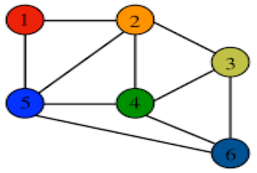
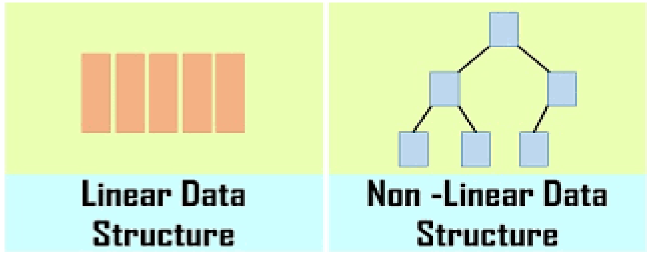
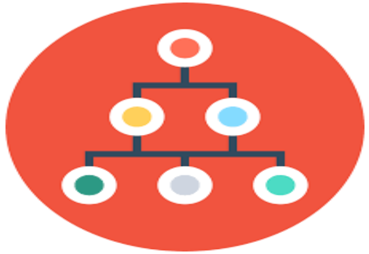
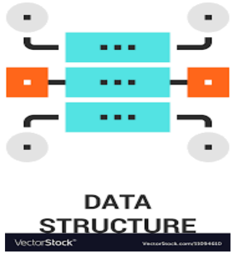
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 3
NON LINEAR DATA STRUCTURE





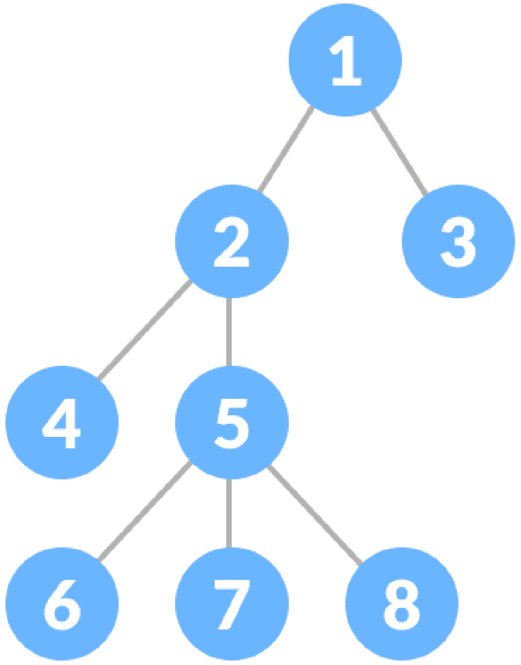




SUBJECT:DATA STRUCTURE CODE:3130702	PREPARED BY: ASST.PROF.PARAS NARKHEDE (CSE DEPARTMENT,ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

TREE DEFINITION & CONCEPT

- A tree is a nonlinear hierarchical data structure that consists of nodes connected by edges.
- Other data structures such as arrays, linked list, stack, and queue are linear data structures that store data sequentially. In order to perform any operation in a linear data structure, the time complexity increases with the increase in the data size. But, it is not acceptable in today's computational world.
- Different tree data structures allow quicker and easier access to the data as it is a non-linear data structure.



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

TREE TERMINOLOGIES

Node

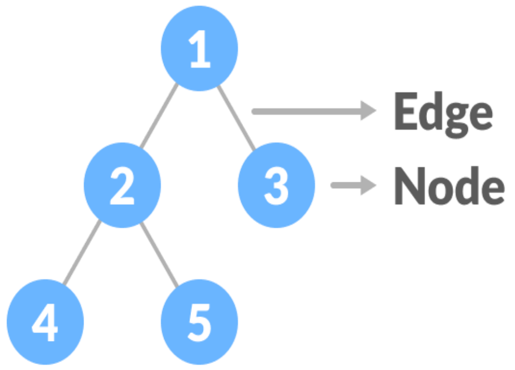
A node is an entity that contains a key or value and pointers to its child nodes.

The last nodes of each path are called leaf nodes or external nodes that do not contain a link/pointer to child nodes.

The node having at least a child node is called an internal node.

Edge

It is the link between any two nodes.



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

TREE TERMINOLOGIES

Root

It is the topmost node of a tree.

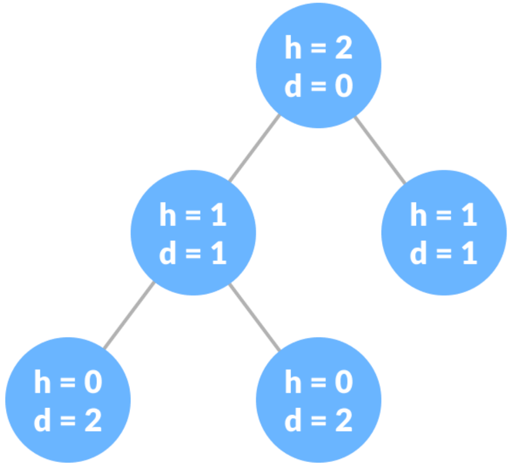
Height of a Node

The height of a node is the number of edges from the node to the deepest leaf (ie. the longest path from the node to a leaf node).

Depth of a Node

The depth of a node is the number of edges from the root to the node.

TREE TERMINOLOGIES

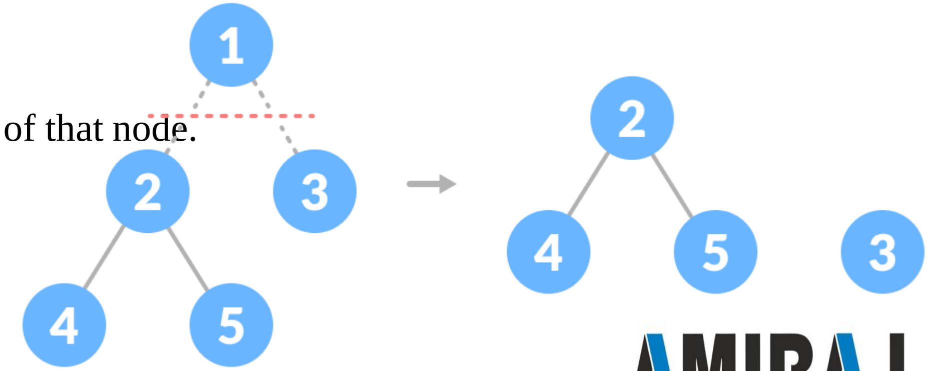


Height of a Tree

The height of a Tree is the height of the root node or the depth of the deepest node.

Degree of a Node

The degree of a node is the total number of branches of that node.



Forest

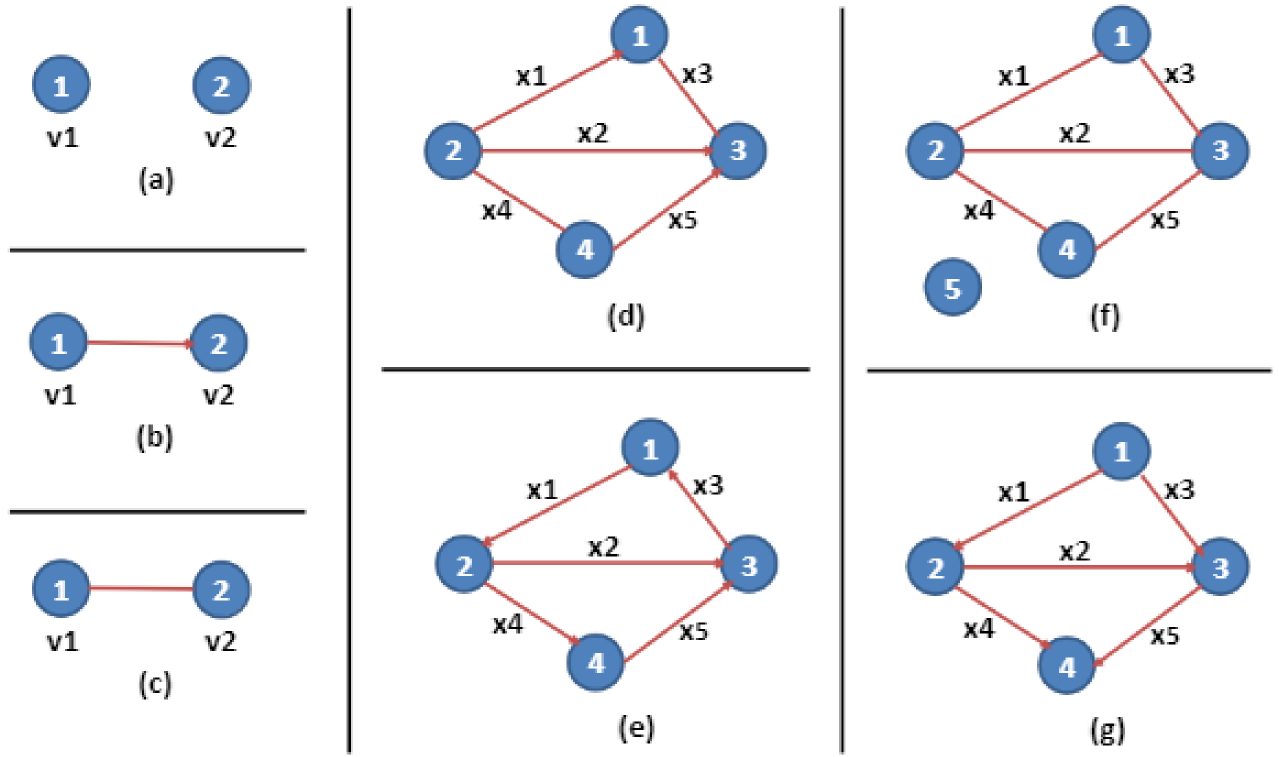
A collection of disjoint trees is called a forest.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

TREE APPLICATIONS

- Binary Search Trees(BSTs) are used to quickly check whether an element is present in a set or not.
- Heap is a kind of tree that is used for heap sort.
- A modified version of a tree called Tries is used in modern routers to store routing information.
- Most popular databases use B-Trees and T-Trees, which are variants of the tree structure we learned above to store their data
- Compilers use a syntax tree to validate the syntax of every program you write.

BASIC NOTATION OF GRAPH THEORY



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BASIC NOTATION OF GRAPH THEORY

- ❖ Consider diagrams shown in above figure
- ❖ Every diagrams represent Graphs
- ❖ Every diagram consists of a set of points which are shown by dots or circles and are sometimes labelled V1, V2, V3… OR 1,2,3…
- ❖ In every diagrams, certain pairs of such points are connected by lines or arcs
- ❖ Note that every arc start at one point and ends at another point

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BASIC NOTATION OF GRAPH THEORY

- **Graph**
  - **A graph G** consist of a **non-empty set V** called the **set of nodes** (points, vertices) of the graph, a **set E** which is the **set of edges** and a **mapping** from the set of edges **E to** a set of **pairs of elements of V**
  - It is also convenient to write a graph as **G=(V,E)**
  - Notice that definition of graph implies that to every edge of a graph G, we can associate a pair of nodes of the graph. If an edge X Є E is thus associated with a pair of nodes (u,v) where u, v Є V then we says that edge x connect u and v
- **Adjacent Nodes**
  - Any two nodes which are connected by an edge in a graph are called adjacent nodes

GRAPH DEFINITIONS

Directed & Undirected Edge

In a graph **G=(V,E)** an **edge** which is **directed** from one end to another end is called a **directed edge**, while the edge which has no specific direction is called **undirected edge**

Directed graph (Digraph)

A graph in which **every edge is directed** is called directed graph or digraph e.g. **b,e & g** are directed graphs

Undirected graph

A graph in which **every edge is undirected** is called undirected graph e.g. **c & f** are undirected graphs

Mixed Graph

If **some** of the **edges** are **directed** and **some are undirected** in graph then the graph is called mixed graph e.g. **d** is mixed graph

Distinct Edges

In case of directed edges, **two possible edges** between any pair of nodes which **are opposite in direction** are considered **Distinct**.

Parallel Edges

In some directed as well as undirected graphs, we may have **certain pairs of nodes joined by more than one edges**, such edges are called Parallel edges

Multigraph

Any **graph** which **contains** some **parallel edges** is called **multigraph**

If there is no more then one edge between a pair of nodes then such a graph is called **Simple graph**

Weighted Graph

A graph in which **weights are assigned to every edge** is called weighted graph

Isolated Node

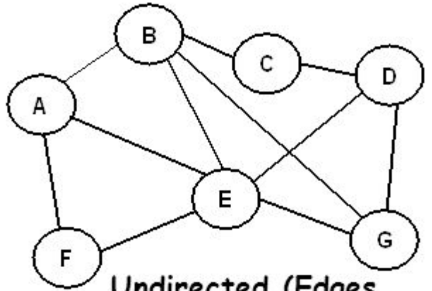
In a graph a **node** which is **not adjacent to any other node** is called isolated node

Null Graph

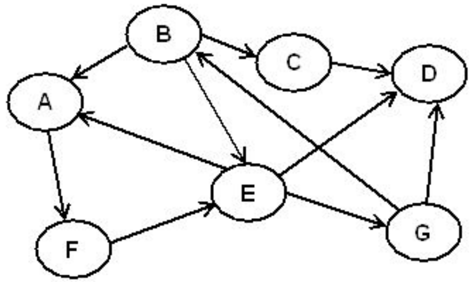
A graph **containing only isolated nodes** are called null graph. In other words set of edges in null graph is empty

Types of Graphs

Directed vs. undirected

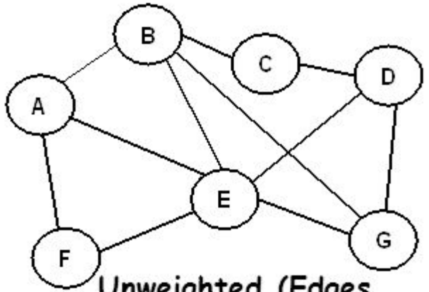


Undirected (Edges have no direction)

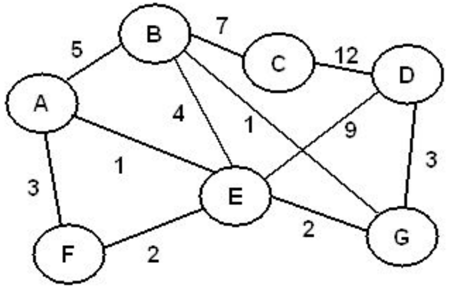


Directed (Edges have directions)

Weighted vs. unweighted



Unweighted (Edges have no cost/weight)



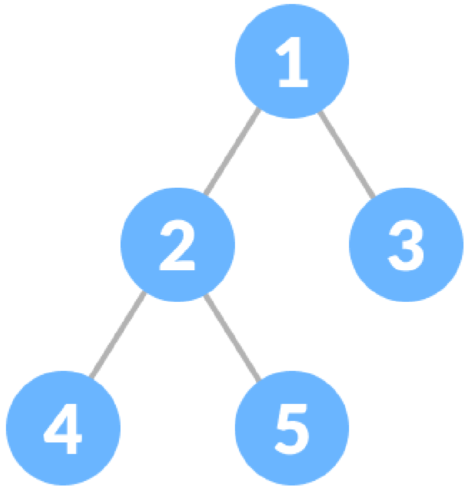
Weighted (Edges have associated cost/weight)

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

REPRESENTATION OF BINARY TREE

A binary tree is a tree data structure in which each parent node can have at most two children.

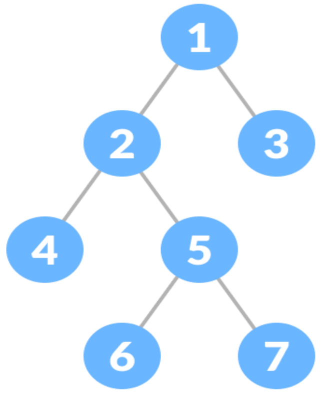
For example: In the image below, each element has at most two children.



