Graphs

A Graph is a non-linear data structure consisting of vertices and edges. The vertices are sometimes also referred to as nodes and the edges are lines or arcs that connect any two nodes in the graph. More formally a Graph is composed of a set of vertices(**V**) and a set of edges(**E**). The graph is denoted by **G(V, E)**.

Representations of Graph

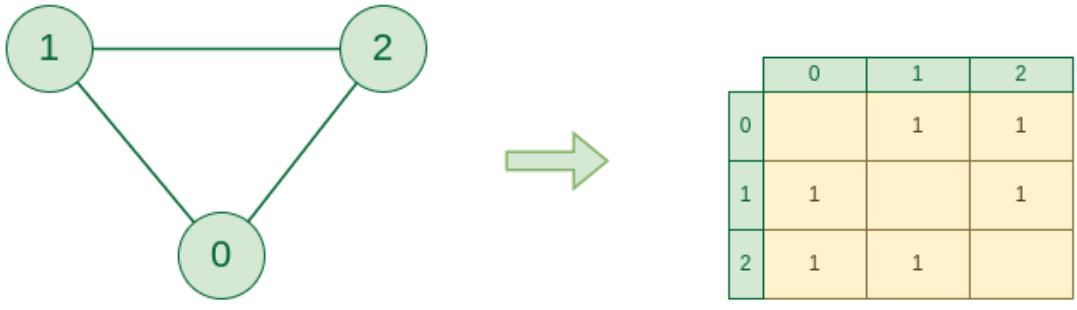
1. Adjacency Matrix
2. Adjacency Linked List

Adjacency Matrix

An adjacency matrix is a way of representing a graph as a matrix of Boolean (0's and 1's)

Let's assume there are **n** vertices in the graph So, create a 2D matrix **adjMat[n][n]** having dimension n x n.

- *If there is an edge from vertex **i** to **j**, mark **adjMat[i][j]** as **1**.*
- *If there is no edge from vertex **i** to **j**, mark **adjMat[i][j]** as **0**.*



Undirected Graph

Adjacency Matrix

Graph Representation of Undirected graph to Adjacency Matrix

- **Code:**

```cpp
using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    // adjacency matrix for undirected graph
    // time complexity: O(n)
    int adj[n+1][n+1];
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u][v] = 1;
        adj[v][u] = 1; // this statement will be removed in case of directed graph
    }
    return 0;
}
```

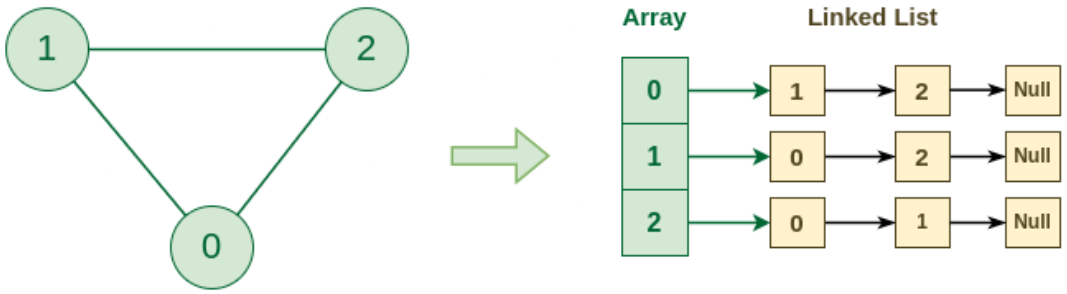
- **Space Complexity:**
    - The space needed to represent a graph using its adjacency matrix is n^2 locations. Space complexity = (n*n), It is a costly method as n^2 locations are consumed.

Adjacency Linked List

An array of Lists is used to store edges between two vertices. The size of array is equal to the number of **vertices (i.e, n)**. Each index in this array represents a specific vertex in the graph. The entry at the index i of the array contains a linked list containing the vertices that are adjacent to vertex **i**.

Let's assume there are **n** vertices in the graph So, create an **array of list** of size **n** as **adjList[n].**

- *adjList[0] will have all the nodes which are connected (neighbour) to vertex **0**.*
- *adjList[1] will have all the nodes which are connected (neighbour) to vertex **1** and so on.*



Undirected Graph

Adjacency List

Graph Representation of Undirected graph to Adjacency List

- **Code:**

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    // adjacency list for undirected graph
    // time complexity: O(2E)
    vector<int> adj[n+1];
    for(int i = 0; i < m; i++)
    {
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        adj[v].push_back(u); //excluded in undirected
    }
    return 0;
}
```

- **Space Complexity:**
  - In this representation, for an undirected graph, each edge data appears twice. For example, nodes 1 and 2 are adjacent hence node 2 appears in the list of node 1, and node 1 appears in the list of node 2. So, the space needed to represent an undirected graph using its adjacency list is 2 x E locations, where E denotes the number of edges.
  - Space complexity = O(2xE)

Weighted Graphs

Previously we considered all the graphs to have a unit weight(or cost) to traverse from one edge to another. However, what if the cost to travel from one edge to another is different. In such cases we also include weights to be considered per edge.

```
using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    // adjacency matrix for weighted undirected graph
    // initialise all weights to 0

    int adj[n+1][n+1]={0};

    for(int i = 0; i < m; i++)
    {
        int u, v, w;
        cin >> u >> v >> w;
        adj[u][v] = w;
        adj[v][u] = w; // this statement will be
removed in case of directed graph
    }
    return 0;
}
```