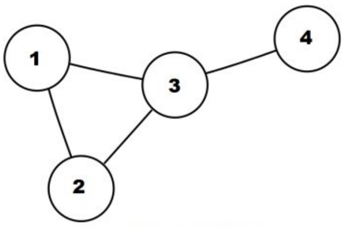
Practical–8
Implementation of Graph

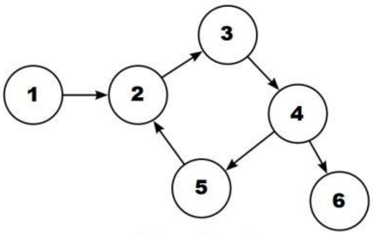
Types of Graphs in Data Structure

The most common types of graphs in data structure are mentioned below:

1. Undirected: A graph in which all the edges are bi-directional.
The edges do not point in a specific direction.



Undirected Graph



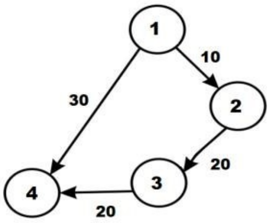
Directed Graph

2. Directed: A graph in which all the edges are uni-directional. The edges point in a single direction.

3. Weighted Graph: A graph that has a value associated with every edge. The values corresponding to the edges are called weights. A value in a weighted graph can represent quantities such as cost, distance, and time, depending on the graph. Weighted graphs are typically used in modeling computer networks.

An edge in a weighted graph is represented as (u, v, w), where:

- u is the source vertex
- v is the destination vertex
- w represents the weight associated to go from u to v



Weighted Graph

4. Unweighted Graph: A graph in which there is no value or weight associated with the edge. All the graphs are unweighted by default unless there is a value associated.

An edge of an unweighted graph is represented as (u, v), where:

- u represents the source vertex

- v is the destination vertex Graph Representation in Data Structure

Adjacency Matrix

An Adjacency Matrix is the simplest way to represent a graph. It is a 2D array of V x V vertices with each row and column representing a vertex. The matrix is consists of “0” or “1”. 0 depicts that there is no path while 1 represents that there is a path.

Operations on Graph in Data Structure

Following are the basic graph operations in data structure:

- Add/Remove Vertex – Add or remove a vertex in a graph.
- Add/Remove Edge – Add or remove an edge between two vertices.
- Check if the graph contains a given value.
- Find the path from one vertex to another vertex.

Graph Traversal in Data Structure

Graph traversal is the process of visiting or updating each vertex in a graph. The order in which they visit the vertices is used to classify the traversals. There are two ways to implement a graph traversal:

1. Breadth-First Search (BFS) – It is a traversal operation that horizontally traverses the graph. It traverses all the nodes at a single level before moving to the next level. It begins at the root of the graph and traverses all the nodes at a single depth level before moving on to the next depth level.
2. Depth-First Search (DFS): This is another traversal operation that traverses the graph vertically. It starts with the root node of the graph and investigates each branch as far as feasible before backtracking.

Depth-First Search (DFS):

```
#include <stdio.h>
#define MAX 8
void depth_first_search(int adj[][MAX],int visited[],int
start)
{
   int stack[MAX];
   int top=-1, i;
   printf("%d-",start);
   visited[start] = 1;
   stack[++top] = start;
   while(top != -1)
   {
      start = stack[top];
      for(i = 0; i < MAX; i++)
      {
            if(adj[start][i] && visited[i] == 0)
            {
            stack[++top] = i;
```

```
           printf("%d", i);
           visited[i] = 1;
           break;
           }
      }
      if(i == MAX)
      top--;
   }
}
int main()
{
   int visited[MAX] = {0}, i, j;
   int adj[MAX][MAX];
   printf("\n Enter the adjacency matrix: ");
   for(i = 0; i < MAX; i++)
      for(j = 0; j < MAX; j++)
      scanf("%d", &adj[i][j]);

   printf("DFS Traversal: ");
   depth_first_search(adj,visited,0);
   printf("\n");
   return 0;
}
```

Breadth-First Search (BFS)

```
#include <stdio.h>
#define MAX 8
void breadth_first_search(int adj[][MAX],int visited[],int
start)
{
   int queue[MAX],rear = -1,front =-1, i;
   queue[++rear] = start;
   visited[start] = 1;
   while(rear != front)
   {
      start = queue[++front];
      if(start == 4)
      printf("5\t");
      else
      printf("%d \t",start);
      for(i = 0; i < MAX; i++)
      {
           if(adj[start][i] == 1 && visited[i] == 0)
           {
                queue[++rear] = i;
                visited[i] = 1;
           }
      }
   }
}
int main()
{
   int visited[MAX] = {0};
```

```
        int adj[MAX][MAX], i, j;
        printf("\n Enter the adjacency matrix: ");

        for(i = 0; i < MAX; i++)
           for(j = 0; j < MAX; j++)
                scanf("%d", &adj[i][j]);

        breadth_first_search(adj,visited,0);
        return 0;
    }
```

Exercise

1. Write a program to implement undirected graph using adjacency matrix. Print the information of the graph such as number of edges, edges list, degree of each vertex.
2. Write a program to implement traversal of graph using DFS
3. Write a program to implement traversal of graph using BFS.