TYPES OF BINARY TREE

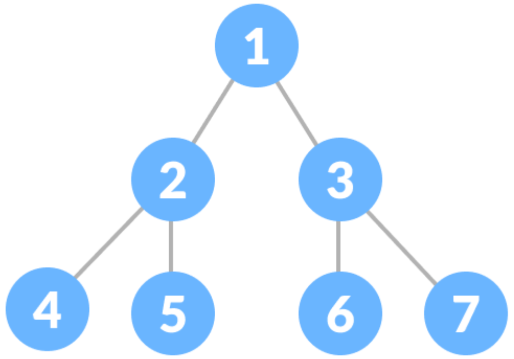
Full Binary Tree

A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children.



Perfect Binary Tree

A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.

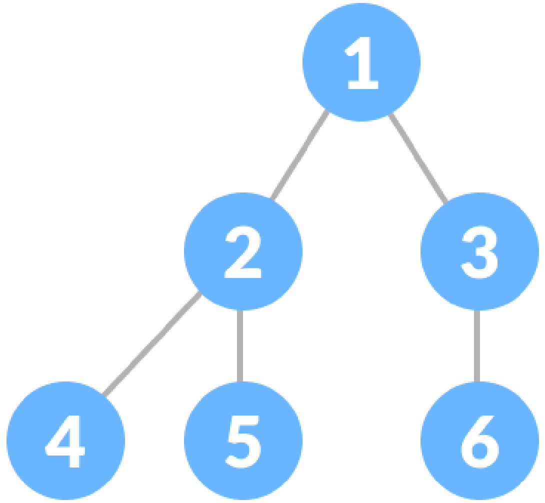


TYPES OF BINARY TREE

Complete Binary Tree

A complete binary tree is just like a full binary tree, but with two major differences

1. Every level must be completely filled
2. All the leaf elements must lean towards the left.
3. The last leaf element might not have a right sibling i.e. a complete binary tree doesn't have to be a full binary tree.

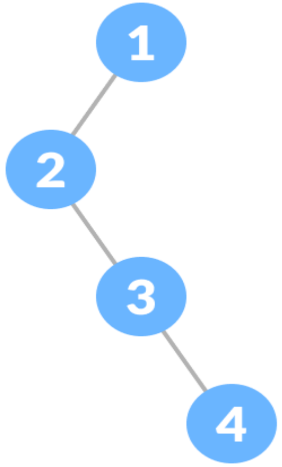


AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

TYPES OF BINARY TREE

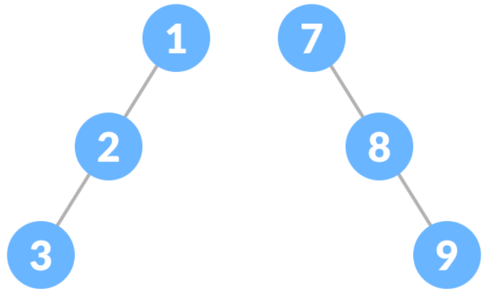
Degenerate or Pathological Tree

A degenerate or pathological tree is the tree having a single child either left or right.



Skewed Binary Tree

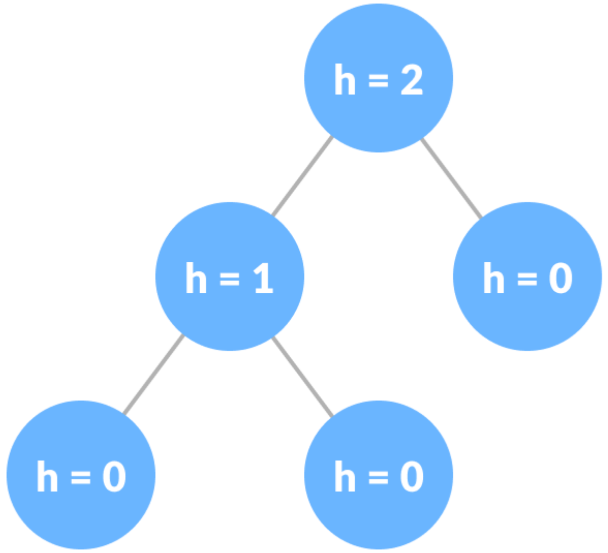
A skewed binary tree is a pathological/degenerate tree in which the tree is either dominated by the left nodes or the right nodes. Thus, there are two types of skewed binary tree: left-skewed binary tree and right-skewed binary tree



TYPES OF BINARY TREE

Balanced Binary Tree

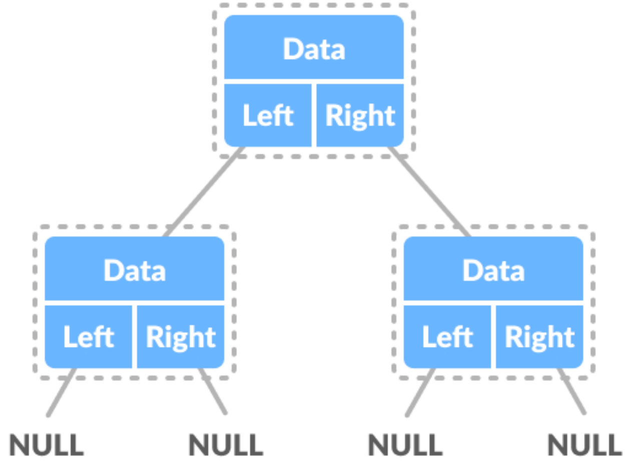
It is a type of binary tree in which the difference between the left and the right subtree for each node is either 0 or 1.



BINARY TREE REPRESENTATION

A node of a binary tree is represented by a structure containing a data part and two pointers to other structures of the same type.

```
struct node
{
 int data;
 struct node *left;
 struct node *right;
   };
```

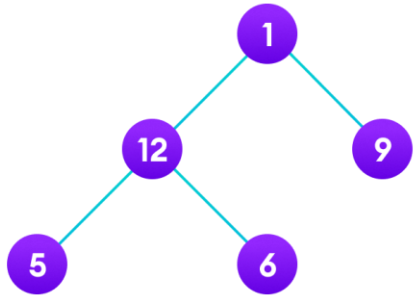


AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BINARY TREE TRAVERSAL

Traversing a tree means visiting every node in the tree. You might, for instance, want to add all the values in the tree or find the largest one. For all these operations, you will need to visit each node of the tree.

Linear data structures like arrays, stacks, queues, and linked list have only one way to read the data. But a hierarchical data structure like a tree can be traversed in different ways.



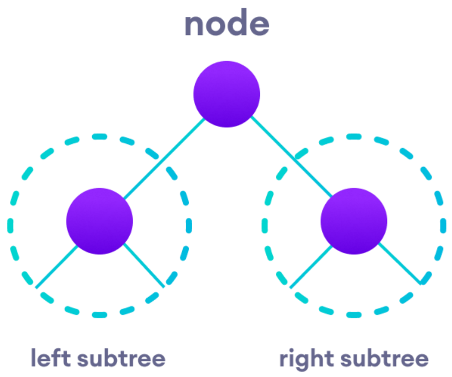
Instead, we use traversal methods that take into account the basic structure of a tree i.e.

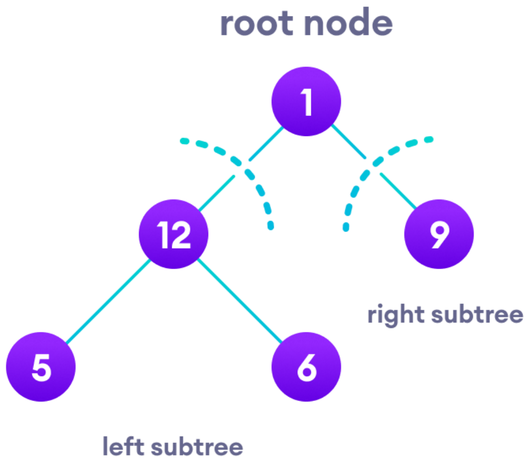
```
struct node {
    int data;
    struct node* left;
    struct node* right;
    }
```

The struct node pointed to by left and right might have other left and right children so we should think of them as sub-trees instead of sub-nodes.

According to this structure, every tree is a combination of

- A node carrying data
- Two subtrees





AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

INORDER TRAVERSAL

1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

```
inorder(root->left)

display(root->data)

   inorder(root->right)
```

PREORDER TRAVERSAL

1. Visit root node
2. Visit all the nodes in the left subtree
3. Visit all the nodes in the right subtree

```
display(root->data)

preorder(root->left)

   preorder(root->right)
```

POSTORDER TRAVERSAL

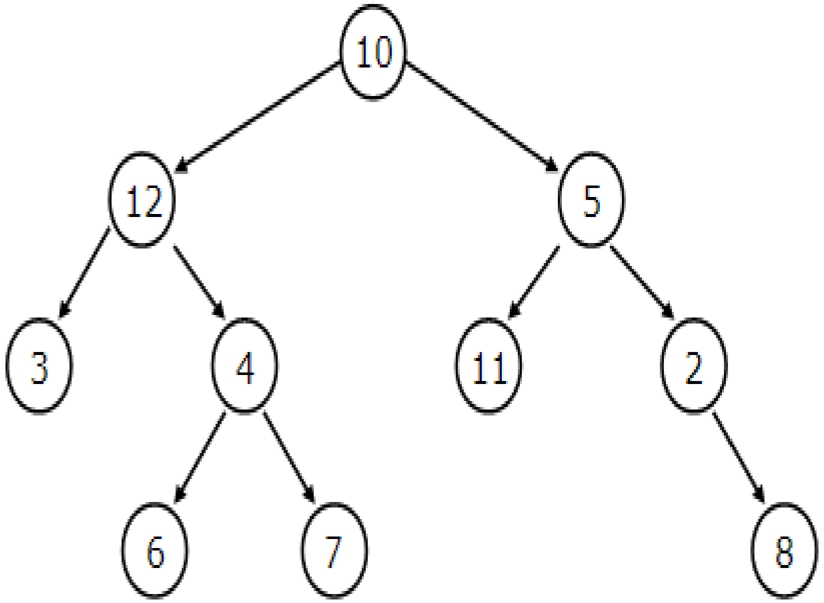
Postorder traversal

1. Visit all the nodes in the left subtree
2. Visit all the nodes in the right subtree
3. Visit the root node

```
postorder(root->left)

postorder(root->right)

  display(root->data)
```



Levelorder tree traversal
10, 12, 5, 3, 4, 11, 2, 6, 7, 8

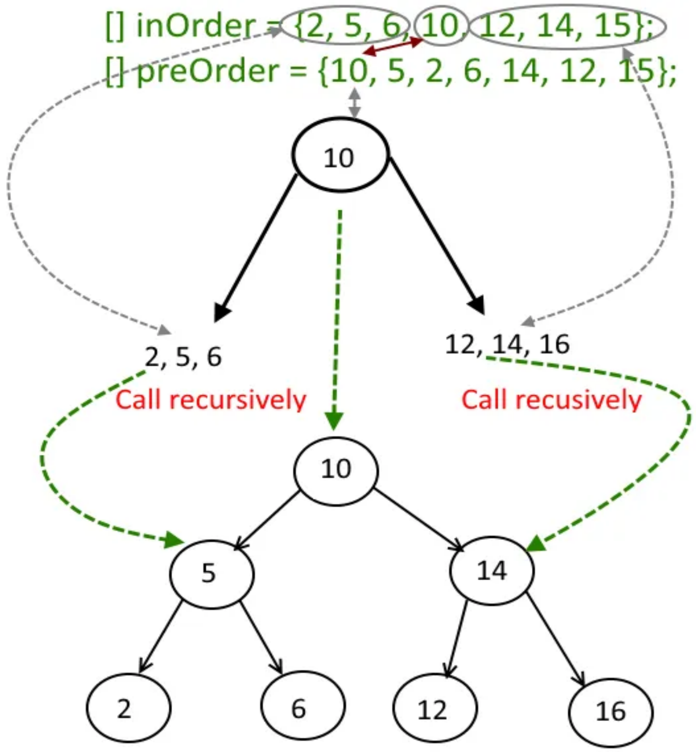
Inorder tree traversal
3, 12, 6, 4, 7, 10, 11, 5, 2, 8

Preorder tree traversal
10, 12, 3, 4, 6, 7, 5, 11, 2, 8

Postorder tree traversal
3, 6, 7, 4, 12, 11, 8, 2, 5, 10

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CONSTRUCT A BINARY TREE FROM TRAVERSAL

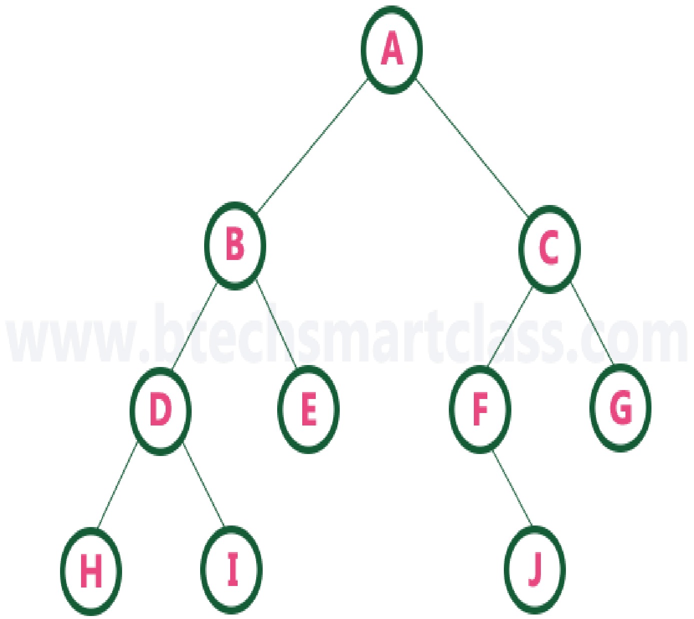

[] inOrder = {2, 5, 6, 10, 12, 14, 15};
[] preOrder = {10, 5, 2, 6, 14, 12, 15};
10
2, 5, 6
12, 14, 16
Call recursively
Call recusively
10
5
14
2
6
12
16

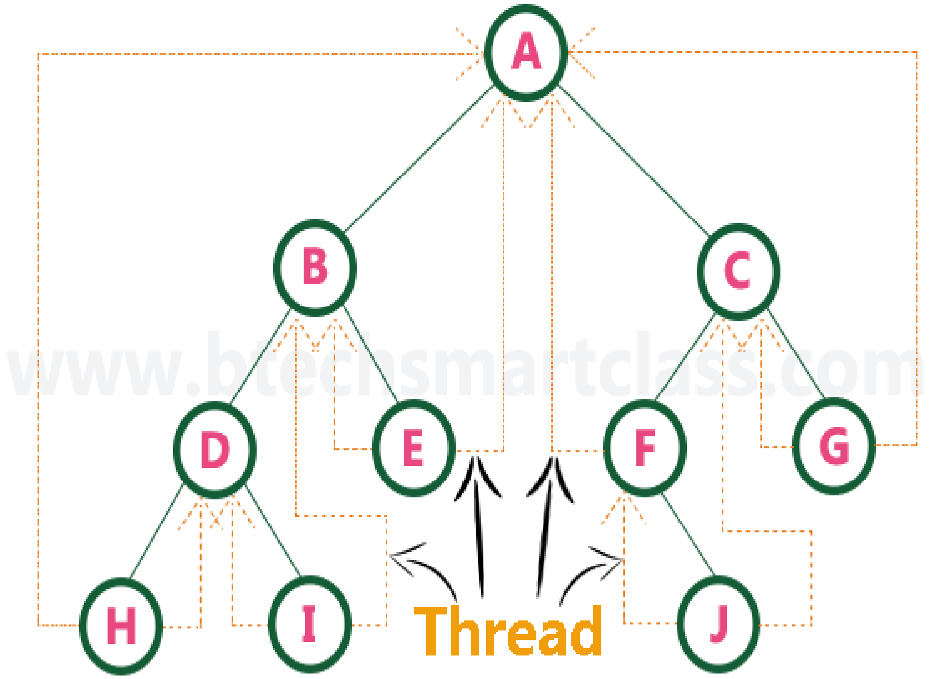
THREAD BINARY TREE

A binary tree can be represented using array representation or linked list representation. When a binary tree is represented using linked list representation, the reference part of the node which doesn't have a child is filled with a NULL pointer. In any binary tree linked list representation, there is a number of NULL pointers than actual pointers. Generally, in any binary tree linked list representation, if there are **2N** number of reference fields, then **N+1** number of reference fields are filled with NULL (**N+1 are NULL out of 2N**). This NULL pointer does not play any role except indicating that there is no link (no child).

