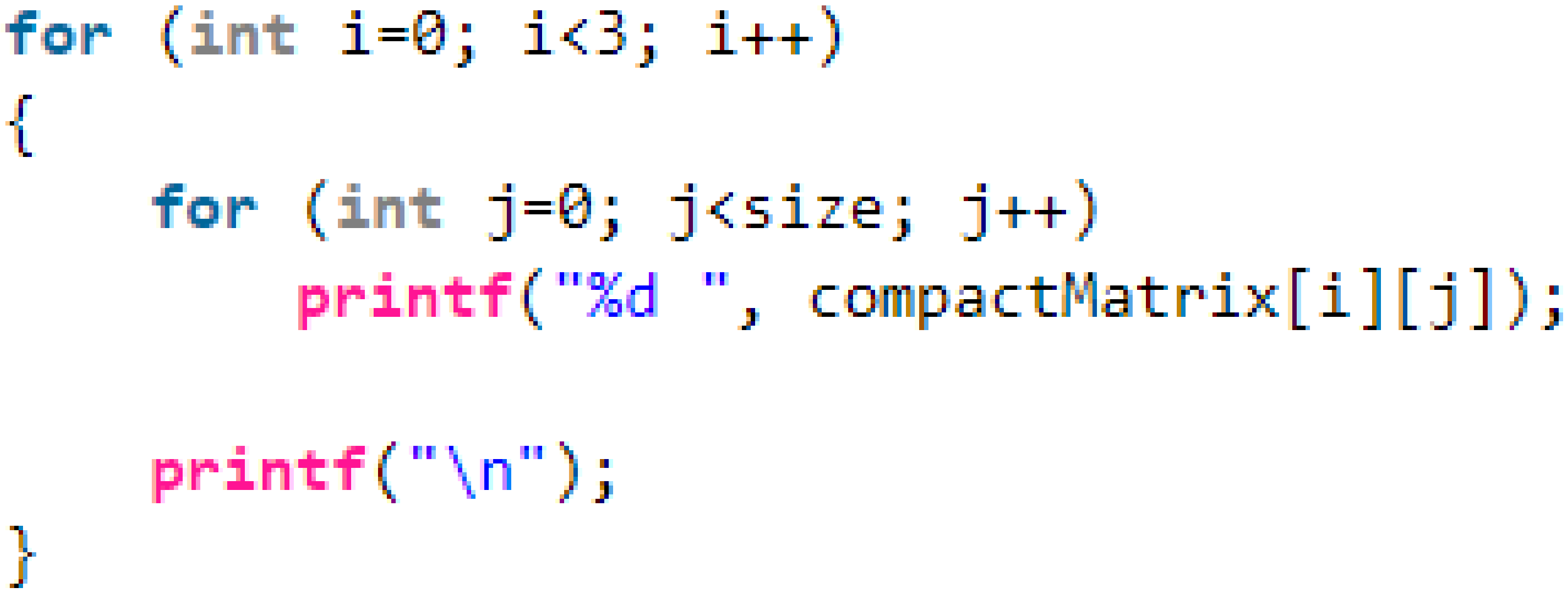


```
// number of columns in compactMatrix (size) must be
// equal to number of non - zero elements in
// sparseMatrix
int compactMatrix[3][size];

// Making of new matrix
int k = 0;
for (int i = 0; i < 4; i++)
    for (int j = 0; j < 5; j++)
        if (sparseMatrix[i][j] != 0)
        {
            compactMatrix[0][k] = i;
            compactMatrix[1][k] = j;
            compactMatrix[2][k] = sparseMatrix[i][j];
            k++;
        }
```

