

**OBJECTIVES:**

- Revised different algorithms that will help us in this course to solve different problems.

**BACKGROUND STUDY**

- Should have prior knowledge on any programming language to implement the algorithm.
- Need knowledge on function, 2D array, link list, and pointer.

**RECOMMENDED READING**

- Let Us C – By Yashavant P. Kanetkar (Recursion).
- Introduction to Algorithms – By Thomas H. Cormen, Charles E. Leiserson, Ronal L. Rivest, and Clifford Stein (Graph Basics).

**Recursion**

A function is called 'recursive' if a statement within the body of a function calls the same function. Sometimes called 'circular definition', recursion is thus the process of defining something in terms of itself. Suppose we want to calculate the factorial value of an integer. As we know, the factorial of a number is the product of all the integers between 1 and that number. For example, 4 factorial is  $4 * 3 * 2 * 1$ . This can also be expressed as  $4! = 4 * 3!$  where '!' stands for factorial. Thus factorial of a number can be expressed in the form of itself. Hence this can be programmed using recursion. Following is the recursive version of the function to calculate the factorial value.

```
main( )
{
    int a, fact ;
    printf ( "\nEnter any number " ) ;
    scanf ( "%d", &a ) ;
    fact = rec ( a ) ;
    printf ( "Factorial value = %d", fact ) ;
}
rec ( int x )
{
    int f ;
    if ( x == 1 )
        return ( 1 ) ;
    else
        f = x * rec ( x - 1 ) ;
    return ( f ) ;
}
```

And here is the output for four runs of the program

```
Enter any number 1
Factorial value = 1
Enter any number 2
Factorial value = 2
Enter any number 3
```

## Graph Basics

A graph  $G = (V, E)$  can be represented using adjacency matrix or adjacency list. The adjacency-list representation provides a compact way to represent *sparse* graphs. We may prefer an adjacency-matrix representation, however, when the graph is *dense*.

The **adjacency-list representation** of a graph  $G = (V, E)$  consists of an array  $Adj$  of  $|V|$  lists, one for each vertex in  $V$ . For each  $u \in V$ , the adjacency list  $Adj[u]$  contains all the vertices  $v$  such that there is an edge  $(u, v) \in E$ . That is,  $Adj[u]$  consists of all the vertices adjacent to  $u$  in  $G$ . (Alternatively, it may contain pointers to these vertices.)

For the **adjacency-matrix representation** of a graph  $G = (V, E)$ , we assume that the vertices are numbered  $1, 2, \dots, |V|$  in some arbitrary manner. Then the adjacency-matrix representation of a graph  $G$  consists of a  $|V| \times |V|$  matrix  $A = (a_{ij})$  such that -

$$a_{ij} = \begin{cases} 1 & \text{if } (i, j) \in E, \\ 0 & \text{otherwise.} \end{cases}$$

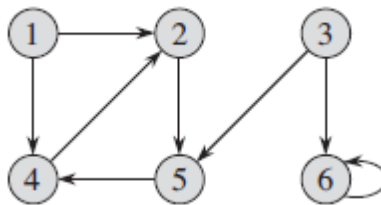
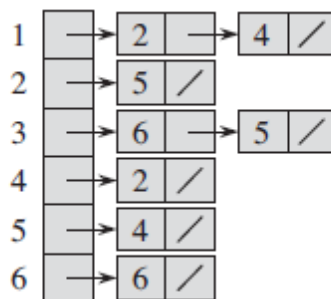


Figure 1: A directed graph

The directed graph in Figure-1 can be represented using adjacency list as following –



The directed graph in Figure-1 can be represented using adjacency matrix as following –

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Both adjacency list and adjacency matrix can be used to find the neighbors of a node. For example, in adjacency list, link list corresponds to a cell represents all neighbors of a node for which that cell is used. And in adjacency matrix if we can find a column value 1 in a row then the node corresponds to that column is a neighbor of the node correspond to that row.

### **LABROTORY EXERCISES**

1. Write a recursive function to obtain the first 25 numbers of a Fibonacci sequence. In a Fibonacci sequence the sum of two successive terms gives the third term. Following are the first few terms of the Fibonacci sequence:

1 1 2 3 5 8 13 21 34 55 89...

2. Write a program which can take a directed graph as input and represent it in an adjacency matrix (Let's say the graph shown in Figure-1). Print the adjacency matrix and check its correctness.

**Input:** a graph  $G$

**Output:** adjacency matrix

3. Write a function *find\_neighbors( $n$ )* which finds out all neighbors of node  $n$  using the adjacency matrix generated in Exercise-2.

**Input:** a graph  $G$  and a node  $n$  of  $G$

**Output:** neighbors of node  $n$