A. J. Perlis and C. Thornton have proposed new binary tree called "***Threaded Binary Tree***", which makes use of NULL pointers to improve its traversal process. In a threaded binary tree, NULL pointers are replaced by references of other nodes in the tree. These extra references are called as ***threads***.

Threaded Binary Tree is also a binary tree in which all left child pointers that are NULL (in Linked list representation) points to its in-order predecessor, and all right child pointers that are NULL (in Linked list representation) points to its in-order successor.


A
B
C
D
E
F
G
H
I
J


A
B
C
D
E
F
G
H
I
J
Thread

To convert the above example binary tree into a threaded binary tree, first find the in-order traversal of that tree...

In-order traversal of above binary tree...

H - D - I - B - E - A - F - J - C - G

When we represent the above binary tree using linked list representation, nodes **H, I, E, F, J** and **G** left child pointers are NULL. This NULL is replaced by address of its in-order predecessor respectively (I to D, E to B, F to A, J to F and G to C), but here the node H does not have its in-order predecessor, so it points to the root node A. And nodes **H, I, E, J** and **G** right child pointers are NULL. These NULL pointers are replaced by address of its in-order successor respectively (H to D, I to B, E to A, and J to C), but here the node G does not have its in-order successor, so it points to the root node A.

THREAD BINARY TREE

ADVANTAGES

- **Inorder traversal is faster** than unthreaded version as stack is not required.
- **Effectively determines** the **predecessor and successor** for inorder traversal, for unthreaded tree this task is more difficult.
- **A stack is required** to provide upward pointing information **in binary tree** which **threading provides without stack**.
- It is possible to **generate successor or predecessor** of any node **without** having over head of **stack** with the help of threading.

DISADVANTAGES

- Threaded trees are **unable to share common sub trees**
- If **Negative addressing is not permitted** in programming language, **two additional fields are required**
- **Insertion** into and **deletion** from threaded binary tree are **more time consuming** because both thread and structural link must be maintained

BINARY SEARCH TREE

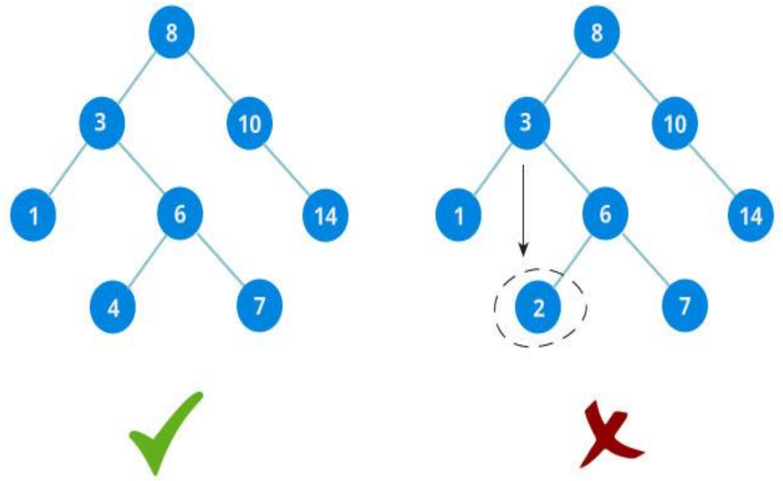
Binary search tree is a data structure that quickly allows us to maintain a sorted list of numbers.

- It is called a binary tree because each tree node has maximum of two children.
- It is called a search tree because it can be used to search for the presence of a number in O(log(n)) time.
- The properties that separates a binary search tree from a regular binary tree is

1. All nodes of left subtree are less than root node
2. All nodes of right subtree are more than root node
3. Both subtrees of each node are also BSTs i.e. they have the above two properties

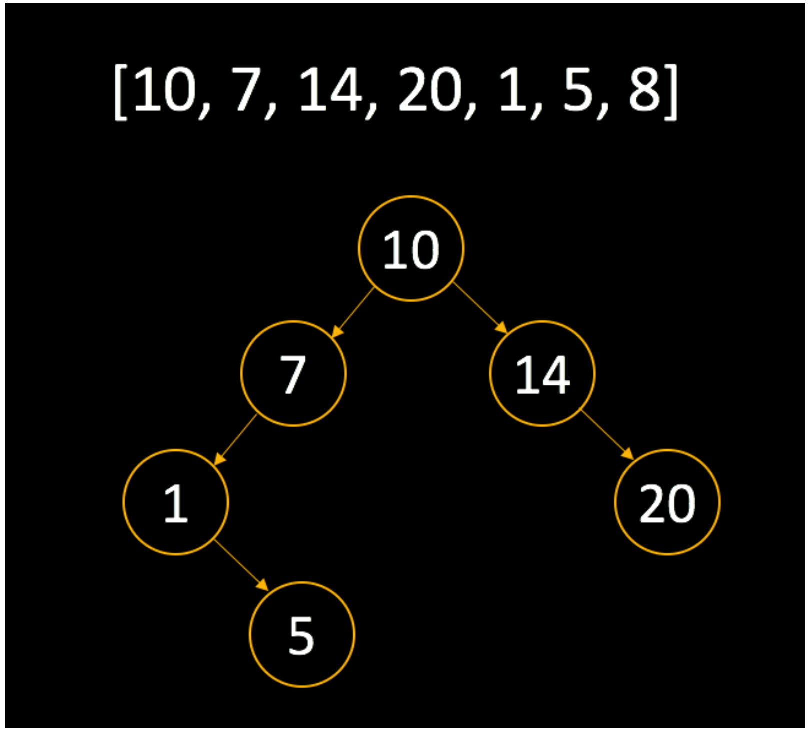
BINARY SEARCH TREE

The binary tree on the right isn't a binary search tree because the right subtree of the node "3" contains a value smaller that it.



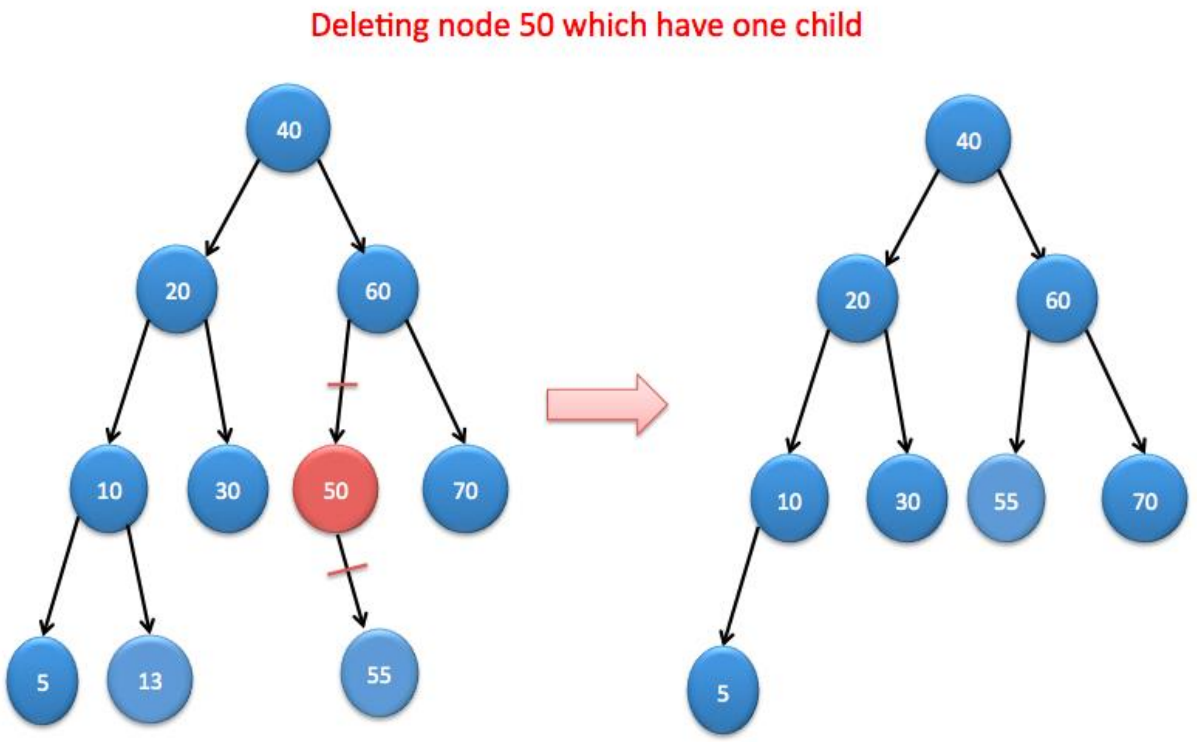
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

BINARY SEARCH TREE IS CONSTRUCTED FROM GIVEN DATA



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

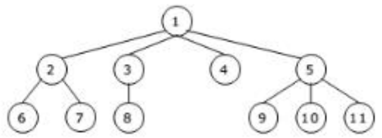
DELETE A NODE FROM BINARY SEARCH TREE



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

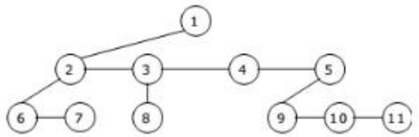
CONVERSION OF GENERAL TREES TO BINARY

Convert the following ordered tree into a binary tree:

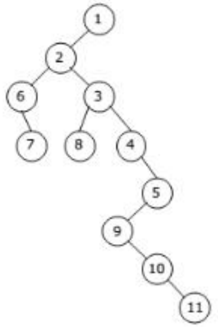


Solution:

Stage 1 tree by using the above mentioned procedure is as follows:



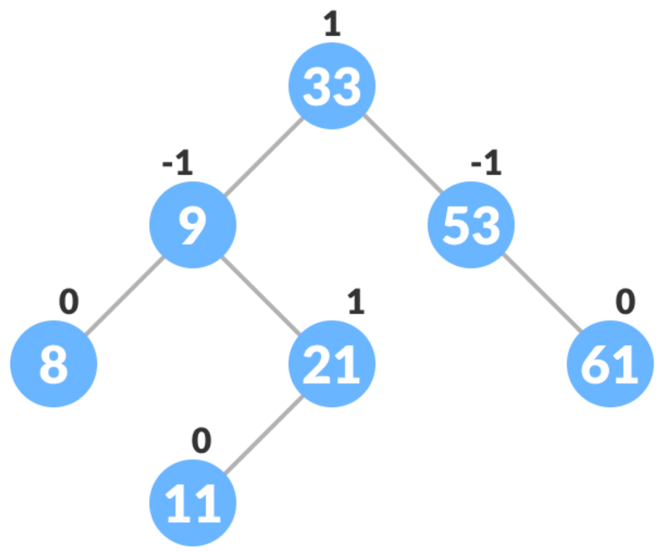
Stage 2 tree by using the above mentioned procedure is as follows:



AVL TREES

AVL tree is a self-balancing binary search tree in which each node maintains extra information called a balance factor whose value is either -1, 0 or +1.

AVL tree got its name after its inventor Georgy Adelson-Velsky and Landis.



Balance Factor

Balance factor of a node in an AVL tree is the difference between the height of the left subtree and that of the right subtree of that node.

Balance Factor = (Height of Left Subtree - Height of Right Subtree) or (Height of Right Subtree - Height of Left Subtree)

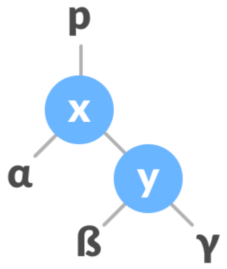
The self balancing property of an avl tree is maintained by the balance factor. The value of balance factor should always be -1, 0 or +1.

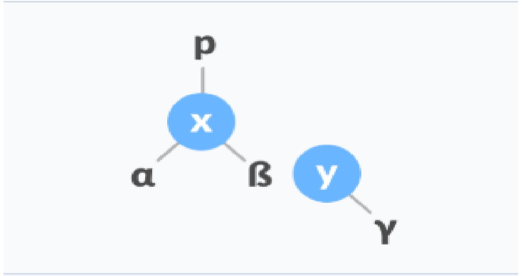
LEFT ROTATION ON AVL TREE

In left-rotation, the arrangement of the nodes on the right is transformed into the arrangements on the left node.

Algorithm

1. Let the initial tree be:Left rotate
2. If y has a left subtree, assign x as the parent of the left subtree of y

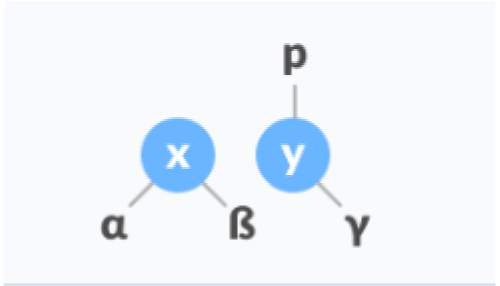




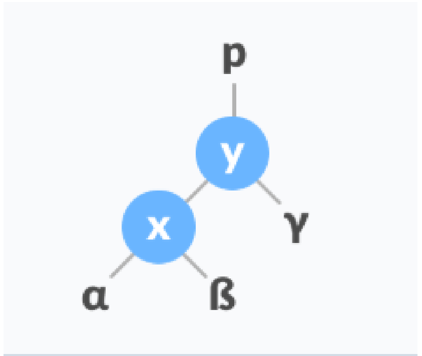
Assign x as the parent of the left subtree of y

LEFT ROTATION ON AVL TREE

- If y has a left subtree, assign x as the parent of the left subtree of y
- If the parent of x is NULL, make y as the root of the tree.
- Else if x is the left child of p, make y as the left child of p.
- Else assign y as the right child of p.
- Make y as the parent of x



Change the parent of x to that of y



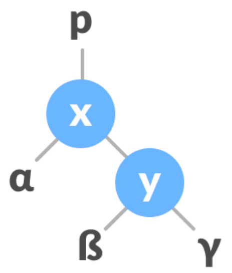
Assign y as the parent of x.

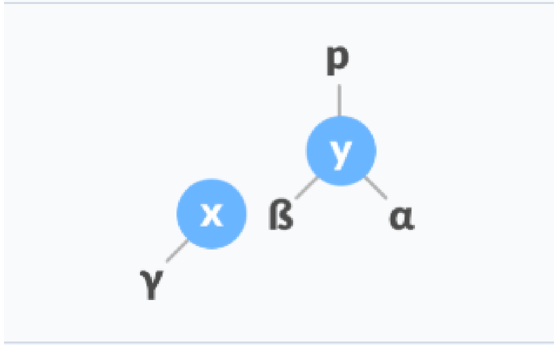
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

RIGHT ROTATION ON AVL TREE

In left-rotation, the arrangement of the nodes on the left is transformed into the arrangements on the right node.

- ❖ Let the initial tree be
- ❖ If x has a right subtree, assign y as the parent of the right subtree of x
- ❖ If the parent of y is NULL, make x as the root of the tree.

