**GRAPHS**

A graph is a pictorial representation of a set of objects where some pairs of objects are connected by links. The interconnected objects are represented by points termed as **vertices**, and the links that connect the vertices are called **edges**.

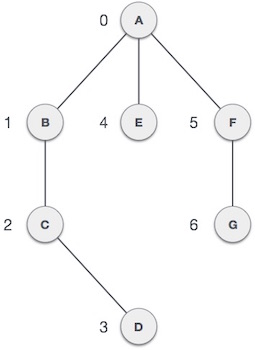
Formally, a graph is a pair of sets **(V, E)**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices. Take a look at the following graph −



**Graph Data Structure**

Mathematical graphs can be represented in data structure. We can represent a graph using an array of vertices and a two-dimensional array of edges. Before we proceed further, let's familiarize ourselves with some important terms −

* **Vertex** − Each node of the graph is represented as a vertex. In the following example, the labelled circle represents vertices. Thus, A to G are vertices. We can represent them using an array as shown in the following image. Here A can be identified by index 0. B can be identified using index 1 and so on.
* **Edge** − Edge represents a path between two vertices or a line between two vertices. In the following example, the lines from A to B, B to C, and so on represents edges. We can use a two-dimensional array to represent an array as shown in the following image. Here AB can be represented as 1 at row 0, column 1, BC as 1 at row 1, column 2 and so on, keeping other combinations as 0.
* **Adjacency** − Two node or vertices are adjacent if they are connected to each other through an edge. In the following example, B is adjacent to A, C is adjacent to B, and so on.
* **Path** − Path represents a sequence of edges between the two vertices. In the following example, ABCD represents a path from A to D.



**Basic Operations**

Following are basic primary operations of a Graph −

* **Add Vertex** − Adds a vertex to the graph.
* **Add Edge** − Adds an edge between the two vertices of the graph.
* **Display Vertex** − Displays a vertex of the graph.

# **Graph Representations**

In graph theory, a graph representation is a technique to store graph into the memory of computer.

To represent a graph, we just need the set of vertices, and for each vertex the neighbors of the vertex (vertices which is directly connected to it by an edge). If it is a weighted graph, then the weight will be associated with each edge.

There are different ways to optimally represent a graph, depending on the density of its edges, type of operations to be performed and ease of use.

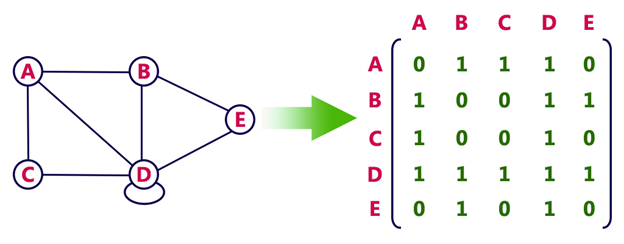
1. **Adjacency Matrix**

* Adjacency matrix is a sequential representation.
* It is used to represent which nodes are adjacent to each other. i.e. is there any edge connecting nodes to a graph.
* In this representation, we have to construct a nXn matrix A. If there is any edge from a vertex i to vertex j, then the corresponding element of A, ai,j = 1, otherwise ai,j= 0.
* If there is any weighted graph then instead of 1s and 0s, we can store the weight of the edge.

### **Example**

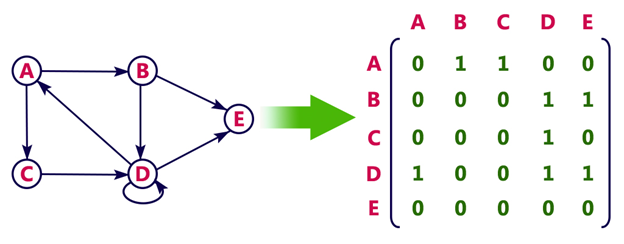
Consider the following **undirected graph representation**:

**Undirected graph representation**



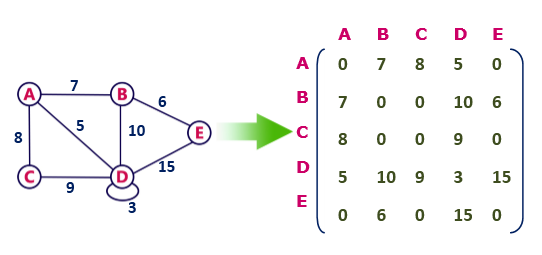
**Directed graph Represenation**

See the directed graph representation:



In the above examples, 1 represents an edge from row vertex to column vertex, and 0 represents no edge from row vertex to column vertex.

