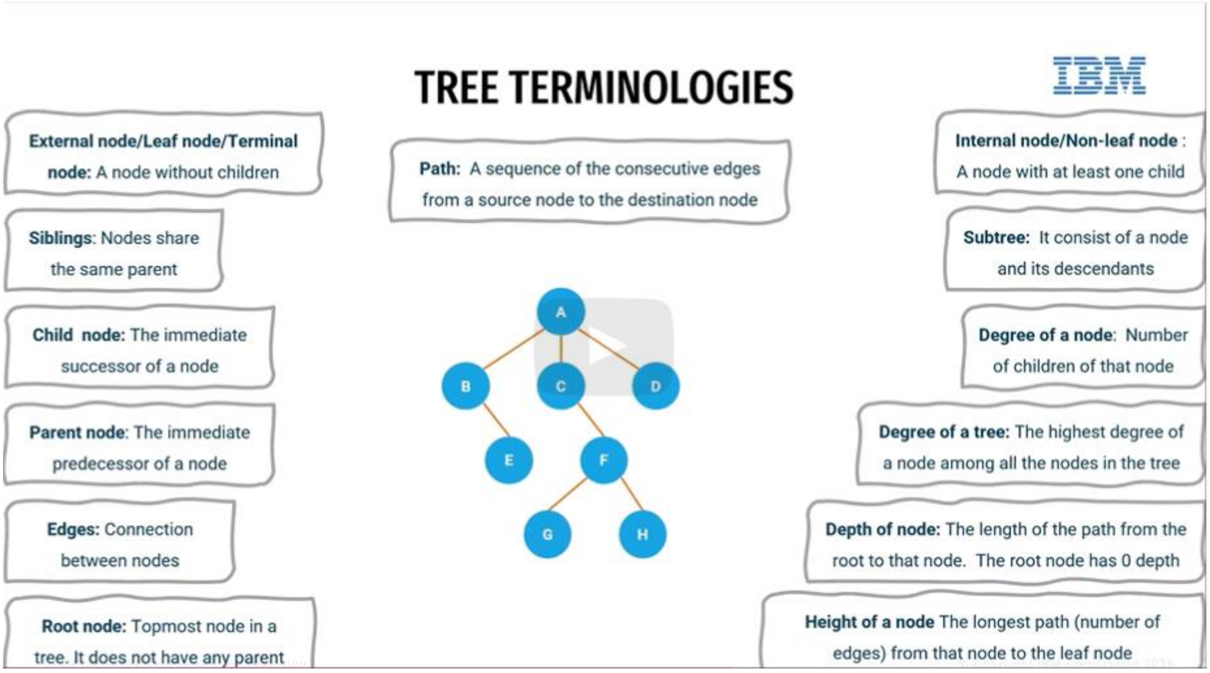
Non-linear Data Structure: Tree and Graph:

Tree is a non-linear data structure consists of a collection of nodes. Each node stores a value and a list of references to its child nodes.

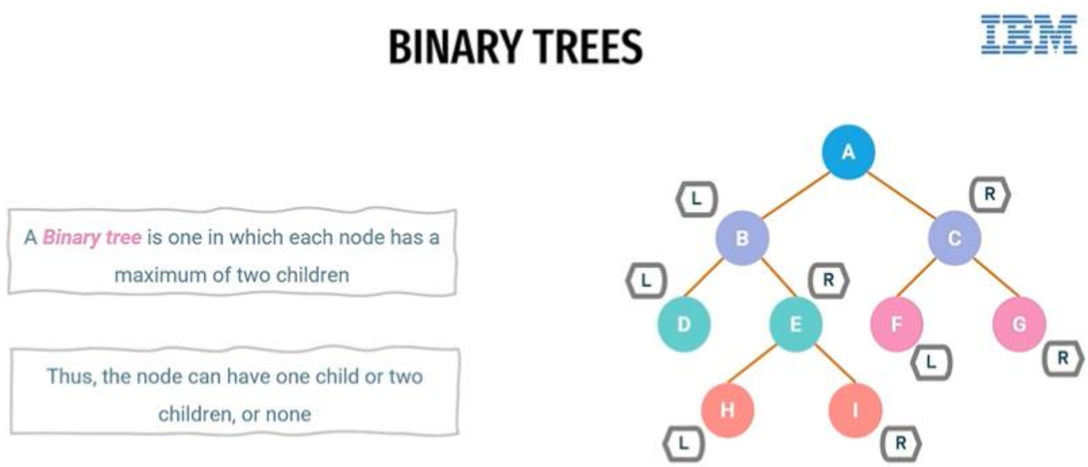
Tree Terminology:

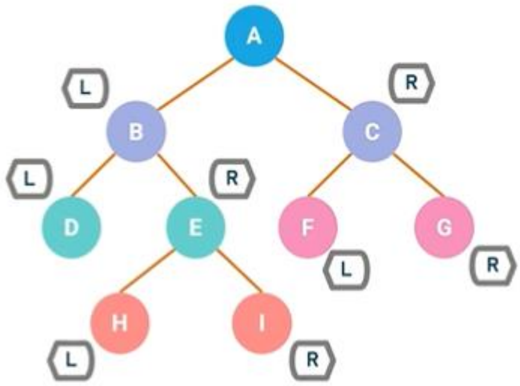


Application of Tree Data Structure

1. To store large volumes of data
2. Used in compilers
3. To implement Database Management Systems
4. Used in OS file systems

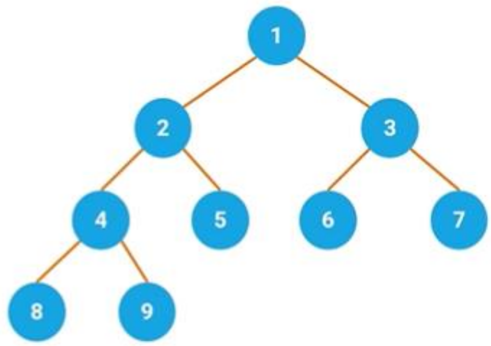
Binary Tree:



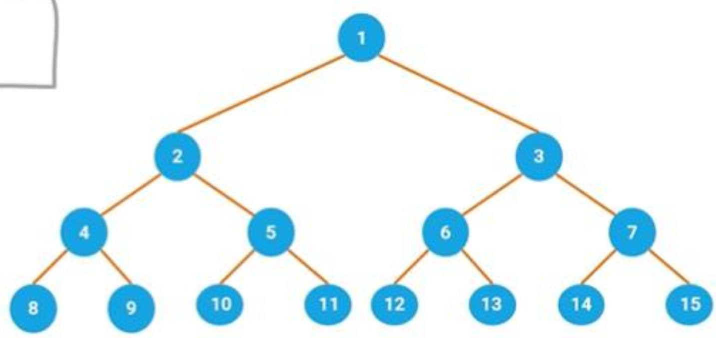


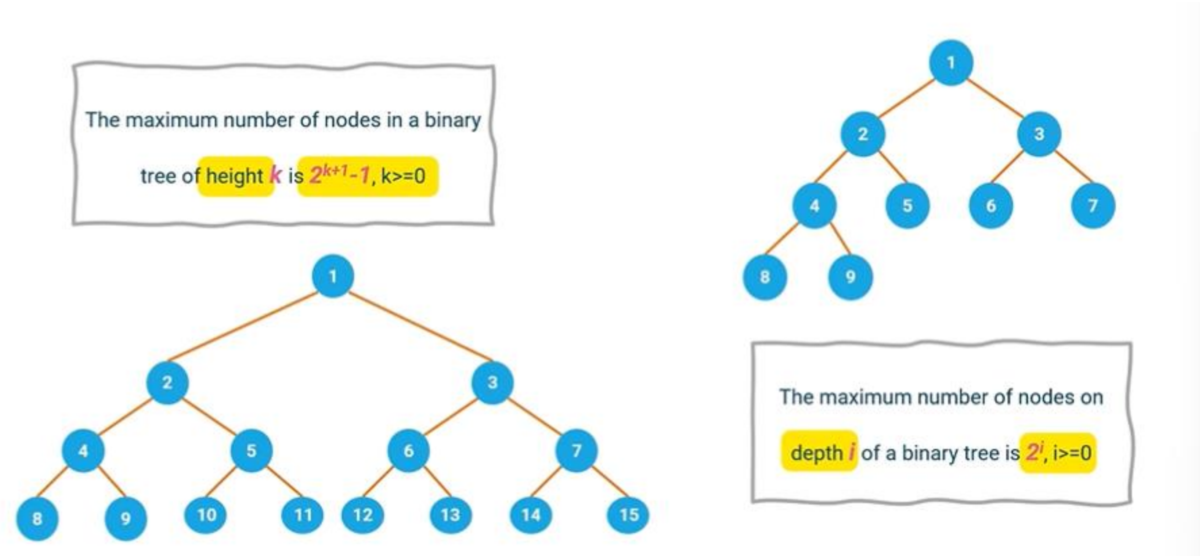
Binary Tree Variations

In a *Complete Binary Tree*, every level is completely filled, except possibly the last, and all nodes are to the far left as much as possible

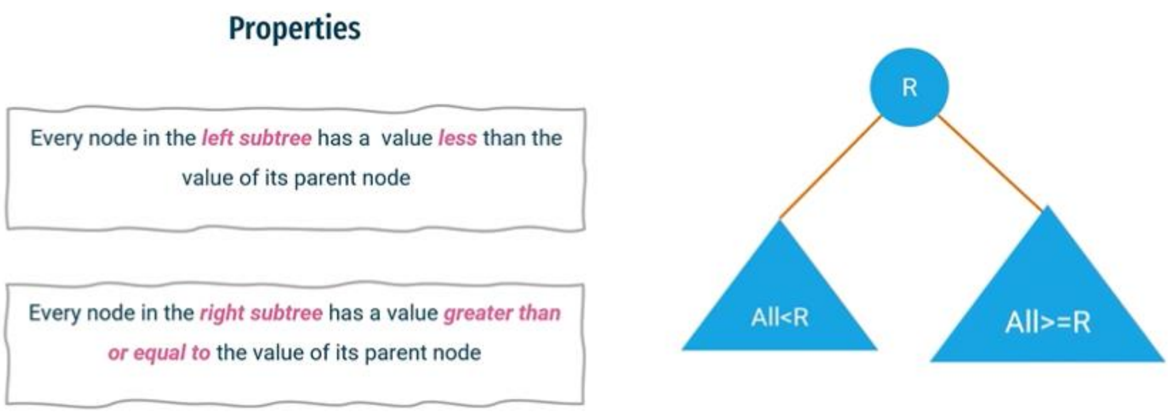


In a *Fully Binary Tree*, every node other than the leaves has two children





Binary Search Tree:



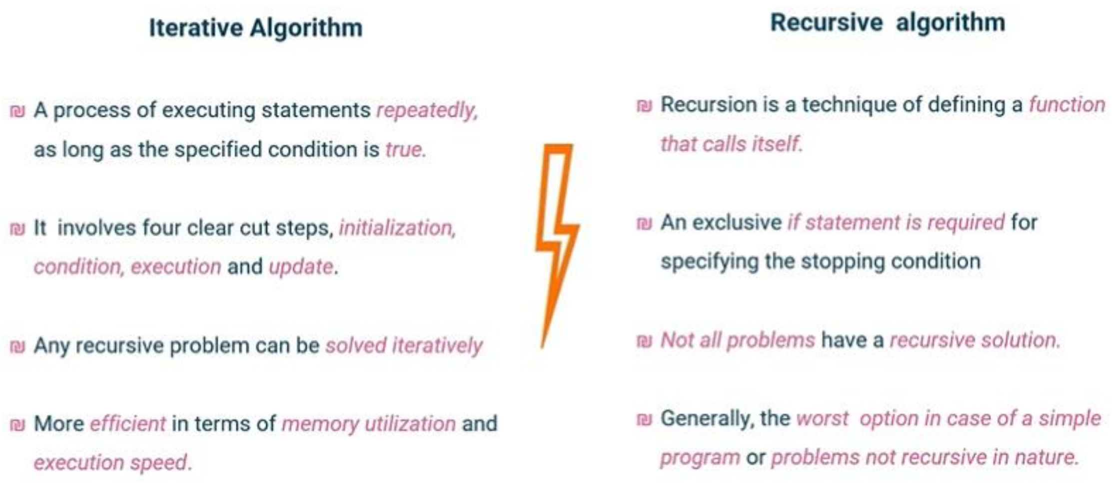
Recursion:

Key components are:

1. **Composition** - Smaller to Bigger problems.
2. **Decomposition** - Bigger to Smaller problems.

3. **Base/stop case** - The stop case.