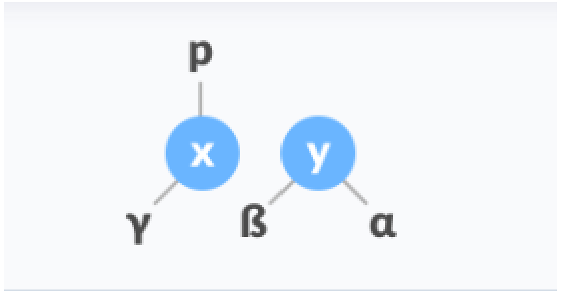




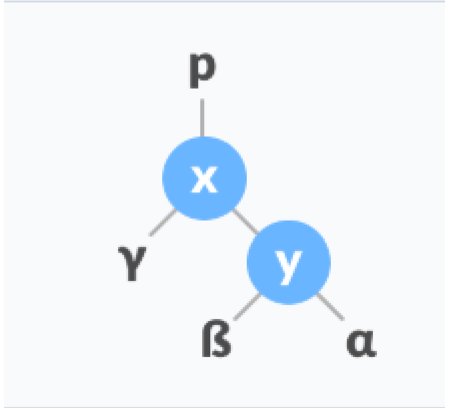
Assign y as the parent of the right subtree of x

RIGHT ROTATION ON AVL TREE

- Else if y is the right child of its parent p, make x as the right child of p.
- Else assign x as the left child of p.
- Make x as the parent of y



Assign the parent of y as the parent of x.

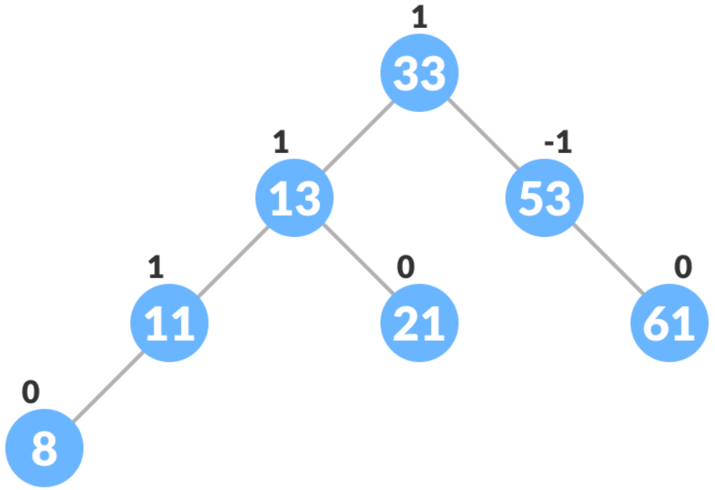


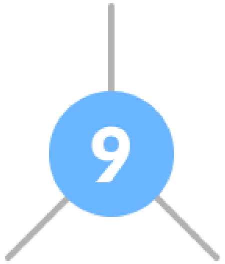
Assign x as the parent of y

ALGORITHM TO INSERT A NODE

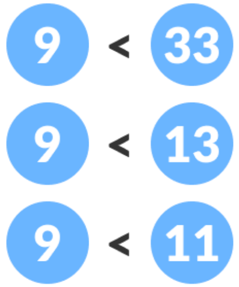
A newNode is always inserted as a leaf node with balance factor equal to 0.

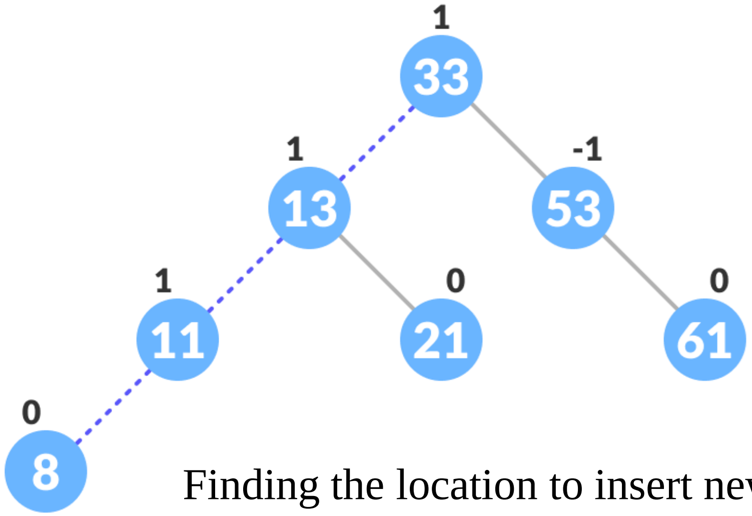
1. Let the initial tree be





New node





Finding the location to insert newNode

2.Go to the appropriate leaf node to insert a newNode using the following recursive steps. Compare newKey with rootKey of the current tree.

1. If newKey < rootKey, call insertion algorithm on the left subtree of the current node until the leaf node is reached.
2. Else if newKey > rootKey, call insertion algorithm on the right subtree of current node until the leaf node is reached.
3. Else, return leafNode

3.Compare leafKey obtained from the above steps with newKey:

1. If newKey < leafKey, make newNode as the leftChild of leafNode.
2. Else, make newNode as rightChild of leafNode

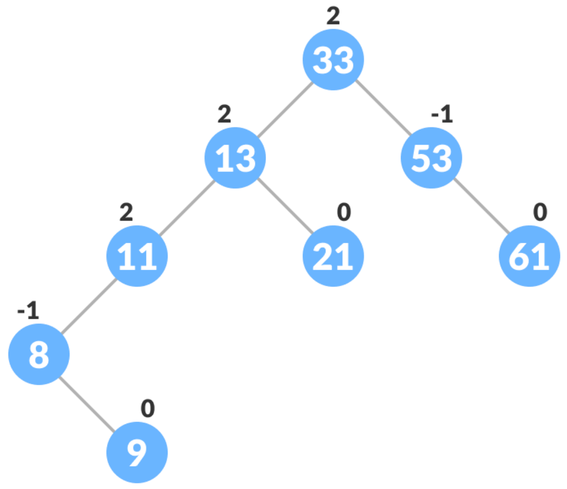
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

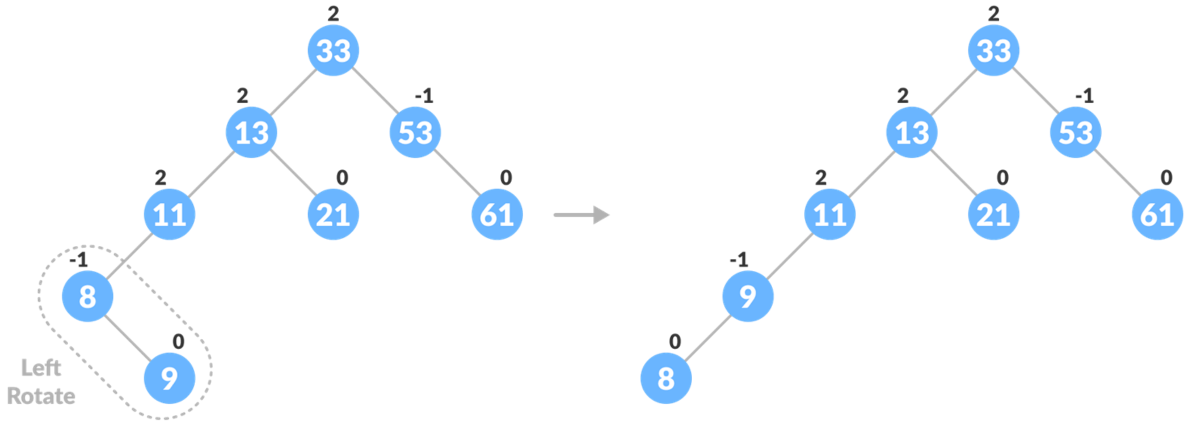
ALGORITHM TO INSERT A NODE

4.Update balanceFactor of the nodes.

5.If the nodes are unbalanced, then rebalance the node.

1. If balanceFactor > 1, it means the height of the left subtree is greater than that of the right subtree. So, do a right rotation or left-right rotation
    1. If newNodeKey < leftChildKey do right rotation.
    2. Else, do left-right rotation

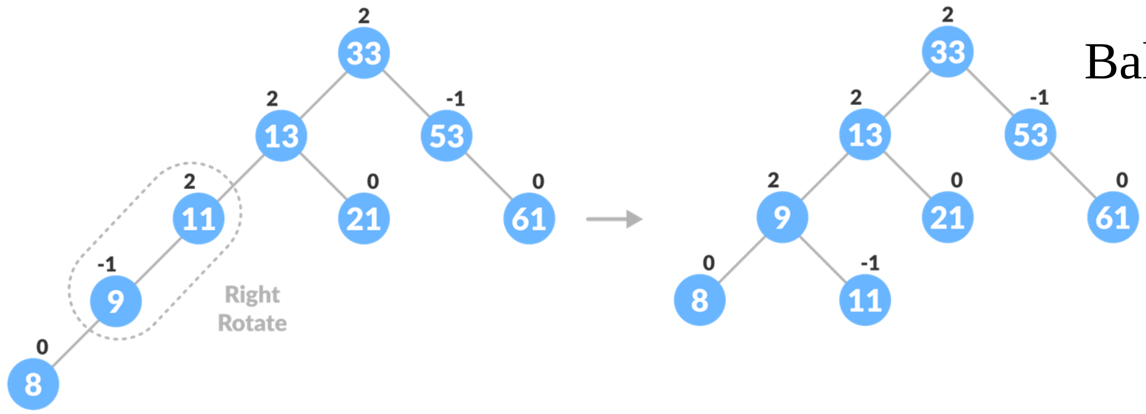


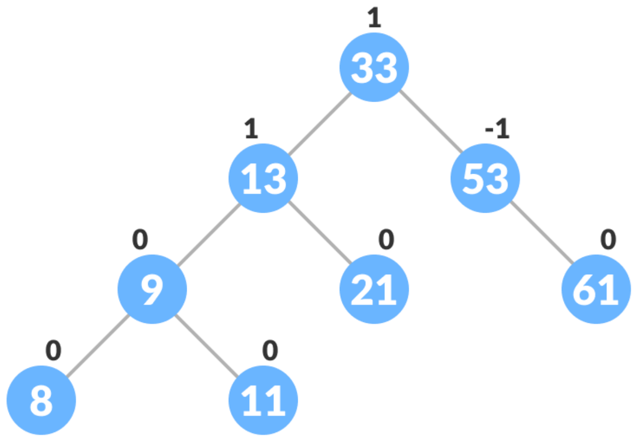


Balancing the tree with rotation

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Balancing the tree with rotation





Final Balanced Tree

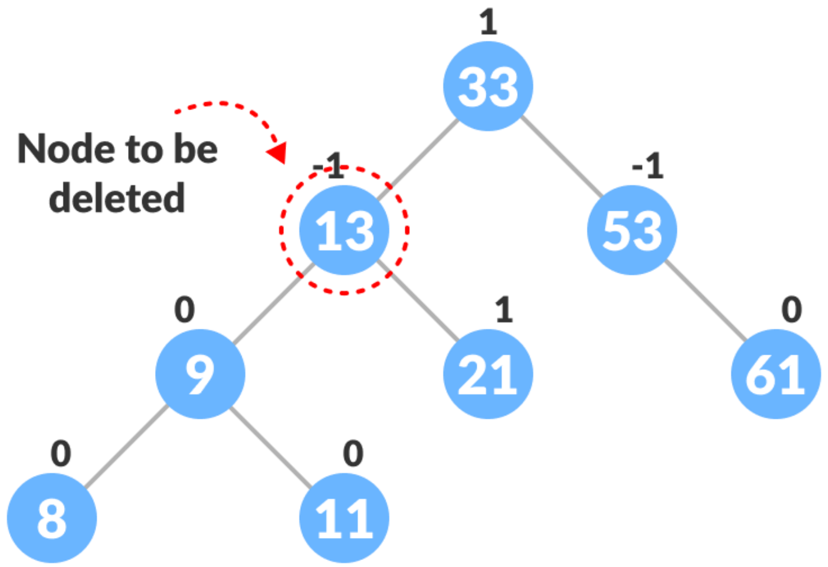
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

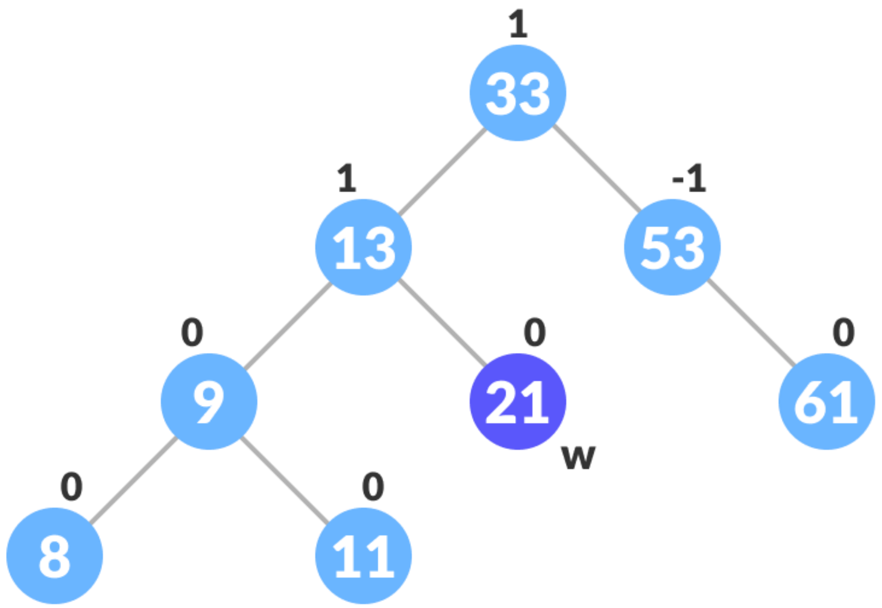
ALGORITHM TO DELETE A NODE

A node is always deleted as a leaf node. After deleting a node, the balance factors of the nodes get changed. In order to rebalance the balance factor, suitable rotations are performed.

1. Locate nodeToBeDeleted (recursion is used to find nodeToBeDeleted in the code used below).
2. There are three cases for deleting a node:

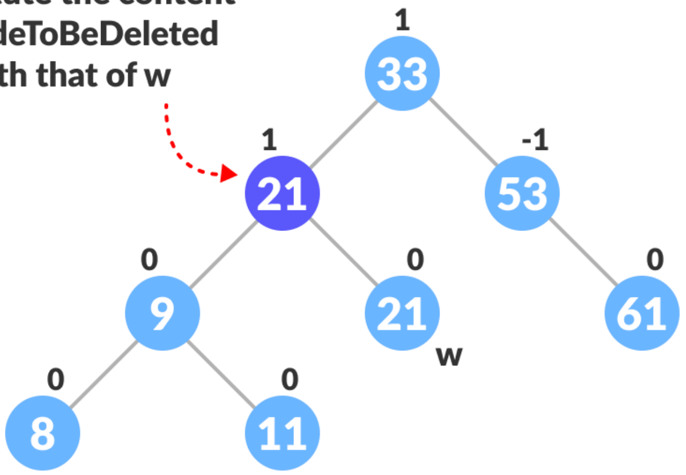
- ❖ If nodeToBeDeleted is the leaf node (ie. does not have any child), then remove nodeToBeDeleted.
- ❖ If nodeToBeDeleted has one child, then substitute the contents of nodeToBeDeleted with that of the child. Remove the child.
- ❖ If nodeToBeDeleted has two children, find the inorder successor w of nodeToBeDeleted (ie. node with a minimum value of key in the right subtree).