**Undirected weighted graph representation**



**Pros:** Representation is easier to implement and follow.

**Cons:** It takes a lot of space and time to visit all the neighbors of a vertex, we have to traverse all the vertices in the graph, which takes quite some time.

## 2. Incidence Matrix

In **Incidence matrix representation**, graph can be represented using a matrix of size:

Total number of vertices by total number of edges.

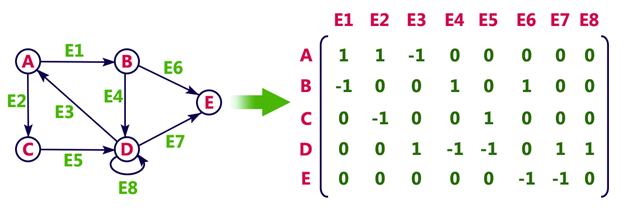
It means if a graph has 4 vertices and 6 edges, then it can be represented using a matrix of 4X6 class. In this matrix, columns represent edges and rows represent vertices.

This matrix is filled with either **0 or 1** or -1. Where,

* 0 is used to represent row edge which is not connected to column vertex.
* 1 is used to represent row edge which is connected as outgoing edge to column vertex.
* -1 is used to represent row edge which is connected as incoming edge to column vertex.

### **Example**

Consider the following directed graph representation.

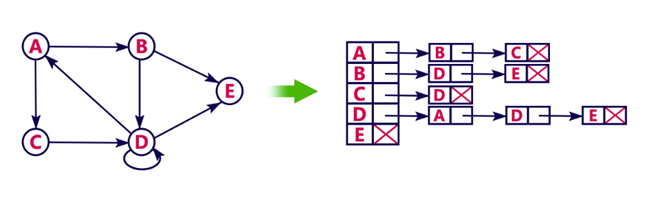


## Adjacency List

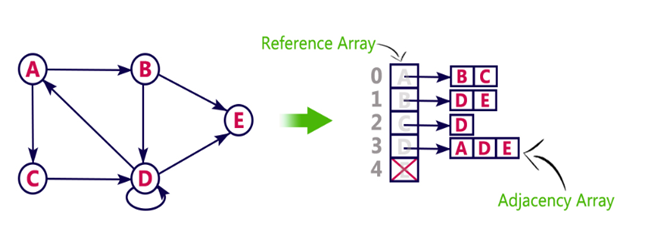
* Adjacency list is a linked representation.
* In this representation, for each vertex in the graph, we maintain the list of its neighbors. It means, every vertex of the graph contains list of its adjacent vertices.
* We have an array of vertices which is indexed by the vertex number and for each vertex v, the corresponding array element points to a **singly linked list** of neighbors of v.

### **Example**

Let's see the following directed graph representation implemented using linked list:



We can also implement this representation using array as follows:



**Pros:**

* Adjacency list saves lot of space.
* We can easily insert or delete as we use linked list.
* Such kind of representation is easy to follow and clearly shows the adjacent nodes of node.

**Cons:**

* The adjacency list allows testing whether two vertices are adjacent to each other but it is slower to support this operation.

# **Graph Traversal Algorithm**

In this part of the tutorial we will discuss the techniques by using which, we can traverse all the vertices of the graph.

Traversing the graph means examining all the nodes and vertices of the graph. There are two standard methods by using which, we can traverse the graphs. Lets discuss each one of them in detail.

* Breadth First Search
* Depth First Search

**Depth First Search (DFS) Algorithm**

Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.