Sparse Matrix – Array Implementation

1. Printing the compact matrix:



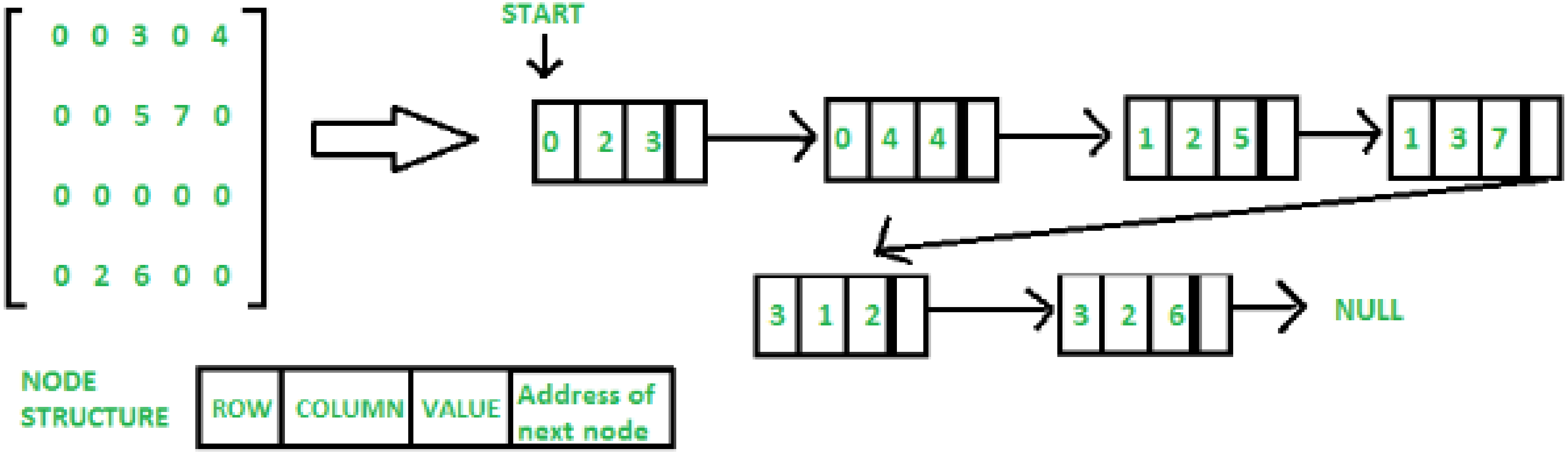
```
for (int i=0; i<3; i++)
{
    for (int j=0; j<size; j++)
        printf("%d ", compactMatrix[i][j]);

    printf("\n");
}
```

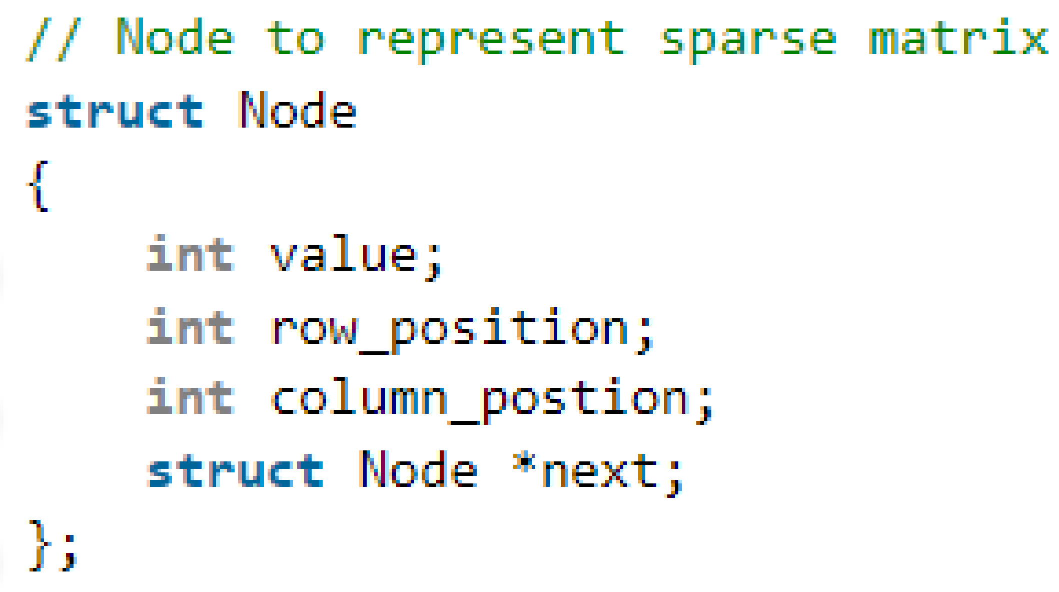
2. Time Complexity: O(NM), where N is the number of rows in the sparse matrix, and M is the number of columns in the sparse matrix.
3. Auxiliary Space: O(NM), where N is the number of rows in the sparse matrix, and M is the number of columns in the sparse matrix.

Sparse Matrix - Using Linked Lists

1. In linked list, each node has four fields.
   a) Row: Index of row, where non-zero element is located
   b) Column: Index of column, where non-zero element is located
   c) Value: Value of the non zero element located at index – (row, column)
   d) Next node: Address of the next node



Sparse Matrix - Using Linked Lists



```
// Node to represent sparse matrix
struct Node
{
    int value;
    int row_position;
    int column_postion;
    struct Node *next;
};
```

Sparse Matrix - Using Linked Lists

```c
// Function to create new node
void create_new_node(struct Node** start, int non_zero_element,
                     int row_index, int column_index )
{
    struct Node *temp, *r;
    temp = *start;
    if (temp == NULL)
    {
        // Create new node dynamically
        temp = (struct Node *) malloc (sizeof(struct Node));
        temp->value = non_zero_element;
        temp->row_position = row_index;
        temp->column_postion = column_index;
        temp->next = NULL;
        *start = temp;

    }
    else
    {
        while (temp->next != NULL)
            temp = temp->next;

        // Create new node dynamically
        r = (struct Node *) malloc (sizeof(struct Node));
        r->value = non_zero_element;
        r->row_position = row_index;
        r->column_postion = column_index;
        r->next = NULL;
        temp->next = r;

    }
}
```

Thanks!