DATA STRUCTURES AND ALGORITHMS

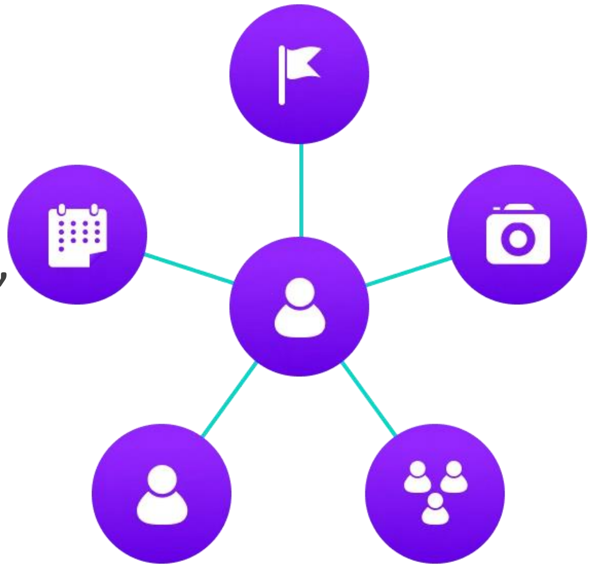
DR SAMABIA TEHSIN

BS (AI)

Graph Data Structure

A graph data structure is a collection of nodes that have data and are connected to other nodes.

Let's try to understand this through an example. On facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.



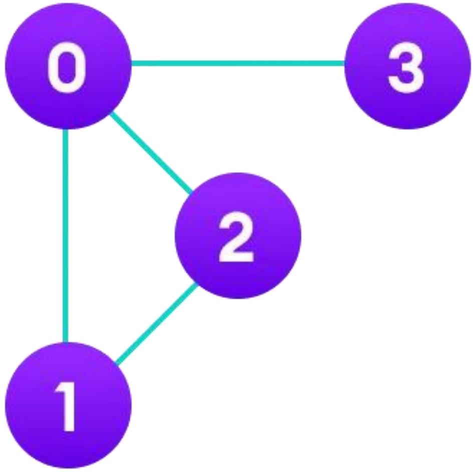
Every relationship is an edge from one node to another. Whether you post a photo, join a group, like a page, etc., a new edge is created for that relationship.

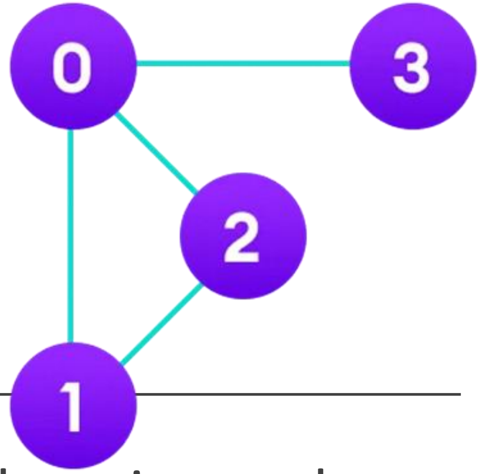
Graph Data Structure

A graph is a data structure (V, E) that consists of

- A collection of vertices V
- A collection of edges E, represented as ordered pairs of vertices (u,v)

V = {0, 1, 2, 3}
E = {(0,1), (0,2), (0,3), (1,2)}
G = {V, E}



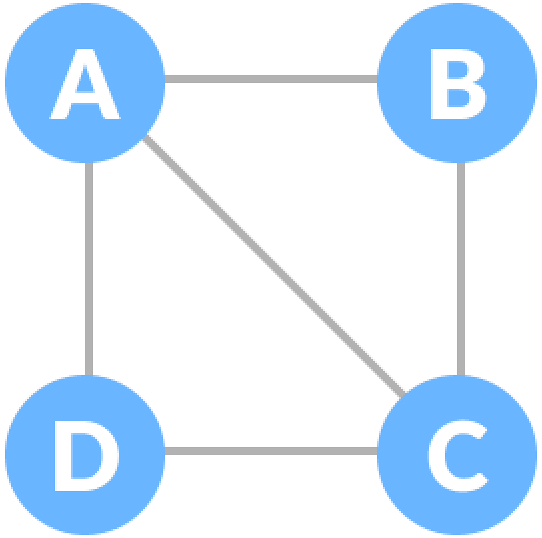


Graph Terminology

- **Adjacency**: A vertex is said to be adjacent to another vertex if there is an edge connecting them. Vertices 2 and 3 are not adjacent because there is no edge between them.
- **Path**: A sequence of edges that allows you to go from vertex A to vertex B is called a path. 0-1, 1-2 and 0-2 are paths from vertex 0 to vertex 2.
- **Directed Graph**: A graph in which an edge (u,v) doesn't necessarily mean that there is an edge (v, u) as well. The edges in such a graph are represented by arrows to show the direction of the edge.