Iteration vs Recursion

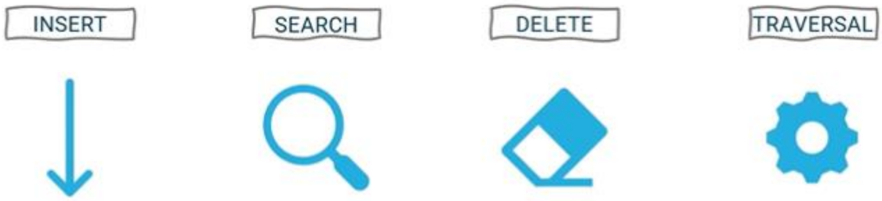


Summary: Recursion

- Recursion is a technique that solves a problem by solving a smaller problem of the same type.
- The key components of a recursive algorithm are: 1. **Decomposition** 2. **Composition** and 3. **Base condition**.
- Solving a problem using the smaller version of the same problem is known as **decomposition.**
- Combining the answers of the smaller problems to form the answer to the larger problem is known as **composition.**
- The smallest problem that can be solved without further decomposing it is known as the **base** or **stopping case.**
- Recursive algorithms solve difficult problems easily and provide readability.
- Recursive algorithms require a lot of memory and computation compared to iterative algorithms.

BST (Binary Search Tree)

MAJOR OPERATIONS



Pre-order traversal

In-order traversal

Post-order traversal

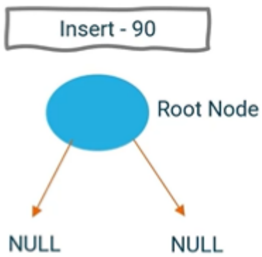
Implementation:

Tree node Implementation

```
class TreeNode {
    int data;
    TreeNode *left, *right;
};
```

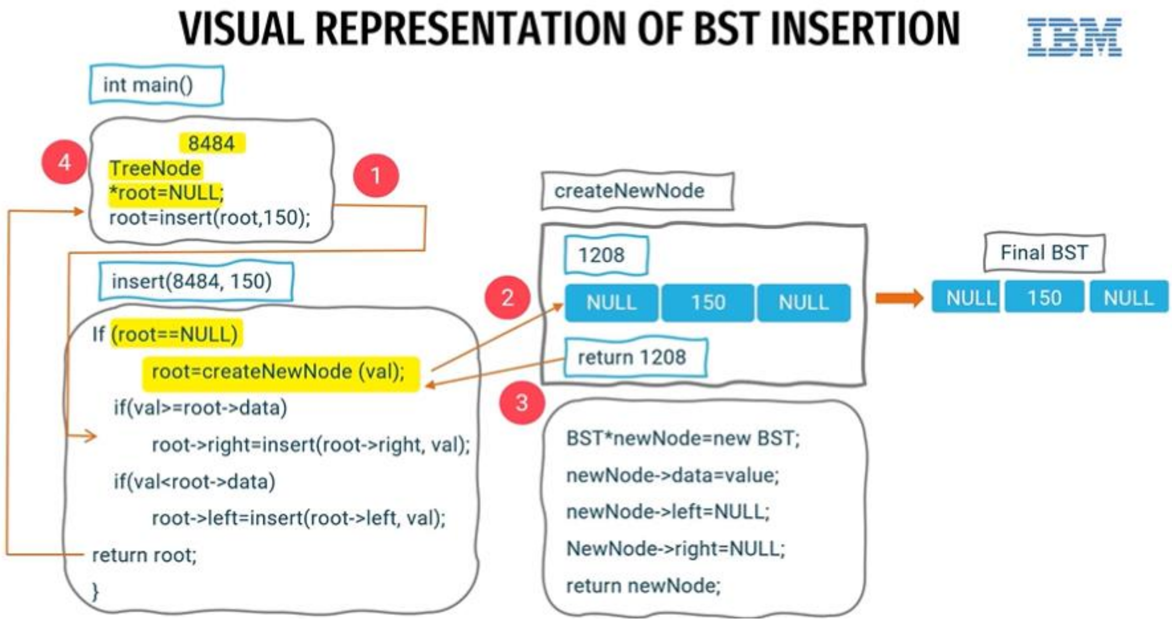
POST ORDER TRAVERSAL ALGORITHM

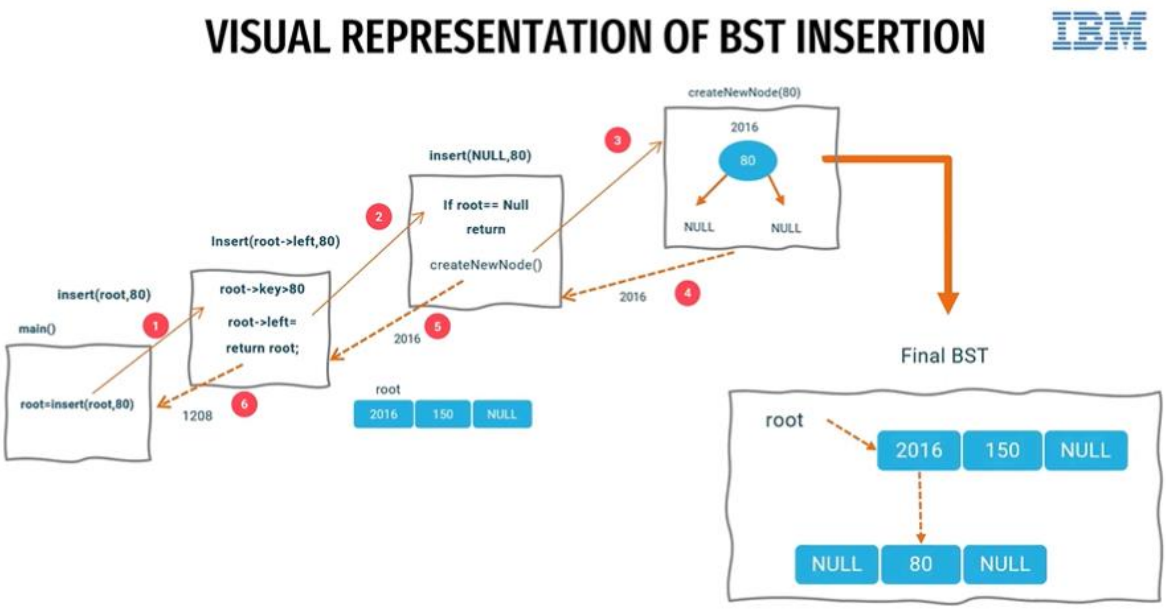
```
insert(root,value) {
    if root == NULL then

    Allocate the memory for the new node
        set the value to root->data ;
        Set root->left   = NULL;
        Set root->right   = NULL;
        return root;

    else if  value < root->data then
        set root->left = insert(root ->left ,value);
        (Node needs to be inserted in left sub-tree. So,
         recursively traverse left sub-tree to find the place
         where the new node needs to be inserted)
```