Node to be deleted
1
33
-1
13
-1
53
0
9
1
21
0
61
0
8
0
11

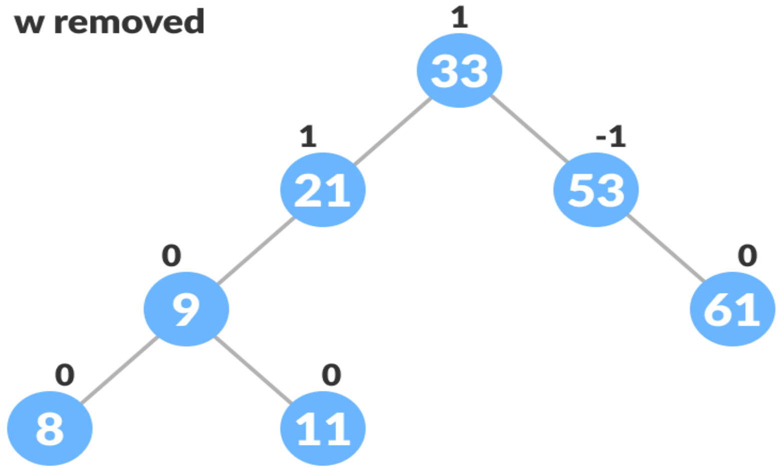

1
33
1
13
-1
53
0
9
0
21
w
0
61
0
8
0
11

- Update balanceFactor of the nodes.
- Rebalance the tree if the balance factor of any of the nodes is not equal to -1, 0 or 1.
- If balanceFactor of currentNode > 1,
  - If balanceFactor of leftChild >= 0, do right rotation
  - Else do left-right rotation.
- If balanceFactor of currentNode < -1,
  - If balanceFactor of rightChild <= 0, do left rotation.
  - Else do right-left rotation.

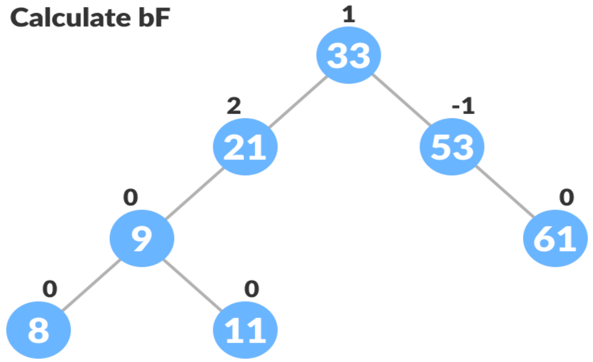
Substitute the content of nodeToBeDeleted with that of w



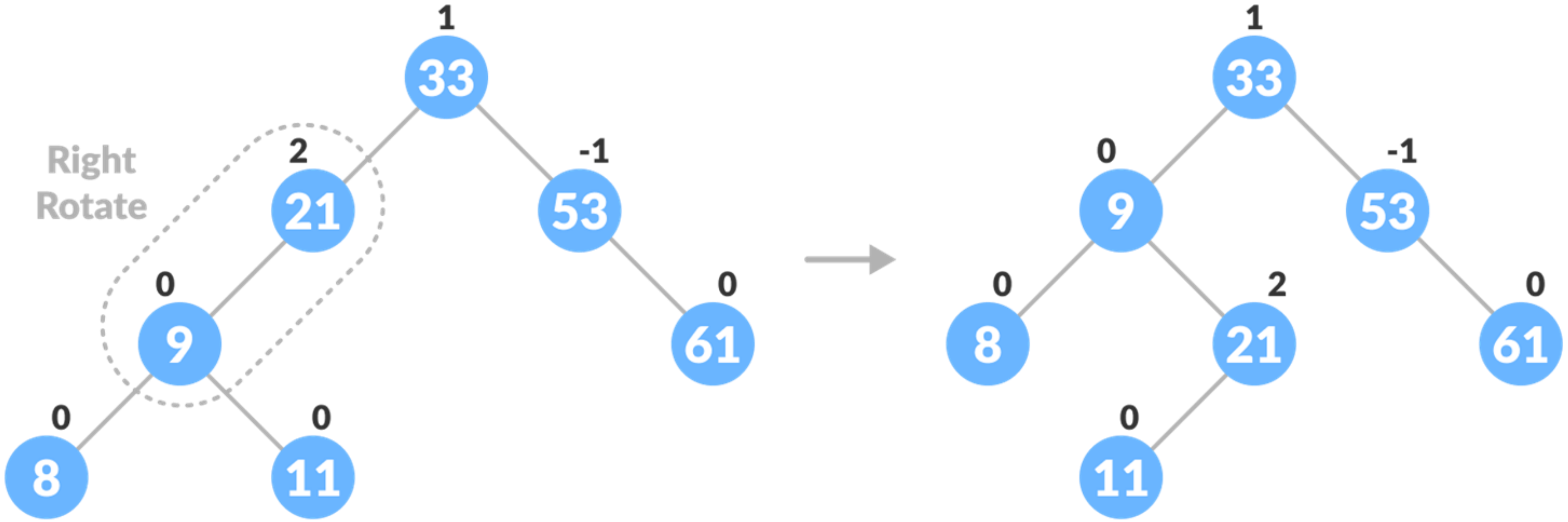
w removed



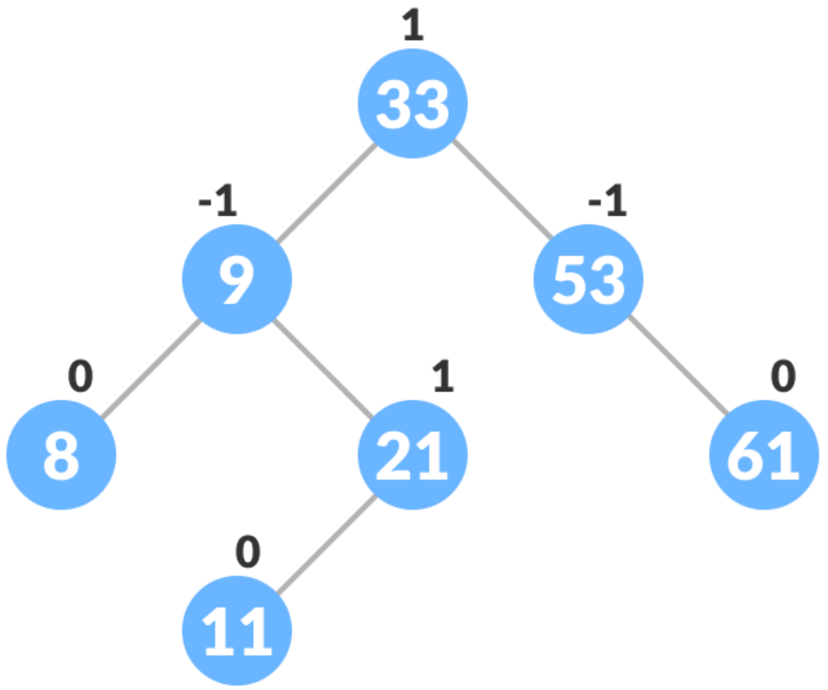
Calculate bF



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

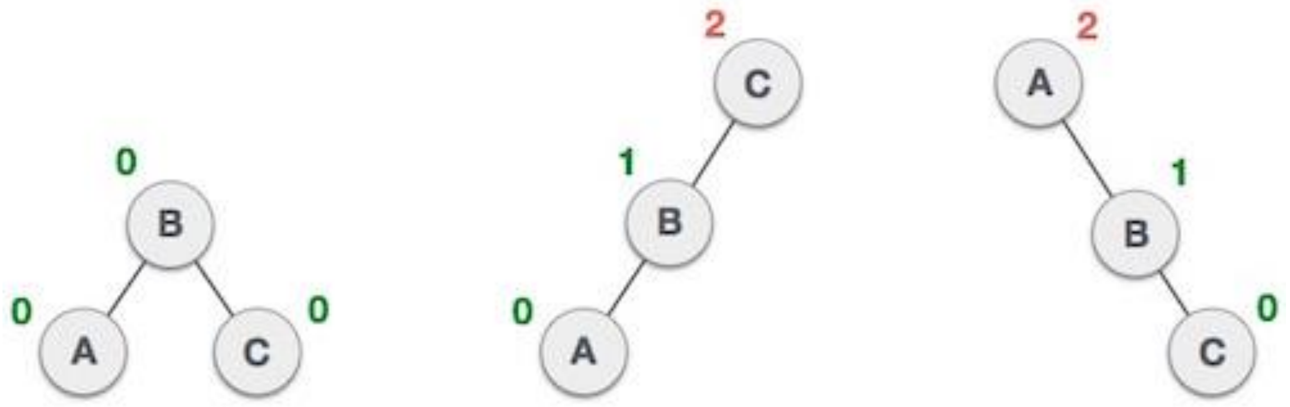


Right Rotate
1
33
2
21
-1
53
0
9
0
61
0
8
0
11
1
33
0
9
-1
53
0
8
2
21
0
61
0
11



AVL FINAL TREE

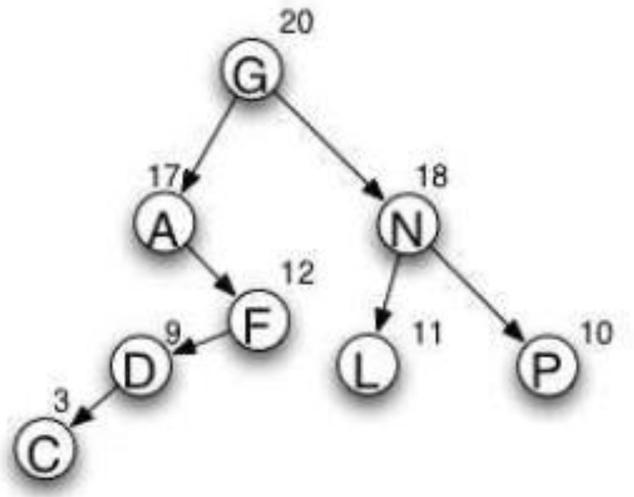
HEIGHT BALANCED TREE



Balanced   Not balanced   Not balanced

WEIGHT BALANCED TREE

A **weight-balanced binary tree** is a binary tree which is balanced based on knowledge of the probabilities of searching for each individual node. Within each subtree, the node with the highest weight appears at the root. This can result in more efficient searching performance. Construction of such a tree is similar to that of a Treap, but node weights are chosen randomly in the latter.



Example of weight balanced tree

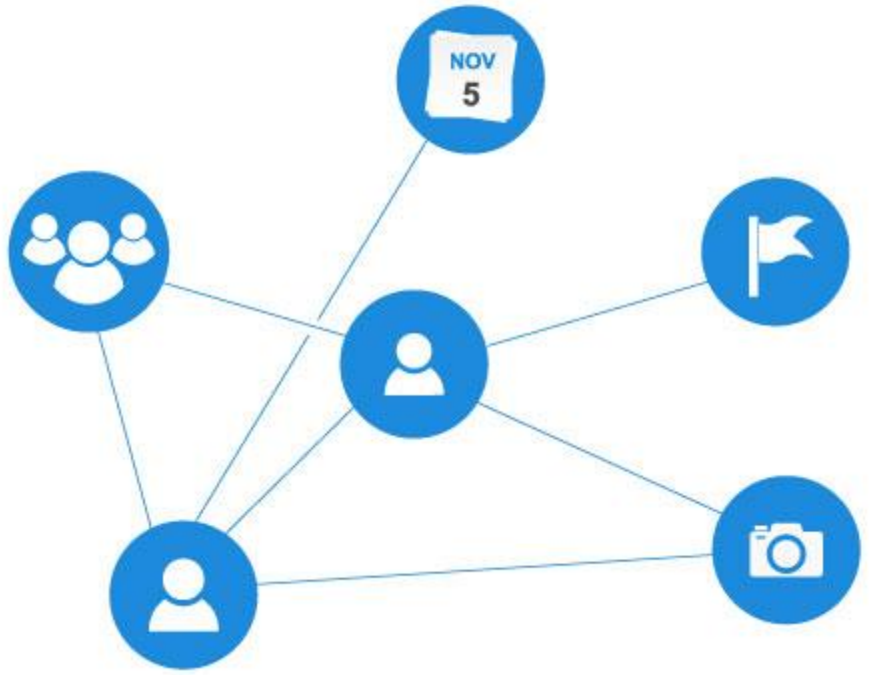
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

GRAPH MATRIX REPRESENTATION OF GRAPHS

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.

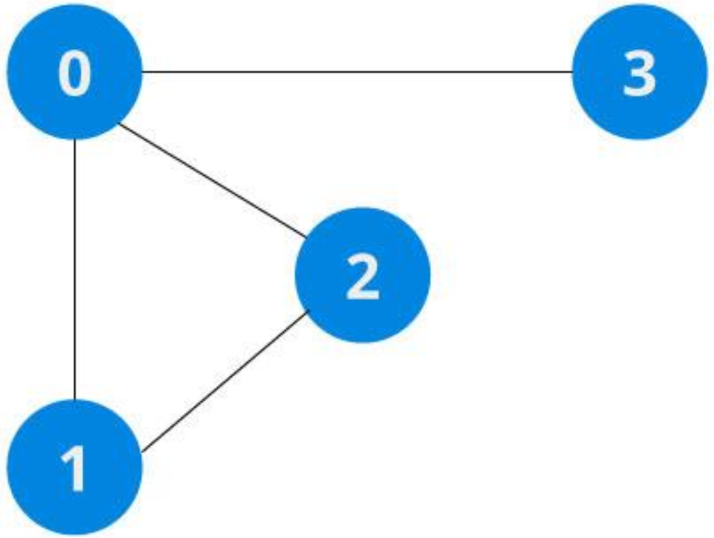


Example of graph data structure

All of facebook is then a collection of these nodes and edges. This is because facebook uses a graph data structure to store its data.

More precisely, 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)



Vertices and edges

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.

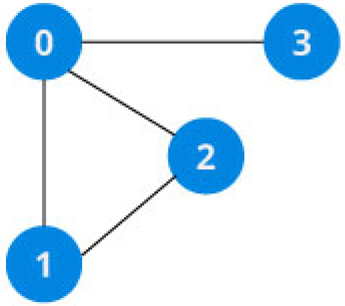
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

GRAPH REPRESENTATION

1. 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.
- ❖ Since 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.
- ❖ Edge lookup(checking if an edge exists between vertex A and vertex B) is extremely fast in adjacency matrix representation but we have to reserve space for every possible link between all vertices(V x V), so it requires more space.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

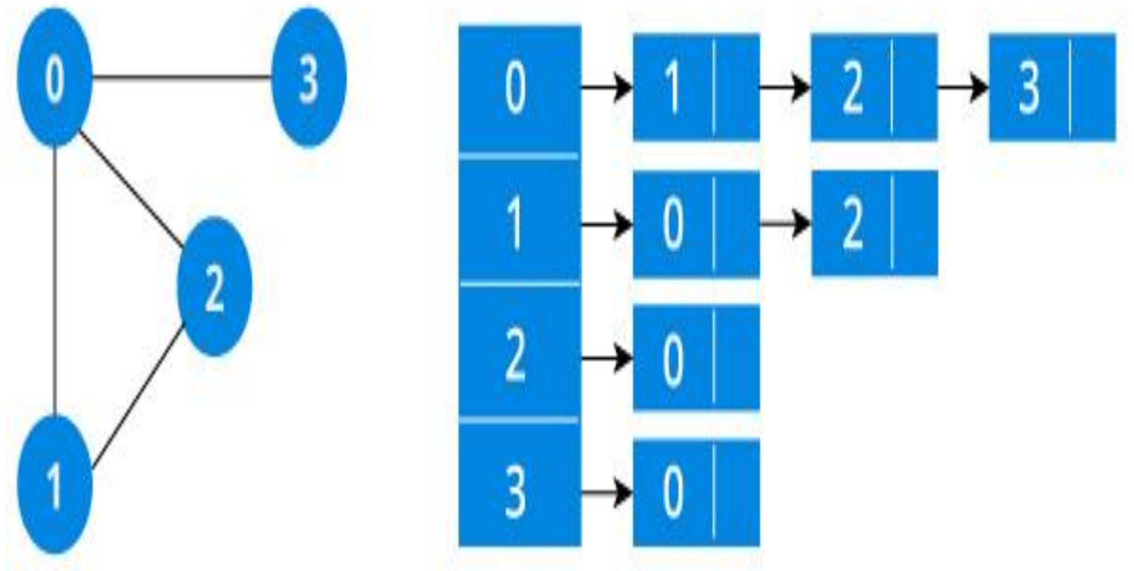


$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 0 & 0 \end{array}$$

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

2. 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.
- The adjacency list for the graph we made in the first example is as follows:
- An adjacency list is efficient in terms of storage because we only need to store the values for the edges. For a graph with millions of vertices, this can mean a lot of saved space.