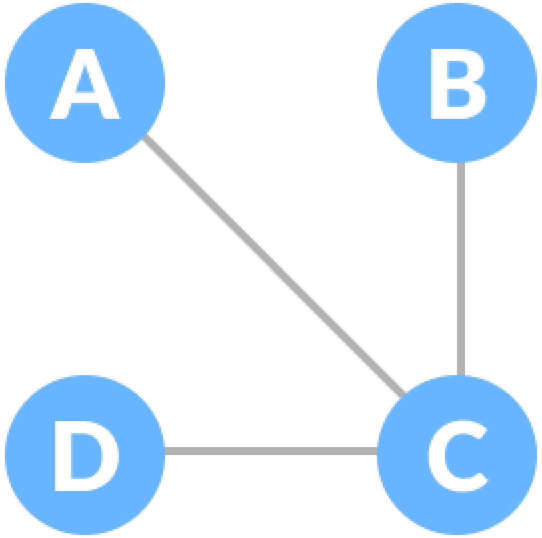
Undirected Graphs

An **undirected graph** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).



Connected Graphs

A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.



Graph Representation

Graphs are commonly represented in two ways:

1. **Adjacency Matrix**
2. **Adjacency List**

Graph Representation: Adjacency Matrix

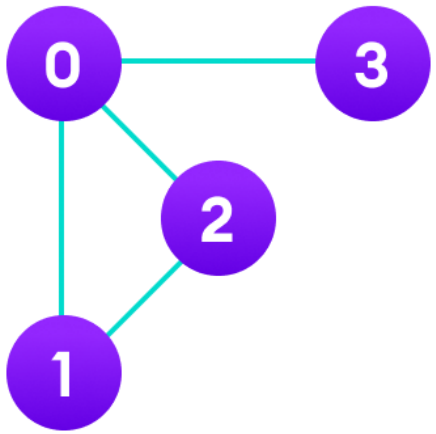
Adjacency Matrix

An adjacency matrix is a 2D array of V x V vertices.

Each row and column represent a vertex.

If the value of any element a[i][j] is 1, it represents that there is an edge connecting vertex i and vertex j.

Graph Representation : Adjacency Matrix



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

It is an undirected graph, for edge (0,2), we also need to mark edge (2,0); making the adjacency matrix symmetric about the diagonal.

Graph Representation : Adjacency Matrix

Each cell in the above table/matrix is represented as A_{ij}, where i and j are vertices. The value of A_{ij} is either 1 or 0 depending on whether there is an edge from vertex i to vertex j.

If there is a path from i to j, then the value of A_{ij} is 1 otherwise its 0. For instance, there is a path from vertex 1 to vertex 2, so A_{12} is 1 and there is no path from vertex 1 to 3, so A_{13} is 0.
In case of undirected graphs, the matrix is symmetric about the diagonal because of every edge (i,j), there is also an edge (j,i).

Graph Representation : Adjacency Matrix

Pros of Adjacency Matrix

- The basic operations like adding an edge, removing an edge, and checking whether there is an edge from vertex i to vertex j are extremely time efficient, constant time operations.
- If the graph is dense and the number of edges is large, an adjacency matrix should be the first choice.
- The biggest advantage, however, comes from the use of matrices. The recent advances in hardware enable us to perform even expensive matrix operations on the GPU.
- By performing operations on the adjacent matrix, we can get important insights into the nature of the graph and the relationship between its vertices.

Graph Representation : Adjacency Matrix

Cons of Adjacency Matrix

- The VxV space requirement of the adjacency matrix makes it a memory hog. Graphs out in the wild usually don't have too many connections and this is the major reason why adjacency lists are the better choice for most tasks.
- While basic operations are easy, operations like inEdges and outEdges are expensive when using the adjacency matrix representation.

```
// Adjacency Matrix representation in C++

#include <iostream>

using namespace std;


class Graph {

   private:

  bool** adjMatrix;

  int numVertices;

public:
 // Initialize the matrix to zero
 Graph(int numVertices) {
   this->numVertices = numVertices;
   adjMatrix = new bool*[numVertices];
   for (int i = 0; i < numVertices; i++) {
        adjMatrix[i] = new bool[numVertices];
        for (int j = 0; j < numVertices; j++)
               adjMatrix[i][j] = false;
   }
 }
```

```
// Add edges
  void addEdge(int i, int j) {
    adjMatrix[i][j] = true;
    adjMatrix[j][i] = true;
  }
  // Remove edges
  void removeEdge(int i, int j) {
    adjMatrix[i][j] = false;
    adjMatrix[j][i] = false;
  }
```

```
// Print the martix
 void toString() {
   for (int i = 0; i < numVertices; i++) {
     cout << i << " : ";
     for (int j = 0; j < numVertices; j++)
       cout <<“j”<< adjMatrix[i][j] << " ";
     cout << "\n";
   }
 }
```

```cpp
 ~Graph() {
    for (int i = 0; i < numVertices; i++)
        delete[] adjMatrix[i];
    delete[] adjMatrix;
  }
};
```