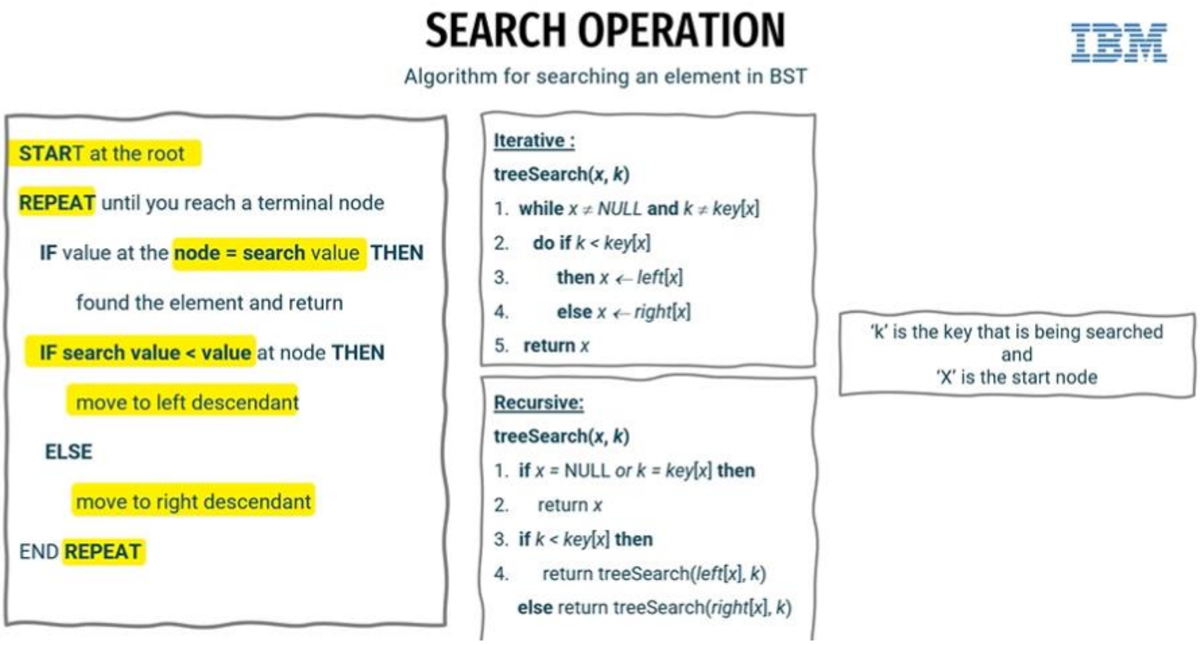
```
    else if value >=  root->data then
            set root->right== insert(root->right,value);
             (Node needs to be inserted in right sub-tree
              So, recursively traverse right sub-tree to find the
               place where the new node needs to be inserted)
    return root;
}
```

Visual Representation of BST:

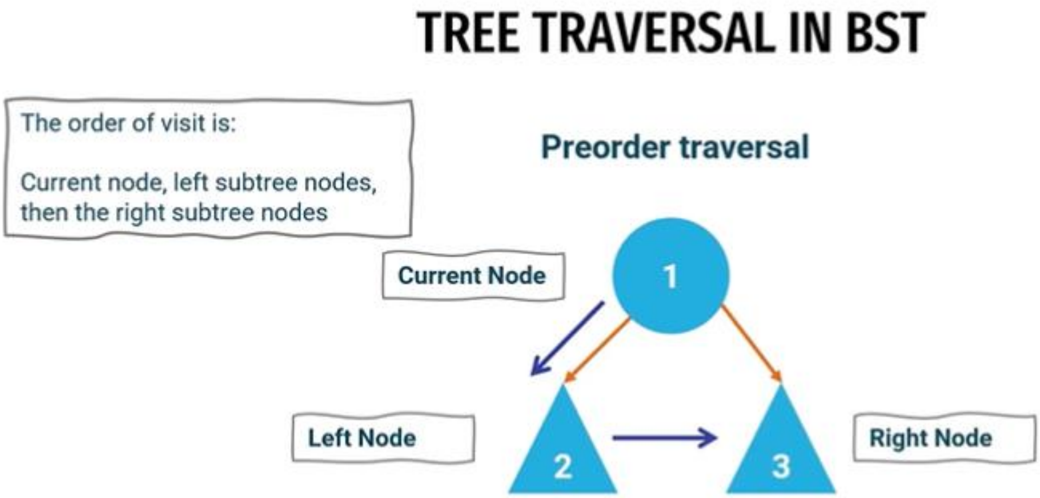


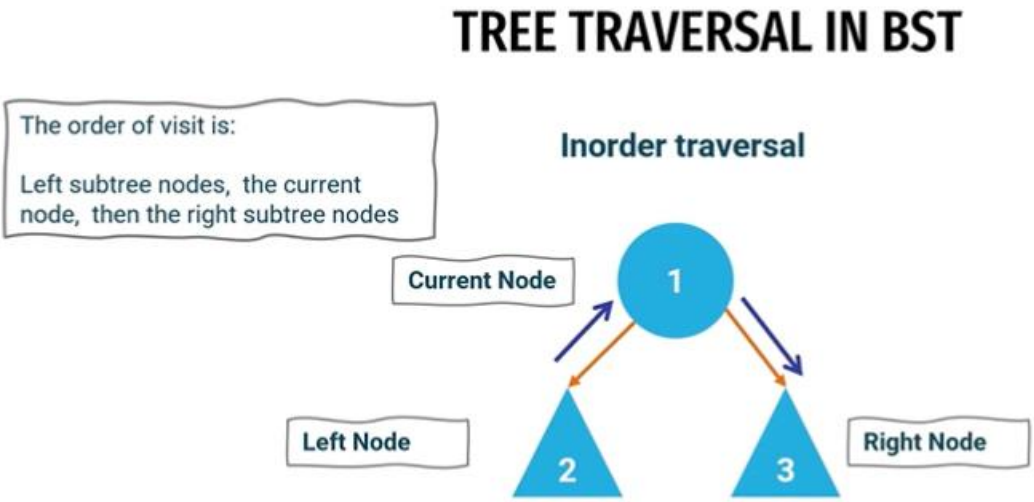


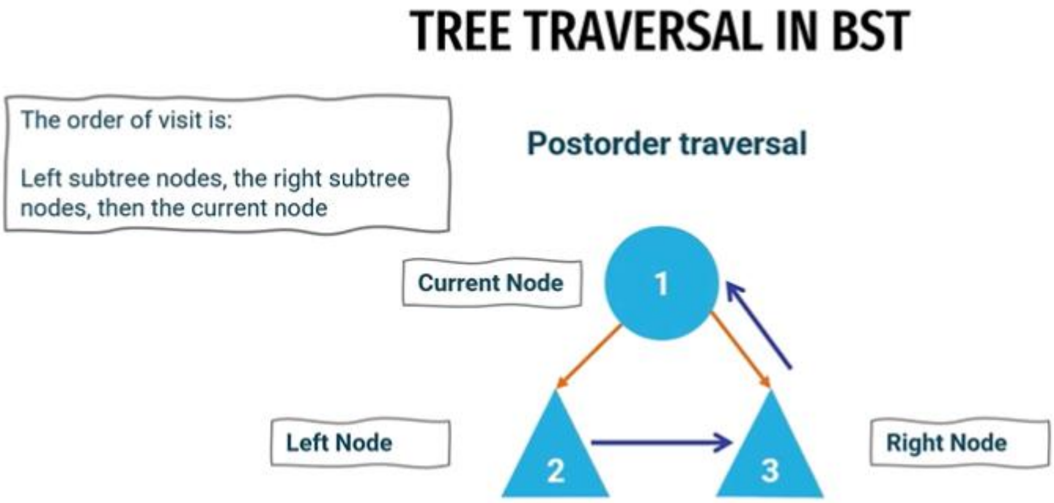
Search Operation:



Tree Traversal in BST:




TREE TRAVERSAL IN BST
The order of visit is:
Left subtree nodes, the current node, then the right subtree nodes
Inorder traversal
Current Node
1
Left Node
2
3
Right Node

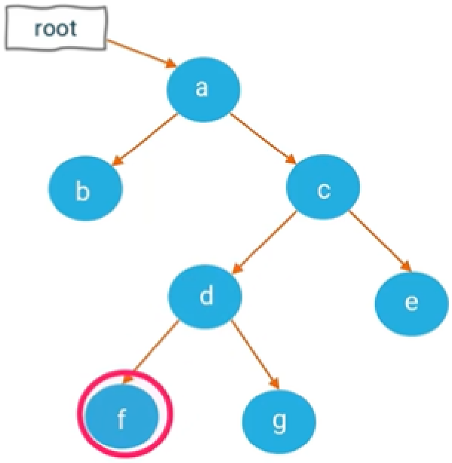

TREE TRAVERSAL IN BST
The order of visit is:
Left subtree nodes, the right subtree nodes, then the current node
Postorder traversal
Current Node
1
Left Node
2
3
Right Node

Algorithms:

PREORDER TRAVERSAL ALGORITHM

```
preOrder(x)
  Input: x is the root of a subtree
  if x!=NULL then
        output key(x)
        preOrder(left(x));
        preOrder(right,x);
```

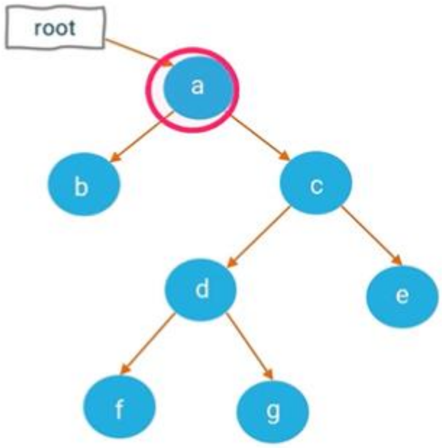
Preorder traversal for the above tree is: a b c d f g e

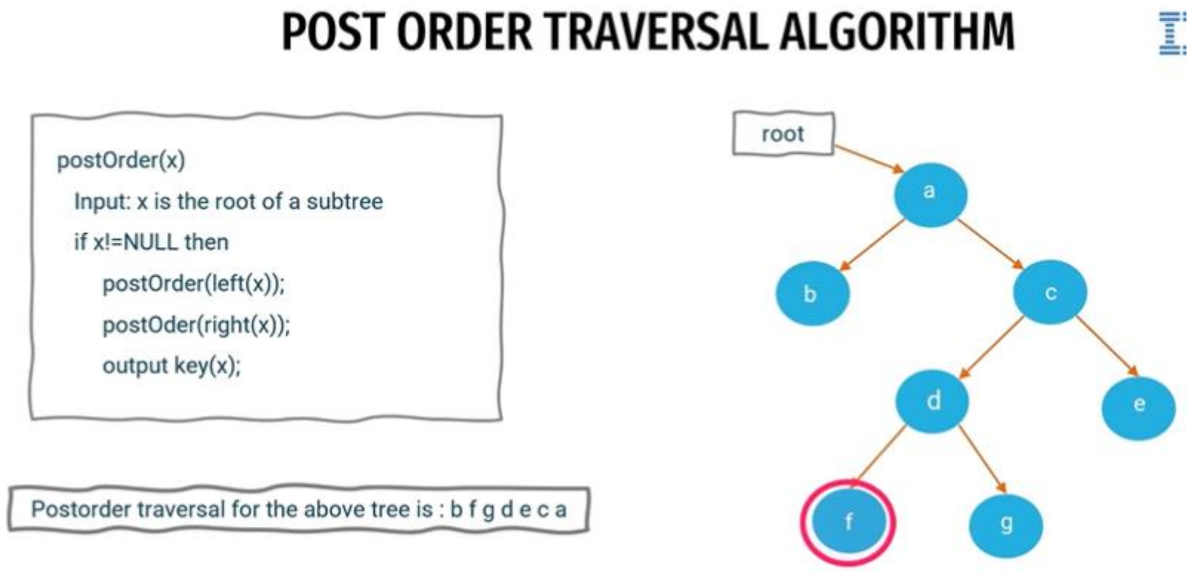


INORDER TRAVERSAL ALGORITHM

```
inOrder(x)
  Input: x is the root of a subtree
  if x!=NULL then
        inOrder(left(x));
        output key(x);
        inOder(right(x));
```

Inorder traversal for the above tree is : b a f d g c e





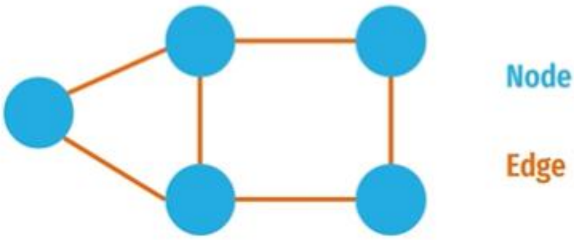
Summary: Tree

- In a non-linear data structure, the data elements are arranged in non-sequential order.
- A tree data structure represents a non-linear hierarchical relationship between the data elements.
- In a binary tree, each node has at the most two children, which are referred to as the left child and the right child.
- In a complete binary tree, every level except possibly the last is filled, and all the nodes are ion the far left as possible.
- A full binary tree in a tree in which every node other than the leaves has two children.
- A Binary Search Tree has two main properties:
    - Every node in the left subtree has a value less than the value of its parent node.
    - Every node in the right subtree has a value greater then or equal to the value of its parent node.
- In preorder traversal, the order of visit is current node, left subtree nodes, then the right subtree nodes.
- In in-order traversal, the order of visit is left subtree nodes, current node, then the right subtree nodes.
- In post-order, the order of visit is left subtree nodes, the right subtree nodes, then the current node.