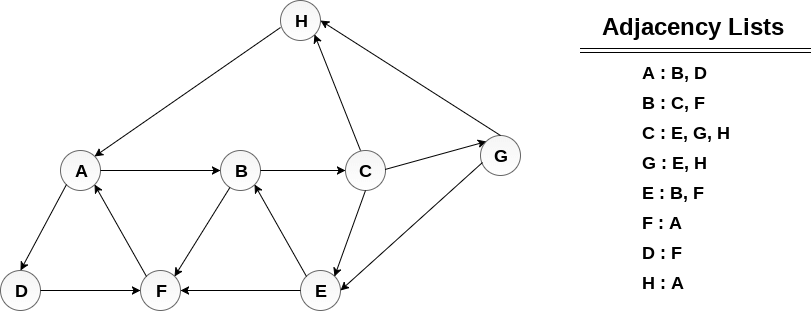
The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

## Algorithm

* **Step 1:** SET STATUS = 1 (ready state) for each node in G
* **Step 2:** Push the starting node A on the stack and set its STATUS = 2 (waiting state)
* **Step 3:** Repeat Steps 4 and 5 until STACK is empty
* **Step 4:** Pop the top node N. Process it and set its STATUS = 3 (processed state)
* **Step 5:** Push on the stack all the neighbours of N that are in the ready state (whose STATUS = 1) and set their  
  STATUS = 2 (waiting state)  
  [END OF LOOP]
* **Step 6:** EXIT

### **Example :**

Consider the graph G along with its adjacency list, given in the figure below. Calculate the order to print all the nodes of the graph starting from node H, by using depth first search (DFS) algorithm.



### **Solution :**

Push H onto the stack

STACK : H

POP the top element of the stack i.e. H, print it and push all the neighbours of H onto the stack that are is ready state.

Print H

STACK : A

Pop the top element of the stack i.e. A, print it and push all the neighbours of A onto the stack that are in ready state.

Print A

Stack : B, D

Pop the top element of the stack i.e. D, print it and push all the neighbours of D onto the stack that are in ready state.

Print D

Stack : B, F

Pop the top element of the stack i.e. F, print it and push all the neighbours of F onto the stack that are in ready state.

Print F

Stack : B

Pop the top of the stack i.e. B and push all the neighbours

Print B

Stack : C

Pop the top of the stack i.e. C and push all the neighbours.

Print C

Stack : E, G

Pop the top of the stack i.e. G and push all its neighbours.

Print G

Stack : E

Pop the top of the stack i.e. E and push all its neighbours.

Print E

Stack :

Hence, the stack now becomes empty and all the nodes of the graph have been traversed.

The printing sequence of the graph will be :

H → A → D → F → B → C → G → E

**RELEVANT READING MATERIAL AND REFERENCES:**

**Source Notes:**

1. <https://www.tutorialspoint.com/data_structures_algorithms/graph_data_structure.htm>
2. <https://www.javatpoint.com/graph-theory-graph-representations>
3. <https://www.javatpoint.com/depth-first-search-algorithm>

**Lecture Video:**

* 1. <https://www.youtube.com/watch?v=5hPfm_uqXmw>
  2. <https://www.youtube.com/watch?v=1n5XPFcvxds&list=PLqM7alHXFySEaZgcg7uRYJFBnYMLti-nh>

**Online Notes:**

1. <http://www.crectirupati.com/sites/default/files/lecture_notes/ds%20ln.pdf>
2. <http://www.vssut.ac.in/lecture_notes/lecture1428550942.pdf>

**Text Book Reading:**

1. Cormen, Leiserson, Rivest, Stein, “*Introduction to Algorithms*”, Prentice Hall of India, 3rd edition 2012. problem, Graph coloring.
2. Lipschutz, S., “*Data Structures, Schaum's Outline Series*”, Tata McGraw Hill.

**Online Book Reference:**

1. <https://www.edutechlearners.com/download/books/DS.pdf>

**In addition: PPT can be also be given.**