Adjacency list representation

ELEMENTARY GRAPH OPERATION

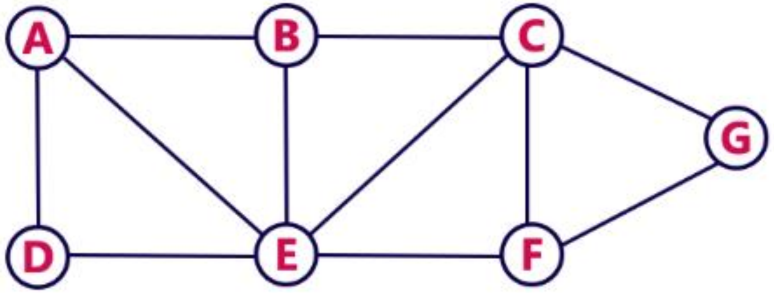
BREADTH FIRST SEARCH

BFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Queue data structure** with maximum size of total number of vertices in the graph to implement BFS traversal.

We use the following steps to implement BFS traversal...

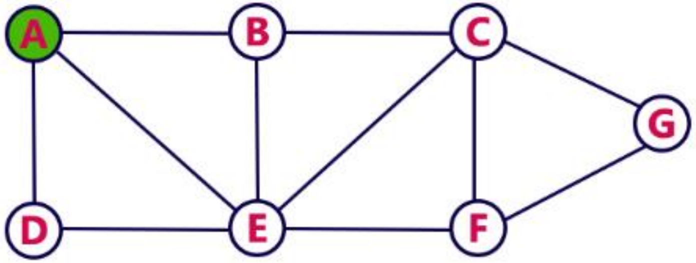
- Step 1 - Define a Queue of size total number of vertices in the graph.
- Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.
- Step 3 - Visit all the non-visited **adjacent** vertices of the vertex which is at front of the Queue and insert them into the Queue.
- Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
- Step 5 - Repeat steps 3 and 4 until queue becomes empty.
- Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

Consider the following example graph to perform BFS traversal



Step 1:

- Select the vertex **A** as starting point (visit **A**).
- Insert **A** into the Queue.

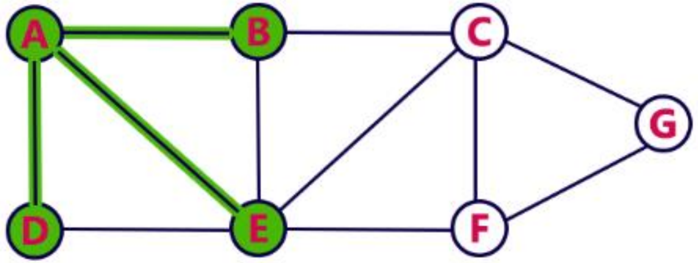


Queue

A						

Step 2:

- Visit all adjacent vertices of **A** which are not visited (**D**, **E**, **B**).
- Insert newly visited vertices into the Queue and delete A from the Queue..

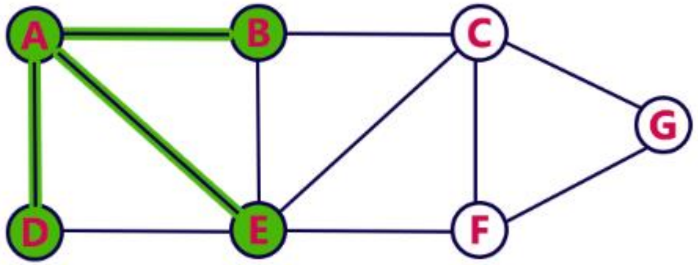


Queue

	D	E	B			

Step 3:

- Visit all adjacent vertices of **D** which are not visited (there is no vertex).
- Delete D from the Queue.

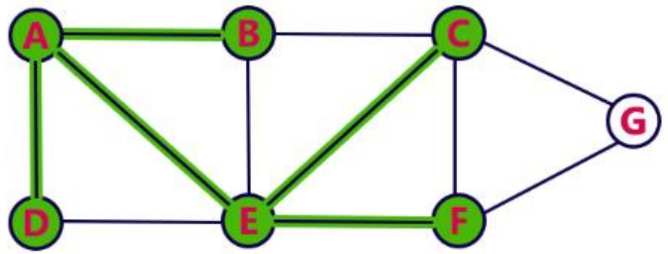


Queue

		E	B			

Step 4:

- Visit all adjacent vertices of **E** which are not visited (**C**, **F**).
- Insert newly visited vertices into the Queue and delete E from the Queue.

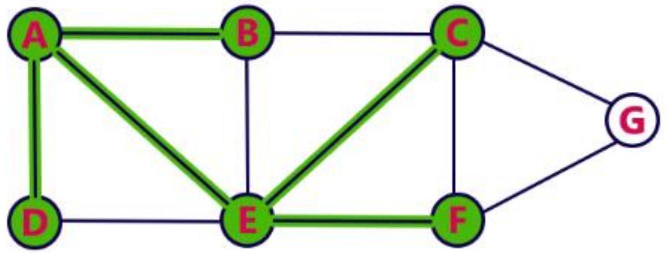


Queue

			B	C	F	

Step 5:

- Visit all adjacent vertices of **B** which are not visited (**there is no vertex**).
- Delete **B** from the Queue.



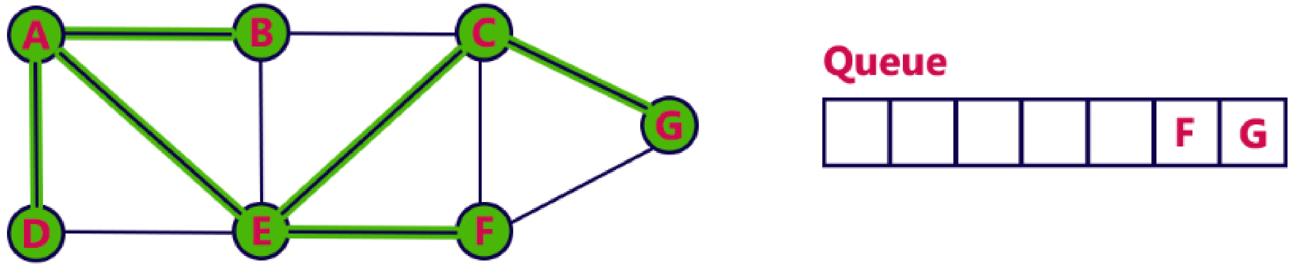
Queue

				C	F	

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

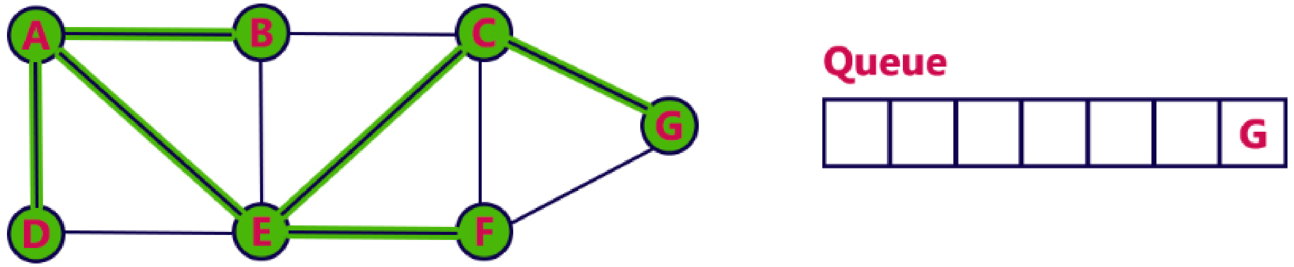
Step 6:

- Visit all adjacent vertices of **C** which are not visited (**G**).
- Insert newly visited vertex into the Queue and delete **C** from the Queue.



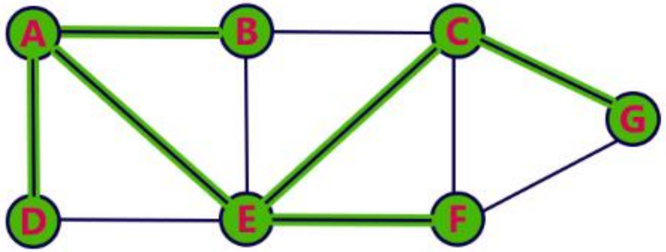
Step 7:

- Visit all adjacent vertices of **F** which are not visited (**there is no vertex**).
- Delete **F** from the Queue.

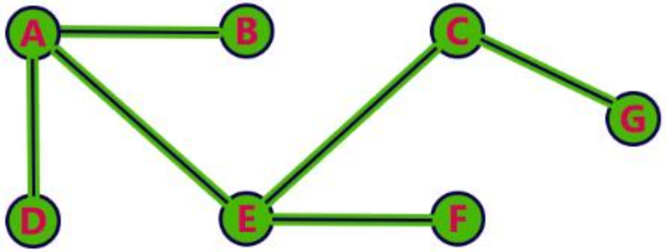


Step 8:

- Visit all adjacent vertices of **G** which are not visited (**there is no vertex**).
- Delete **G** from the Queue.



- Queue became Empty. So, stop the BFS process.
- Final result of BFS is a Spanning Tree as shown below...



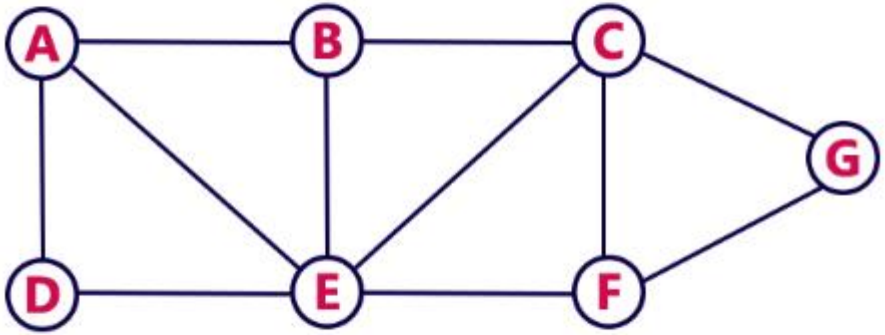
DEPTH FIRST SEARCH

DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal.

We use the following steps to implement DFS traversal...

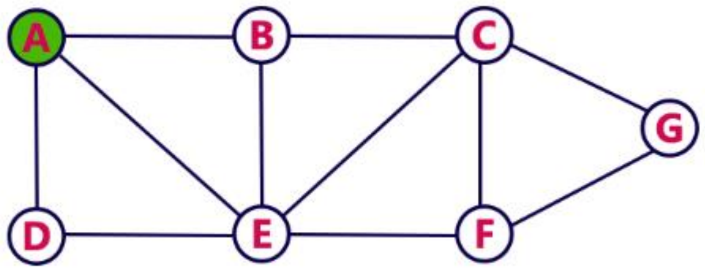
- Step 1 - Define a Stack of size total number of vertices in the graph.
- Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
- Step 3 - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.
- Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- Step 5 - When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.
- Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.
- Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

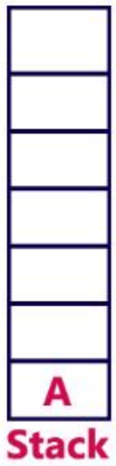
Consider the following example graph to perform DFS traversal



Step 1:

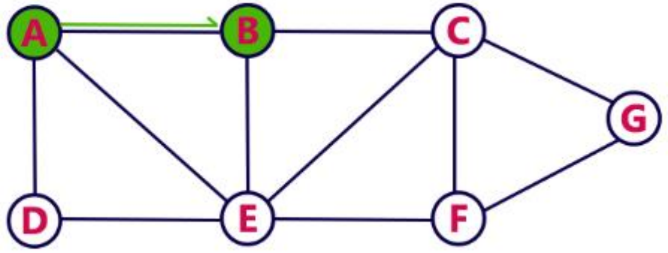
- Select the vertex **A** as starting point (visit **A**).
- Push **A** on to the Stack.

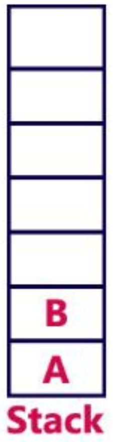




Step 2:

- Visit any adjacent vertex of **A** which is not visited (**B**).
- Push newly visited vertex B on to the Stack.

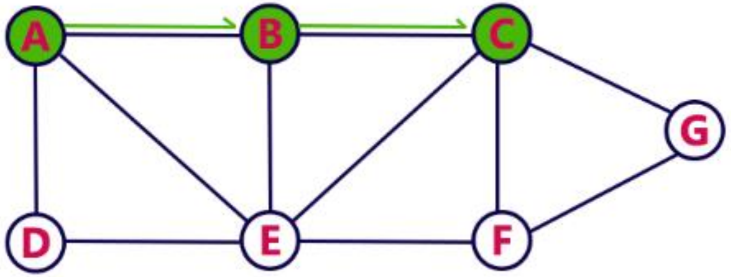


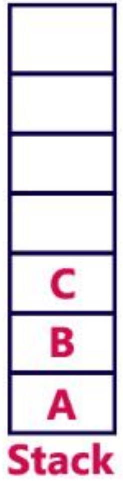


AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Step 3:

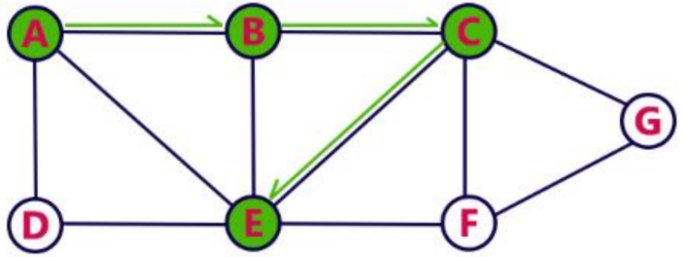
- Visit any adjacent vertext of **B** which is not visited (**C**).
- Push C on to the Stack.