GRAPH:

WHAT IS GRAPH?

A graph is a mathematical structure used for representing, finding, analyzing, and optimizing connections between elements (nodes)



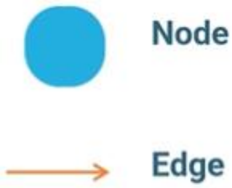
GRAPH - DEFINED

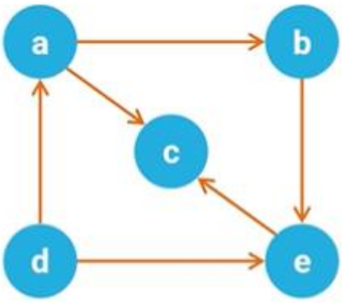
A graph G = (V,E) is composed of:

V: Set of vertices (Nodes)

E: Set of edges connecting the vertices in V

An edge e = (u,v) is pair of vertices





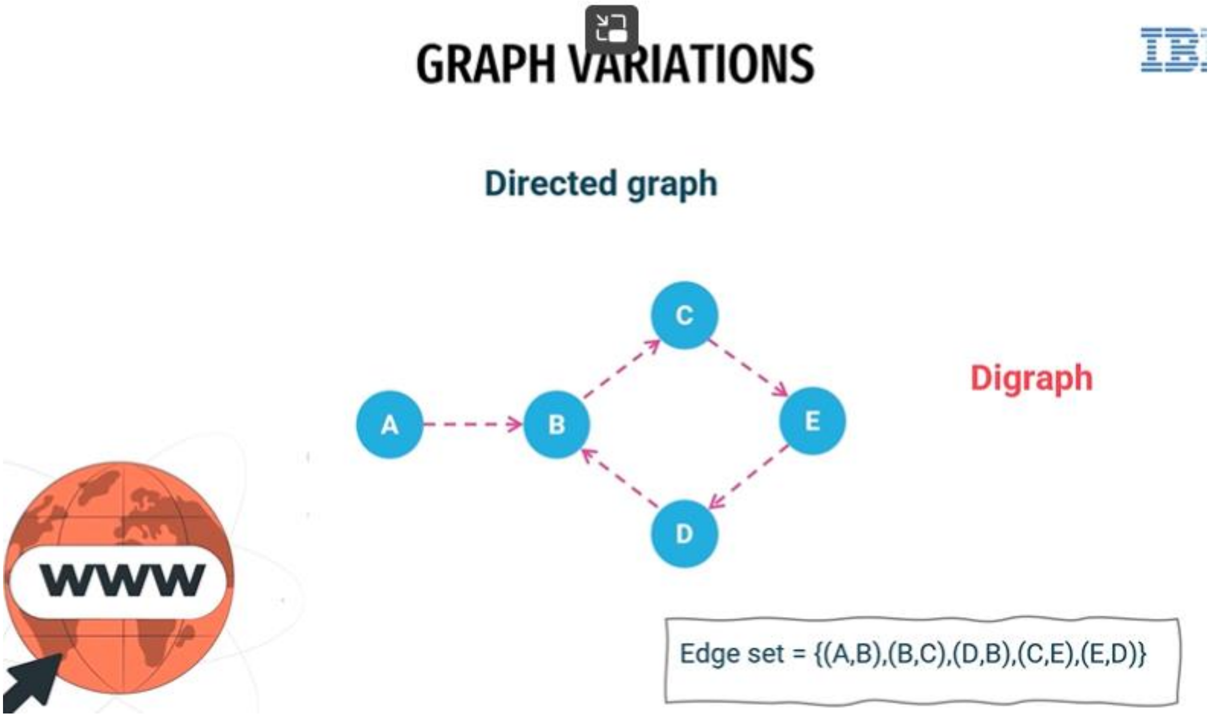
V={a,b,c,d,e}

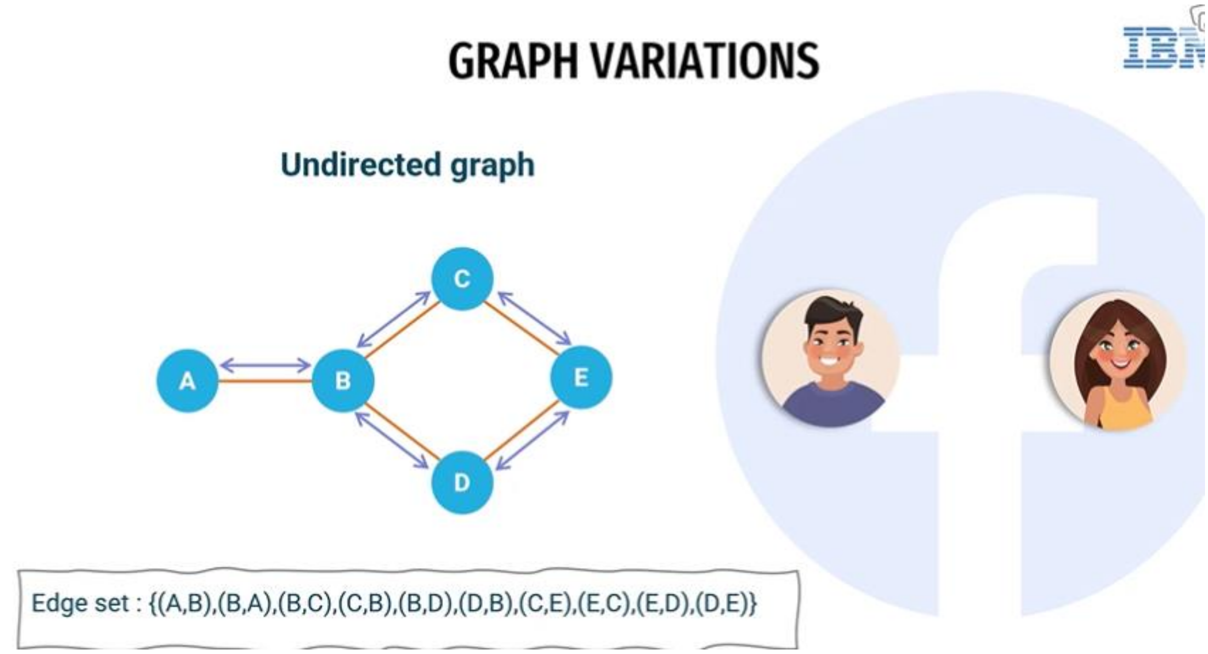
E={(a,b),(a,c),(b,e),(e,c),(d,e),(d,a)}

Applications of Graph:

- GPS Systems
- Google Maps
- Social Networks (Facebook's friend suggestion algorithm)
- Product Recommendation
- WWW / Google Search
- Operations Research
- Travelling Salesman Problem (TSP)

Types of Graphs:





Weighted Graph

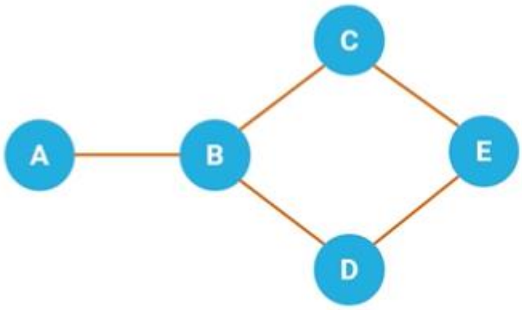
Some weight or value is given to the edges

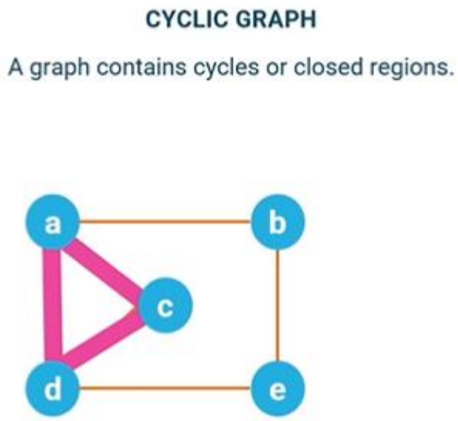
Cost of Travel

Weighted Graph

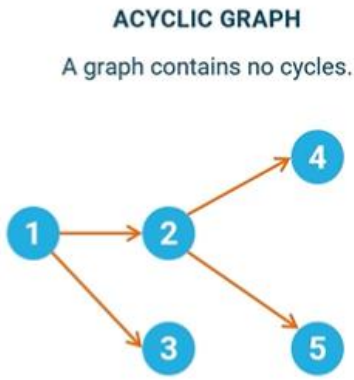
Unweighted Graph

Edges have no weight





a c d a



An Acyclic Directed Graph is a Tree

Graph Terminology:

Term	Description
Vertex	Every individual data element is called a vertex or a node.
Edge (Arc)	A connecting link between two nodes or vertices.
Undirected Edge	It is a bidirectional edge.
Directed Edge	It is a unidirectional edge.
Weighted Edge	An edge with value (cost) on it.
Degree	The total number of edges connected to a vertex in a graph.
Indegree	The total number of incoming edges connected to a vertex.
Outdegree	The total number of outgoing edges connected to a vertex.

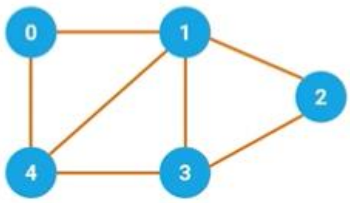
Graph Representation:

1. Adjacency Matrix
2. Adjacency List
3. Incidence Matrix
4. Incidence List

Adjacency Matrix

A 2D array of size V x V. V is the number of vertices in a graph.

A slot graph[i][j] = 1 indicates that there is an edge from vertex i to vertex j .



Adjacency matrix for the given graph:

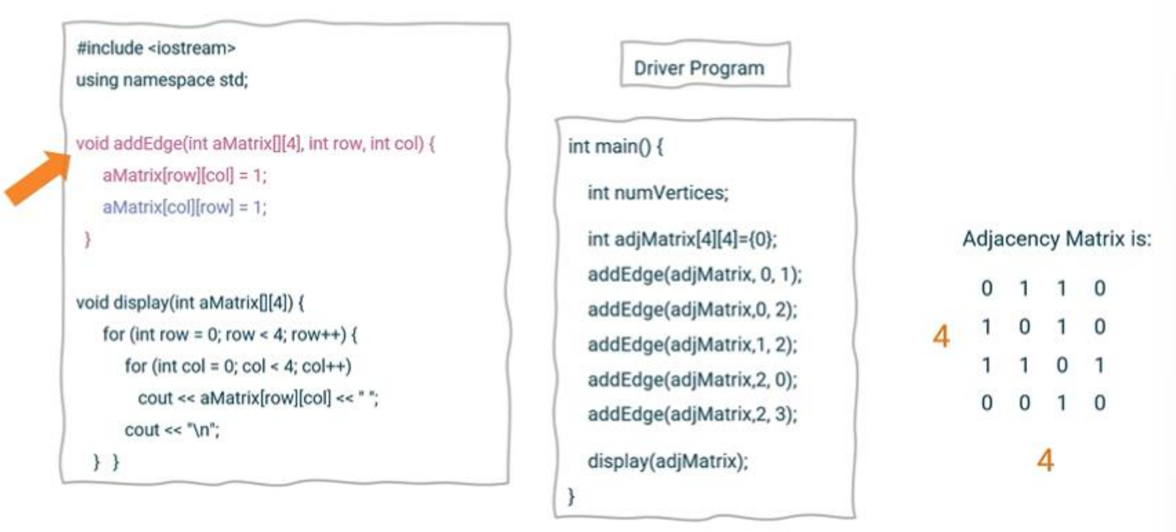
	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0

Adjacency matrix for an undirected graph is always symmetric

Adjacency Matrix is also used to represent weighted graphs

If graph[i][j] = w, then there is an edge from vertex i to vertex j with weight w

Graph Implementation: (Adjacency Matrix)



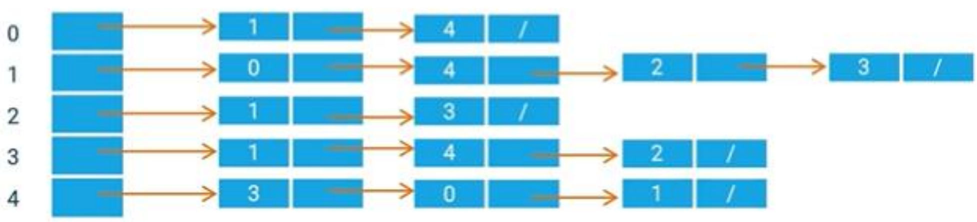
Adjacency List

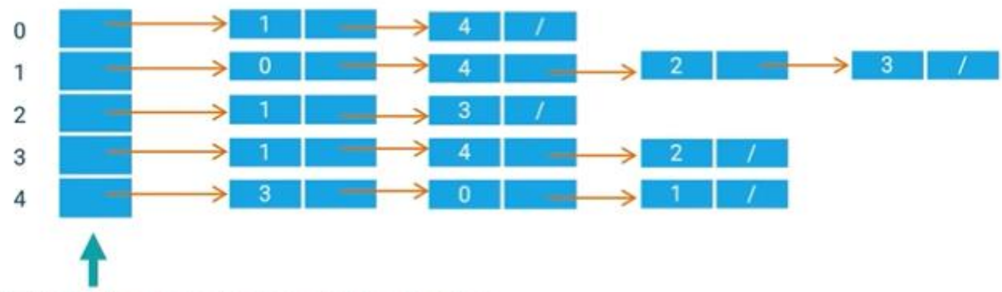
A linked list version of the adjacency table. An array of linked lists is used. The size of the array is equal to the number of vertices.

Let the array be array[5].

An entry in array[i] represents the linked list of vertices adjacent to the ith vertex.

Adjacency list representation of the given graph:





Array of linked lists, where list nodes store labels for neighbours

This representation can also be used to represent a weighted graph.

The weights of edges can be stored in nodes of linked lists.

Graph Implementation: (Adjacency List)

USING ADJACENCY LIST:

```
#include <iostream>
using namespace std;

class Node {
   public:
       int vertex;
       int weight;
       Node* next;
 };

void create(Node* head[]) {
  char ch = 'y';
  int v1, v2, choice,no,weight;
  Node* newNode;
  Node* temp;
  cout<<"0 – Directed Graph\n";
  cout<<"1 – Undirected Graph\n";
  cout<<"Enter Your Choice (0 or 1):\n";
  cin>>choice;
  cout<<"Enter the no. of edges:\n";
 cin>>no;
```

```
for(int i=0;i<no;i++){
  cout<<"\n Enter the starting node,
          ending node and weight:\n";
  cin>>v1;
  cin>>v2;
  cin>>weight;

  newNode = new Node();
  newNode->vertex = v2;
  newNode->weight=weight;
  temp = head[v1];
 if(temp == NULL)  {
    head[v1] = newNode;
  }
  else  {
    while(temp->next != NULL)  {
       temp = temp->next;
    }
    temp->next = newNode;
}
```

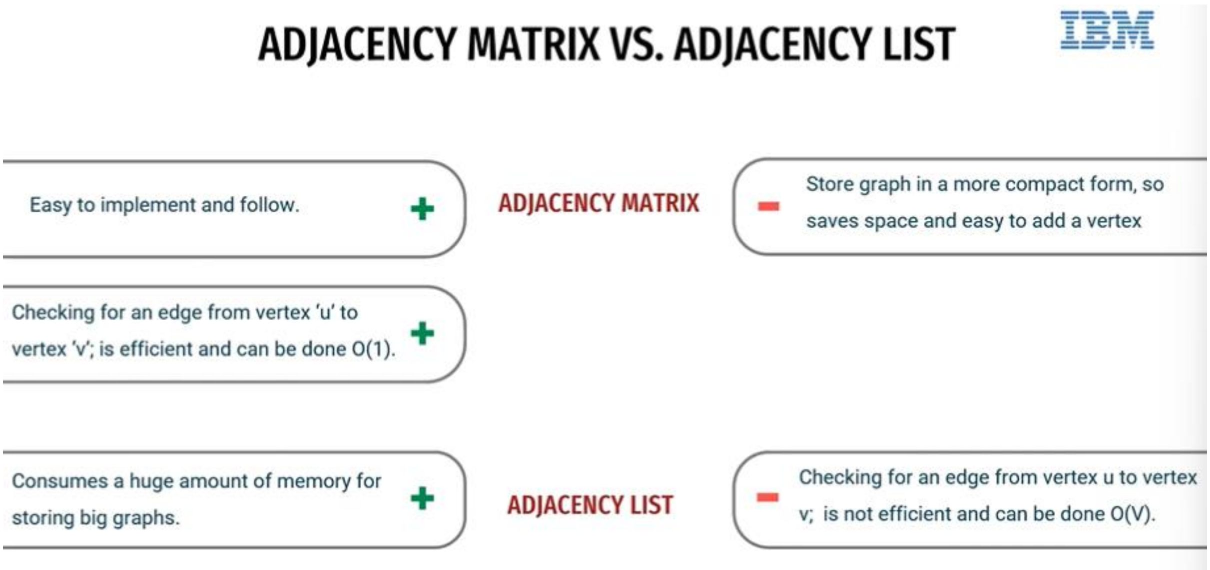
```
if(choice == 1) {
   newNode = new Node();
   newNode->vertex = v1;
   newNode->weight=weight;
   temp = head[v2];
   if(temp == NULL) {
      head[v2] = newNode;
   }
   else  {
      while(temp->next != NULL) {
          temp = temp->next;
      }
      temp->next = newNode;
    }
  }
 }
}
```

```
void display(Node* head[], int n) {
   int v;
   Node* adj;
   cout<<"Adjacency List Is:\n";
   for(v = 0; v < n; v++){
     cout<<"Head["<<v<<"]";
     adj = head[v];
     while(adj != NULL) {
        cout<<adj->vertex<<"=>weight:"<<adj->weight<<" ";
        adj = adj->next;
     }
     cout<<"\n";
   }
}
```

Driver Program

```
int main(){
   char c = 'y';
   int ch, start, n, visited[10],v;
   Node* head[50];
    cout<<"No. of vertices in the graph:\n";
    cin>>n;
    for(v = 0; v < n; v++) {
       head[v] = NULL;
    }
    create(head);
    display(head,n);
}
```

Advantages and Disadvantages:



Summary:

- A graph consists of a set of nodes connected by edged and represents relationships.
- In a graph, nodes create the network and the edges provide the connections between one node to another.
- In Un-directed graphs, the edges have no direction whereas directed graphs have a direction.
- In a weighted graph, the edges have a weight whereas in an unweighted graph the edges have no weight.
- A cyclic graph is a graph that contains cycles or closed regions whereas an acyclic graph contains no cycles.
- Commonly used representations of a graph are 1. Adjacency Matrix and 2. Adjacency List.
- Adjacency Matrix is a 2D array of size V x V where V is the number of vertices in a graph.
- An Adjacency List is an array of linked lists. The size of an array of linked list is equal to the number of vertices.