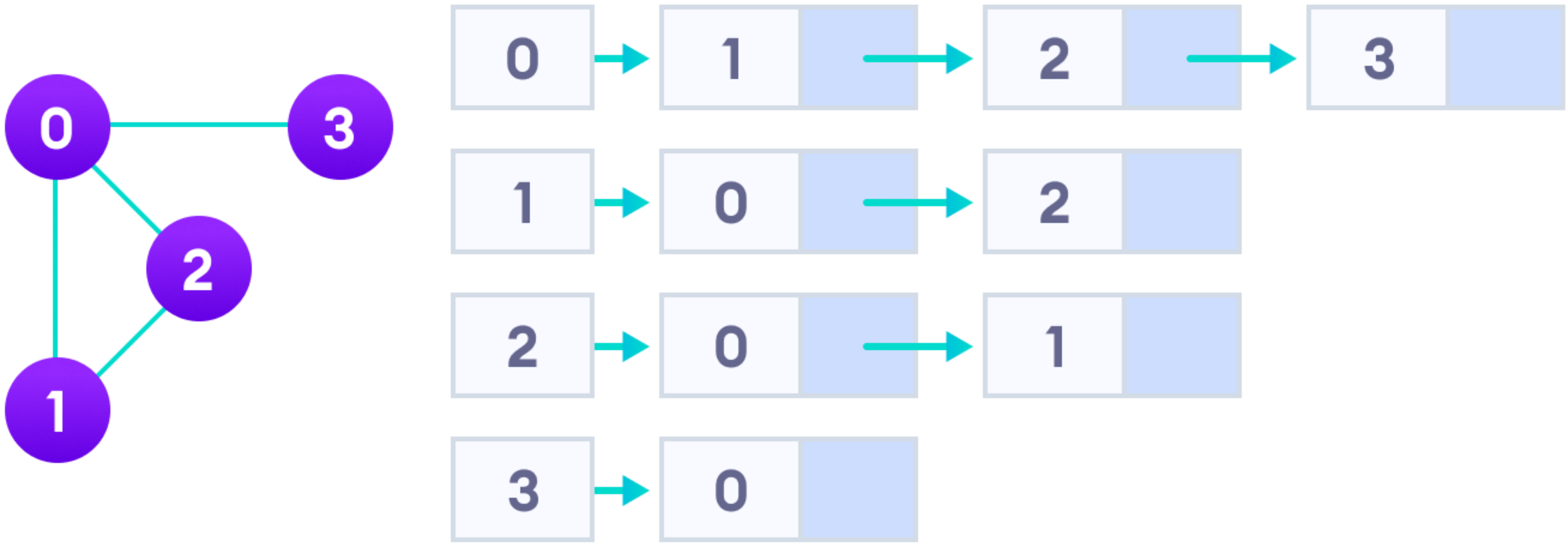
```cpp
int main() {
  Graph g(4);

  g.addEdge(0, 1);
  g.addEdge(0, 2);
  g.addEdge(1, 2);
  g.addEdge(2, 0);
  g.addEdge(2, 3);

  g.toString();
}
```

Graph Representation: Adjacency List

Adjacency List

An adjacency list represents a graph as an array of linked lists.

The index of the array represents a vertex and each element in its linked list represents the other vertices that form an edge with the vertex.


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

Graph Representation: Adjacency List

Pros of Adjacency List

- An adjacency list is efficient in terms of storage because we only need to store the values for the edges. For a sparse graph with millions of vertices and edges, this can mean a lot of saved space.
- It also helps to find all the vertices adjacent to a vertex easily.

Graph Representation: Adjacency List

Cons of Adjacency List

- Finding the adjacent list is not quicker than the adjacency matrix because all the connected nodes must be first explored to find them.

Graph Operations

The most common graph operations are:

- Check if the element is present in the graph
- Graph Traversal
- Add elements(vertex, edges) to graph
- Finding the path from one vertex to another

Credits and Acknowledgements

https://www.gatevidyalay.com

https://www.programiz.com/