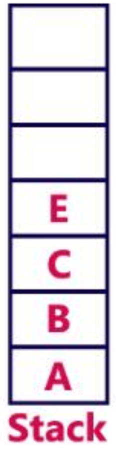




Step 4:

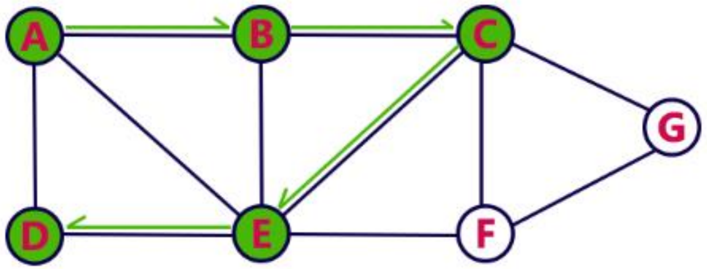
- Visit any adjacent vertext of **C** which is not visited (**E**).
- Push E on to the Stack

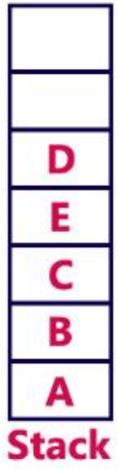




Step 5:

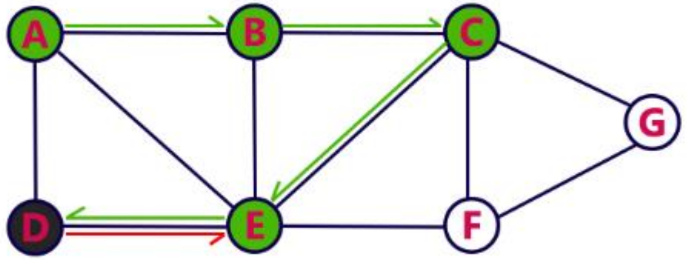
- Visit any adjacent vertext of **E** which is not visited (**D**).
- Push D on to the Stack

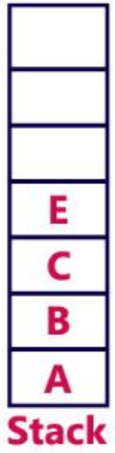




Step 6:

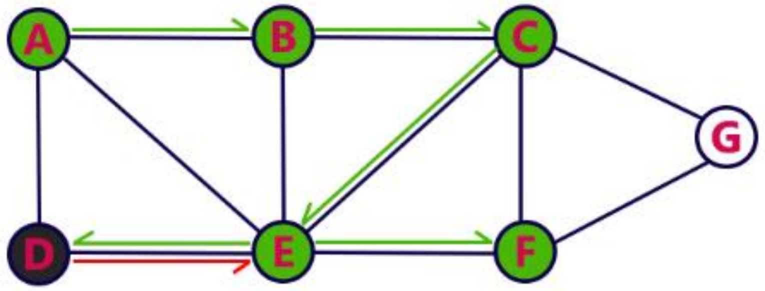
- There is no new vertiex to be visited from D. So use back track.
- Pop D from the Stack.

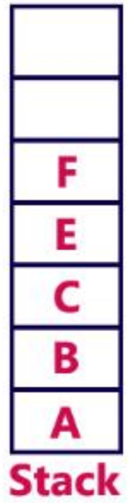




Step 7:

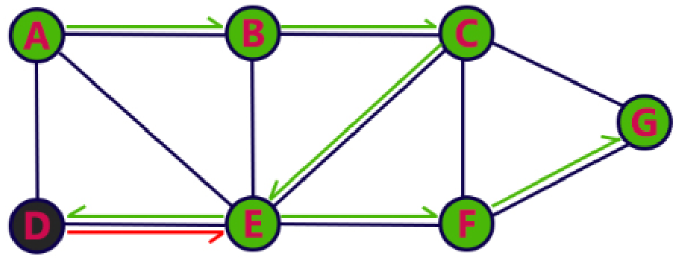
- Visit any adjacent vertex of **E** which is not visited (**F**).
- Push **F** on to the Stack.

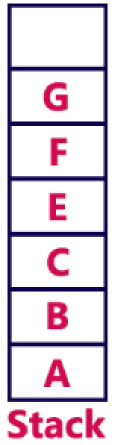




Step 8:

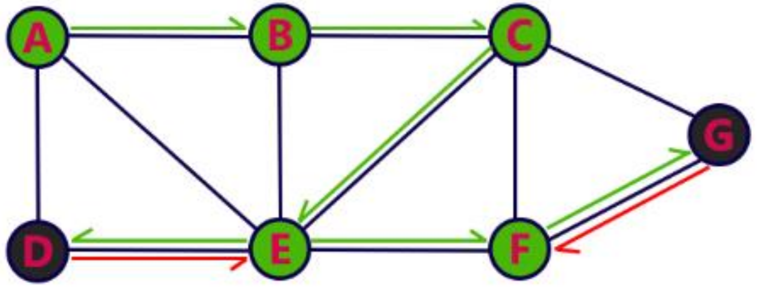
- Visit any adjacent vertex of **F** which is not visited (**G**).
- Push **G** on to the Stack.

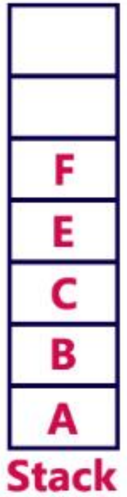




Step 9:

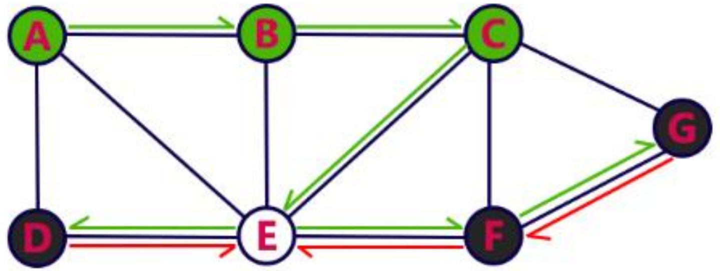
- There is no new vertiex to be visited from G. So use back track.
- Pop G from the Stack.

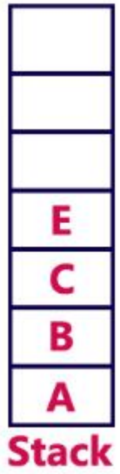




Step 10:

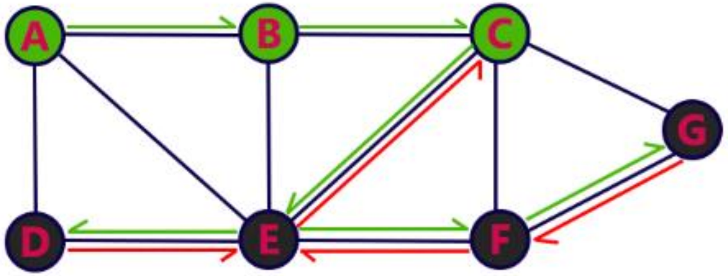
- There is no new vertiex to be visited from F. So use back track.
- Pop F from the Stack.

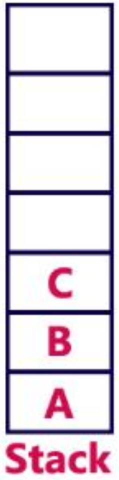




Step 11:

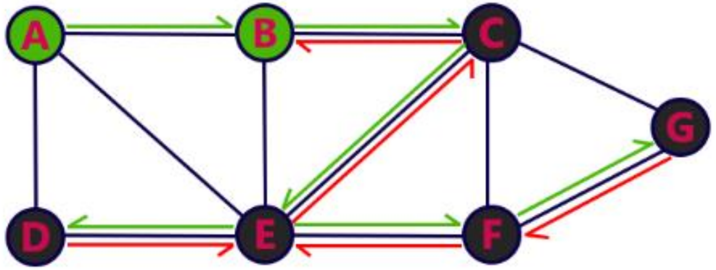
- There is no new vertiex to be visited from E. So use back track.
- Pop E from the Stack.

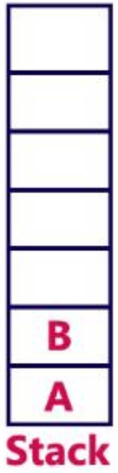




Step 12:

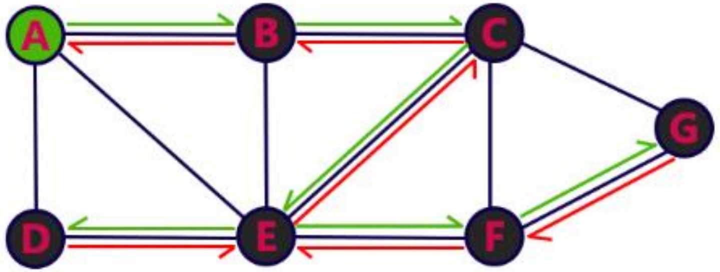
- There is no new vertiex to be visited from C. So use back track.
- Pop C from the Stack.

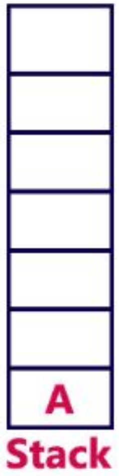




Step 13:

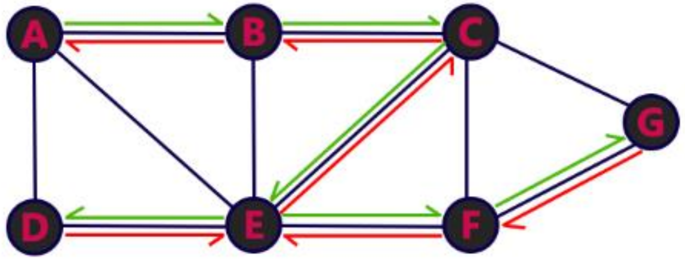
- There is no new vertiex to be visited from B. So use back track.
- Pop B from the Stack.

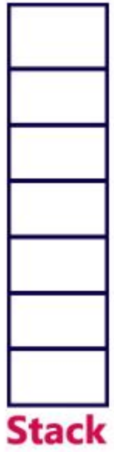




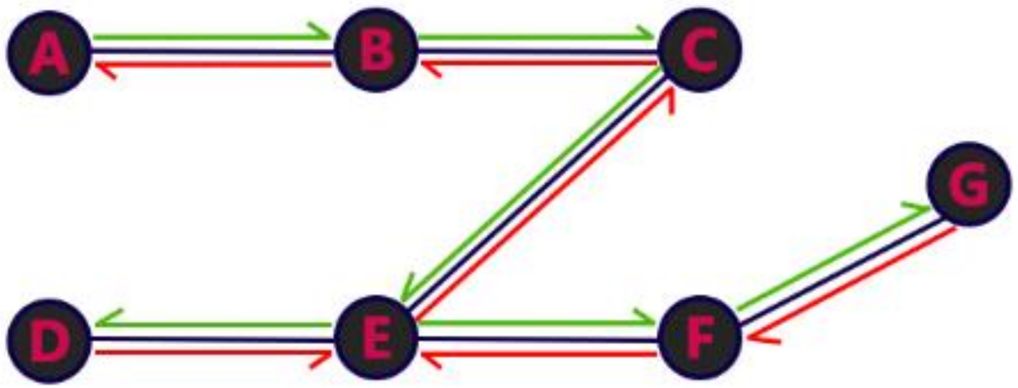
Step 14:

- There is no new vertiex to be visited from A. So use back track.
- Pop A from the Stack.



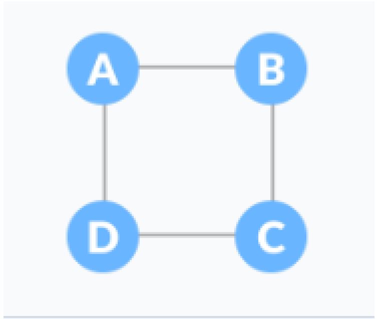


- Stack became Empty. So stop DFS Treversal.
- Final result of DFS traversal is following spanning tree.

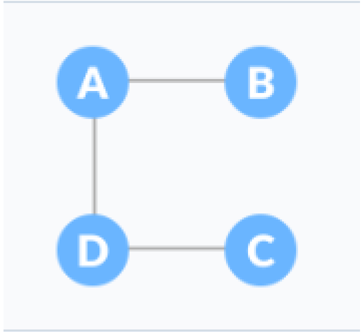


SPANNING TREES

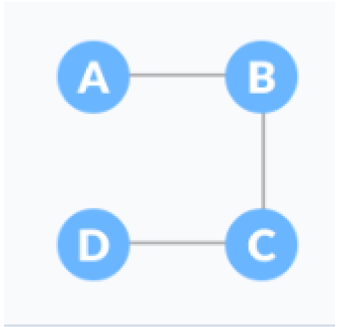
- ❖ A spanning tree is a sub-graph of an undirected and a connected graph, which includes all the vertices of the graph having a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.
- ❖ The edges may or may not have weights assigned to them.
- ❖ The total number of spanning trees with n vertices that can be created from a complete graph is equal to n(n-2).



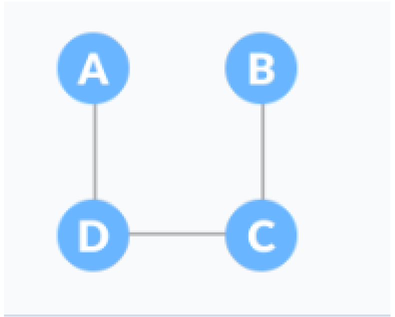
Normal graph



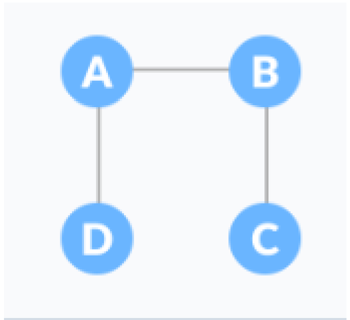
A spanning tree



A spanning tree



A spanning tree



A spanning tree

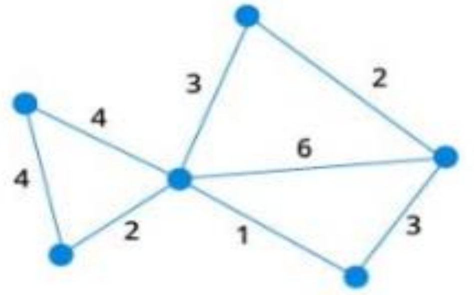
AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

SHORTEST PATH FIRST

- Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph.
- It differs from minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.
- Dijkstra's Algorithm works on the basis that any subpath B -> D of the shortest path A -> D between vertices A and D is also the shortest path between vertices B and D.
- Djikstra used this property in the opposite direction i.e we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbours to find the shortest subpath to those neighbours.
- The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

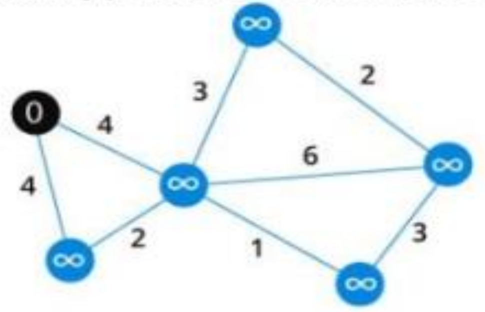
1

Start with a weighted graph



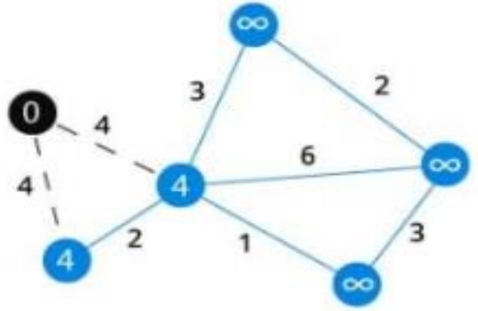
2

Choose a starting vertex and assign infinity path values to all other vertices



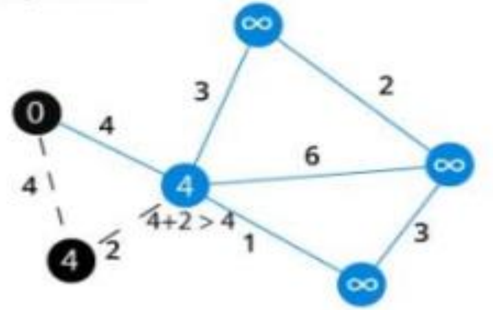
3

Go to each vertex adjacent to this vertex and update its path length



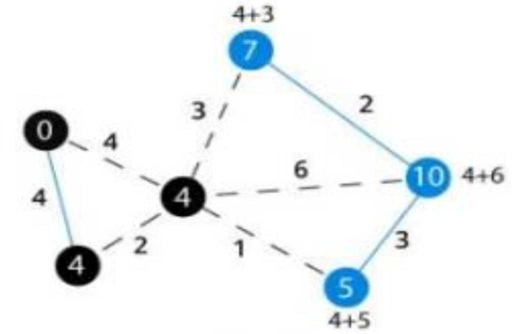
4

If the path length of adjacent vertex is lesser than new path length, don't update it.



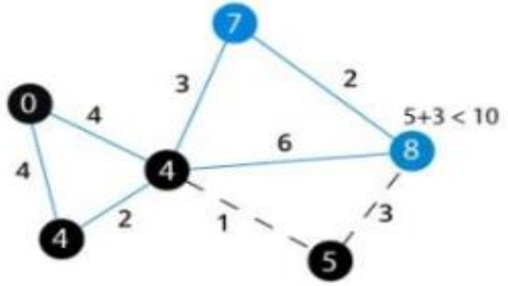
5

Avoid updating path lengths of already visited vertices



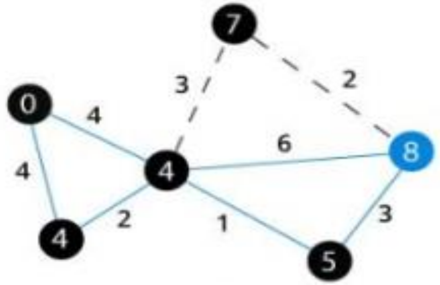
6

After each iteration, we pick the unvisited vertex with least path length. So we chose 5 before 7



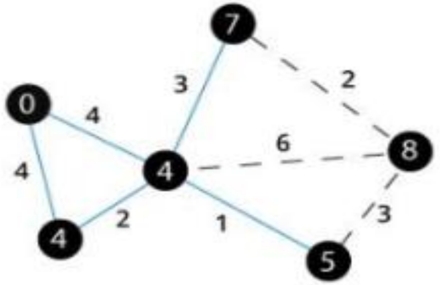
7

Notice how the rightmost vertex has its path length updated twice



8

Repeat until all the vertices have been visited



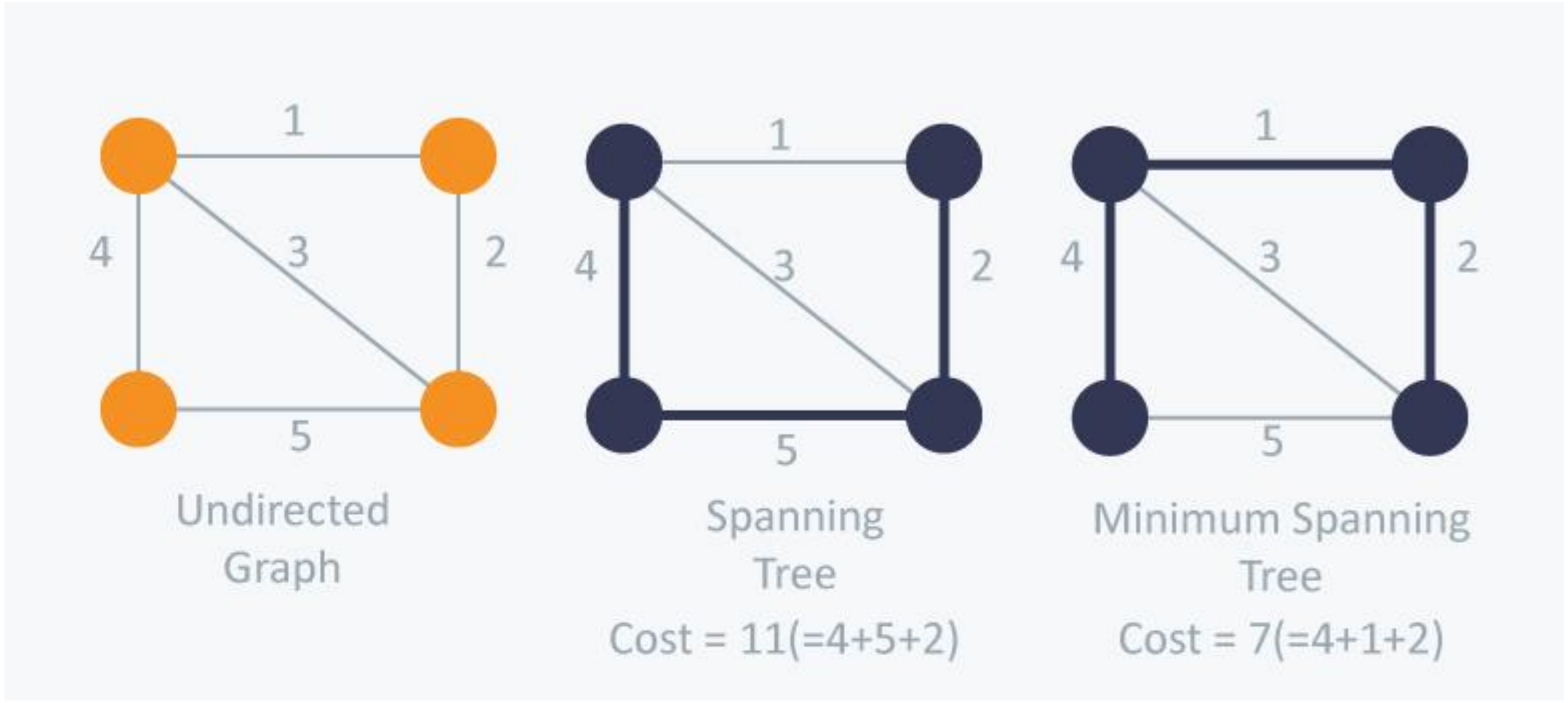
AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

MINIMUM SPANNING TREE

The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.

Minimum spanning tree has direct application in the design of networks. It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching. Other practical applications are:

1. Cluster Analysis
2. Handwriting recognition
3. Image segmentation


1
4
3
2
5
Undirected Graph
1
4
3
2
5
Spanning Tree
Cost = 11(=4+5+2)
1
4
3
2
5
Minimum Spanning Tree
Cost = 7(=4+1+2)

KRUSKAL’S ALGORITHM

Kruskal’s Algorithm builds the spanning tree by adding edges one by one into a growing spanning tree. Kruskal's algorithm follows greedy approach as in each iteration it finds an edge which has least weight and add it to the growing spanning tree.

Algorithm Steps:

- Sort the graph edges with respect to their weights.
- Start adding edges to the MST from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which doesn't form a cycle , edges which connect only disconnected components.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

So now the question is how to check if

2

 vertices are connected or not ?

This could be done using DFS which starts from the first vertex, then check if the second vertex is visited or not. But DFS will make time complexity large as it has an order of

O(V+E)

 where

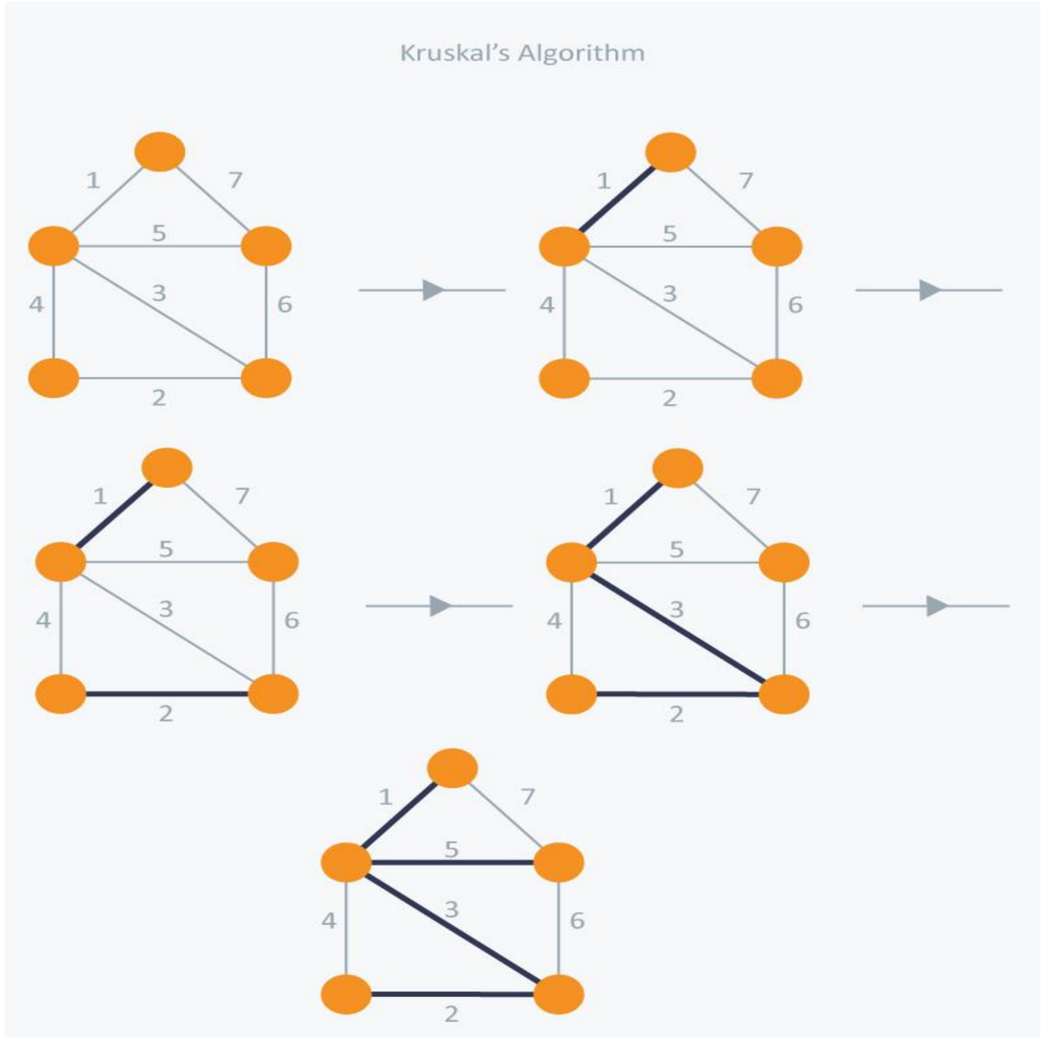
V

 is the number of vertices,

E

 is the number of edges. So the best solution is "Disjoint Sets":

Disjoint sets are sets whose intersection is the empty set so it means that they don't have any element in common.



COLLEGE OF ENGINEERING & TECHNOLOGY

In Kruskal’s algorithm, at each iteration we will select the edge with the lowest weight. So, we will start with the lowest weighted edge first i.e., the edges with weight 1. After that we will select the second lowest weighted edge i.e., edge with weight 2. Notice these two edges are totally disjoint. Now, the next edge will be the third lowest weighted edge i.e., edge with weight 3, which connects the two disjoint pieces of the graph. Now, we are not allowed to pick the edge with weight 4, that will create a cycle and we can’t have any cycles. So we will select the fifth lowest weighted edge i.e., edge with weight 5. Now the other two edges will create cycles so we will ignore them. In the end, we end up with a minimum spanning tree with total cost 11 (= 1 + 2 + 3 + 5).

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

PRIM’S ALGORITHM

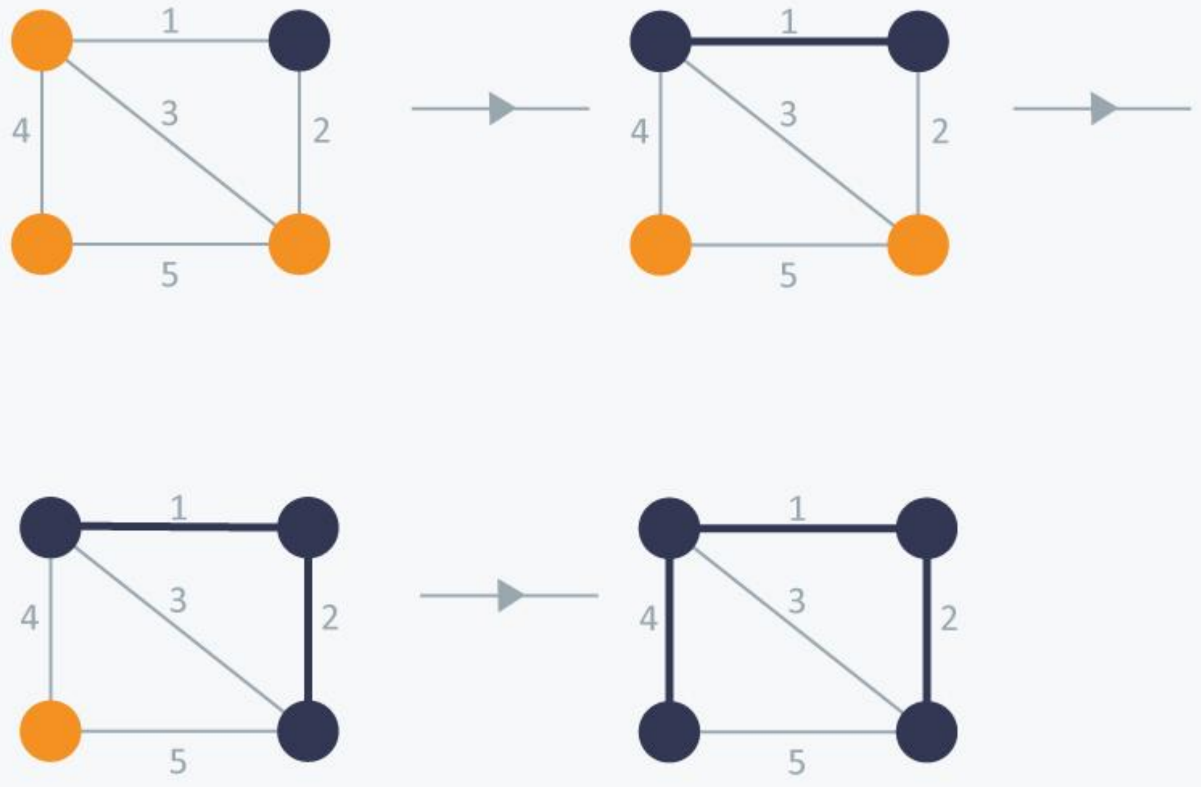
Prim’s Algorithm also use Greedy approach to find the minimum spanning tree. In Prim’s Algorithm we grow the spanning tree from a starting position. Unlike an edge in Kruskal's, we add vertex to the growing spanning tree in Prim's.

Algorithm Steps:

- Maintain two disjoint sets of vertices. One containing vertices that are in the growing spanning tree and other that are not in the growing spanning tree.
- Select the cheapest vertex that is connected to the growing spanning tree and is not in the growing spanning tree and add it into the growing spanning tree. This can be done using Priority Queues. Insert the vertices, that are connected to growing spanning tree, into the Priority Queue.
- Check for cycles. To do that, mark the nodes which have been already selected and insert only those nodes in the Priority Queue that are not marked.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Prim's Algorithm



In Prim’s Algorithm, we will start with an arbitrary node (it doesn’t matter which one) and mark it. In each iteration we will mark a new vertex that is adjacent to the one that we have already marked. As a greedy algorithm, Prim’s algorithm will select the cheapest edge and mark the vertex. So we will simply choose the edge with weight 1. In the next iteration we have three options, edges with weight 2, 3 and 4. So, we will select the edge with weight 2 and mark the vertex. Now again we have three options, edges with weight 3, 4 and 5. But we can’t choose edge with weight 3 as it is creating a cycle. So we will select the edge with weight 4 and we end up with the minimum spanning tree of total cost 7 (= 1 + 2 +4).

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY