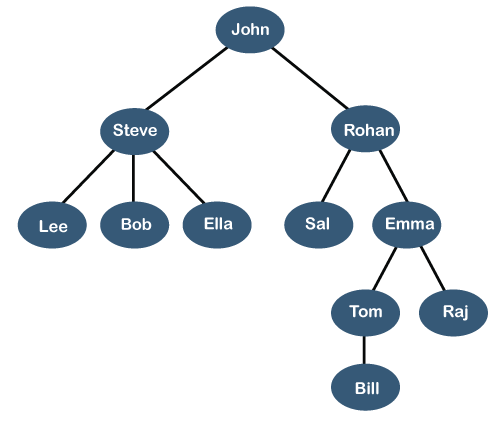
**Tree Data Structure**

We read the linear data structures like an array, linked list, stack and queue in which all the elements are arranged in a sequential manner. The different data structures are used for different kinds of data.

**Some factors are considered for choosing the data structure:**

* **What type of data needs to be stored**?: It might be a possibility that a certain data structure can be the best fit for some kind of data.
* **Cost of operations:** If we want to minimize the cost for the operations for the most frequently performed operations. For example, we have a simple list on which we have to perform the search operation; then, we can create an array in which elements are stored in sorted order to perform the **binary search**. The binary search works very fast for the simple list as it divides the search space into half.
* **Memory usage:** Sometimes, we want a data structure that utilizes less memory.

**A tree** is also one of the data structures that represent hierarchical data. Suppose we want to show the employees and their positions in the hierarchical form then it can be represented as shown below:



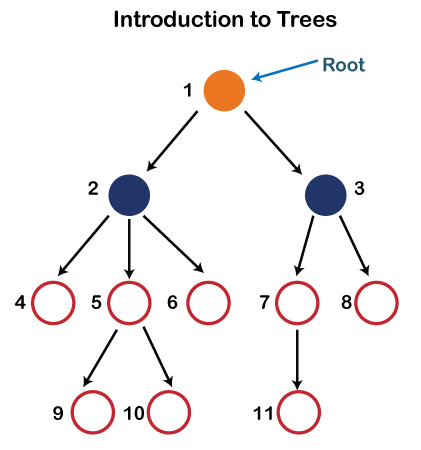
The above tree shows the **organization hierarchy** of some company. In the above structure, **john** is the **CEO** of the company, and John has two direct reports named as **Steve** and **Rohan**. Steve has three direct reports named **Lee, Bob, Ella** where **Steve** is a manager. Bob has two direct reports named **Sal** and **Emma**. **Emma** has two direct reports named **Tom** and **Raj**. Tom has one direct report named **Bill**. This particular logical structure is known as a **Tree**. Its structure is similar to the real tree, so it is named a **Tree**. In this structure, the **root** is at the top, and its branches are moving in a downward direction. Therefore, we can say that the Tree data structure is an efficient way of storing the data in a hierarchical way.

**Let's understand some key points of the Tree data structure.**

* A tree data structure is defined as a collection of objects or entities known as nodes that are linked together to represent or simulate hierarchy.
* A tree data structure is a non-linear data structure because it does not store in a sequential manner. It is a hierarchical structure as elements in a Tree are arranged in multiple levels.
* In the Tree data structure, the topmost node is known as a root node. Each node contains some data, and data can be of any type. In the above tree structure, the node contains the name of the employee, so the type of data would be a string.
* Each node contains some data and the link or reference of other nodes that can be called children.

**Some basic terms used in Tree data structure.**

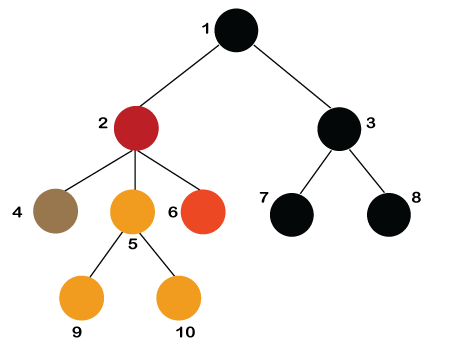
Let's consider the tree structure, which is shown below:



In the above structure, each node is labeled with some number. Each arrow shown in the above figure is known as a **link** between the two nodes.

* **Root:** The root node is the topmost node in the tree hierarchy. In other words, the root node is the one that doesn't have any parent. In the above structure, node numbered 1 is **the root node of the tree.** If a node is directly linked to some other node, it would be called a parent-child relationship.
* **Child node:** If the node is a descendant of any node, then the node is known as a child node.
* **Parent:** If the node contains any sub-node, then that node is said to be the parent of that sub-node.
* **Sibling:** The nodes that have the same parent are known as siblings.
* **Leaf Node:-** The node of the tree, which doesn't have any child node, is called a leaf node. A leaf node is the bottom-most node of the tree. There can be any number of leaf nodes present in a general tree. Leaf nodes can also be called external nodes.
* **Internal nodes:** A node has atleast one child node known as an **internal**
* **Ancestor node:-** An ancestor of a node is any predecessor node on a path from the root to that node. The root node doesn't have any ancestors. In the tree shown in the above image, nodes 1, 2, and 5 are the ancestors of node 10.
* **Descendant:** The immediate successor of the given node is known as a descendant of a node. In the above figure, 10 is the descendant of node 5.

### Properties of Tree data structure

* **Recursive data structure:** The tree is also known as a **recursive data structure**. A tree can be defined as recursively because the distinguished node in a tree data structure is known as a **root node**. The root node of the tree contains a link to all the roots of its subtrees. The left subtree is shown in the yellow color in the below figure, and the right subtree is shown in the red color. The left subtree can be further split into subtrees shown in three different colors. Recursion means reducing something in a self-similar manner. So, this recursive property of the tree data structure is implemented in various applications.  
  
* **Number of edges:** If there are n nodes, then there would n-1 edges. Each arrow in the structure represents the link or path. Each node, except the root node, will have atleast one incoming link known as an edge. There would be one link for the parent-child relationship.
* **Depth of node x:** The depth of node x can be defined as the length of the path from the root to the node x. One edge contributes one-unit length in the path. So, the depth of node x can also be defined as the number of edges between the root node and the node x. The root node has 0 depth.
* **Height of node x:** The height of node x can be defined as the longest path from the node x to the leaf node.

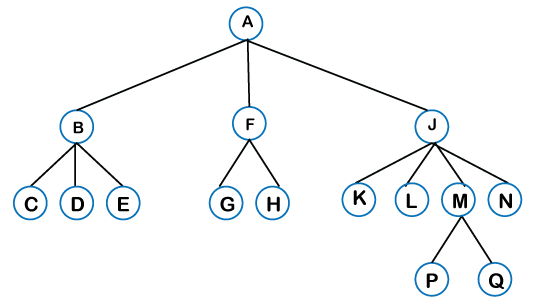
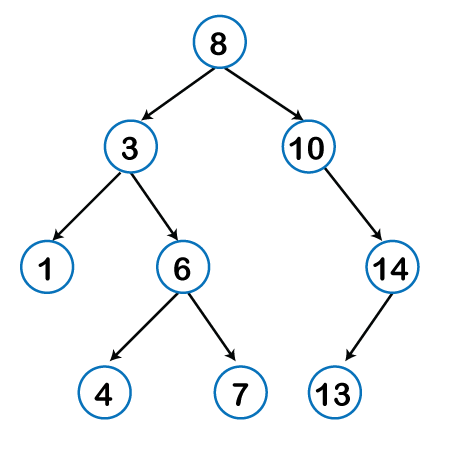
### Applications of trees

The following are the applications of trees:

* **Storing naturally hierarchical data:** Trees are used to store the data in the hierarchical structure. For example, the file system. The file system stored on the disc drive, the file and folder are in the form of the naturally hierarchical data and stored in the form of trees.
* **Organize data:** It is used to organize data for efficient insertion, deletion and searching. For example, a binary tree has a logN time for searching an element.
* **Trie:** It is a special kind of tree that is used to store the dictionary. It is a fast and efficient way for dynamic spell checking.
* **Heap:** It is also a tree data structure implemented using arrays. It is used to implement priority queues.
* **B-Tree and B+Tree:** B-Tree and B+Tree are the tree data structures used to implement indexing in databases.
* **Routing table:** The tree data structure is also used to store the data in routing tables in the routers.

### Types of Tree data structure

**The following are the types of a tree data structure:**

* **General tree:** The general tree is one of the types of tree data structure. In the general tree, a node can have either 0 or maximum n number of nodes. There is no restriction imposed on the degree of the node (the number of nodes that a node can contain). The topmost node in a general tree is known as a root node. The children of the parent node are known as **subtrees**.  
    
  There can be **n** number of subtrees in a general tree. In the general tree, the subtrees are unordered as the nodes in the subtree cannot be ordered.  
  Every non-empty tree has a downward edge, and these edges are connected to the nodes known as **child nodes**. The root node is labeled with level 0. The nodes that have the same parent are known as **siblings**.
* [**Binary tree**](https://www.javatpoint.com/binary-tree)**:** Here, binary name itself suggests two numbers, i.e., 0 and 1. In a binary tree, each node in a tree can have utmost two child nodes. Here, utmost means whether the node has 0 nodes, 1 node or 2 nodes.  
    
  [**Binary Search tree**](https://www.javatpoint.com/binary-search-tree)**:** Binary search tree is a non-linear data structure in which one node is connected to **n** number of nodes. It is a node-based data structure. A node can be represented in a binary search tree with three fields, i.e., data part, left-child, and right-child. A node can be connected to the utmost two child nodes in a binary search tree, so the node contains two pointers (left child and right child pointer).  
  Every node in the left subtree must contain a value less than the value of the root node, and the value of each node in the right subtree must be bigger than the value of the root node.
* [**AVL tree**](https://www.javatpoint.com/avl-tree)

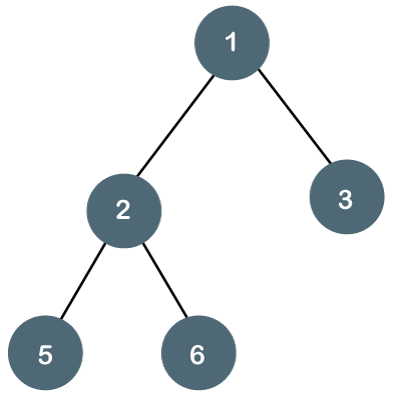
It is one of the types of the binary tree, or we can say that it is a variant of the binary search tree. AVL tree satisfies the property of the ***binary tree*** as well as of the ***binary search tree***. It is a self-balancing binary search tree that was invented by ***Adelson Velsky Lindas***. Here, self-balancing means that balancing the heights of left subtree and right subtree. This balancing is measured in terms of the ***balancing factor***.

We can consider a tree as an AVL tree if the tree obeys the binary search tree as well as a balancing factor. The balancing factor can be defined as the ***difference between the height of the left subtree and the height of the right subtree***. The balancing factor's value must be either 0, -1, or 1; therefore, each node in the AVL tree should have the value of the balancing factor either as 0, -1, or 1.

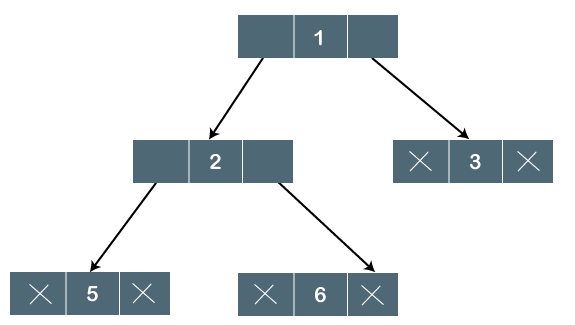
# Binary Tree

The Binary tree means that the node can have maximum two children. Here, binary name itself suggests that 'two'; therefore, each node can have either 0, 1 or 2 children.

**Let's understand the binary tree through an example.**



The above tree is a binary tree because each node contains the utmost two children. The logical representation of the above tree is given below:



In the above tree, node 1 contains two pointers, i.e., left and a right pointer pointing to the left and right node respectively. The node 2 contains both the nodes (left and right node); therefore, it has two pointers (left and right). The nodes 3, 5 and 6 are the leaf nodes, so all these nodes contain **NULL** pointer on both left and right parts.

### Properties of Binary Tree

* At each level of i, the maximum number of nodes is 2i.
* The height of the tree is defined as the longest path from the root node to the leaf node. The tree which is shown above has a height equal to 3. Therefore, the maximum number of nodes at height 3 is equal to (1+2+4+8) = 15. In general, the maximum number of nodes possible at height h is (20 + 21 + 22+….2h) = 2h+1 -1.
* The minimum number of nodes possible at height h is equal to **h+1**.
* If the number of nodes is minimum, then the height of the tree would be maximum. Conversely, if the number of nodes is maximum, then the height of the tree would be minimum.

If there are 'n' number of nodes in the binary tree.

**The minimum height can be computed as:**

As we know that,

n = 2h+1 -1

n+1 = 2h+1

Taking log on both the sides,

log2(n+1) = log2(2h+1)

log2(n+1) = h+1

**h = log2(n+1) - 1**

**The maximum height can be computed as:**

As we know that,

n = h+1

**h= n-1**

### Types of Binary Tree

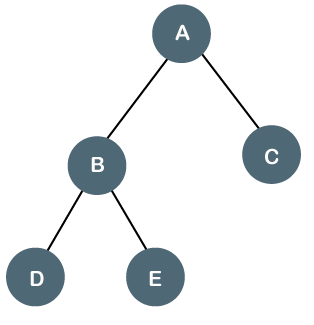
**There are four types of Binary tree:**

* **Full/ proper/ strict Binary tree**
* **Complete Binary tree**
* **Perfect Binary tree**
* **Degenerate Binary tree**
* **Balanced Binary tree**

**1. Full/ proper/ strict Binary tree**

The full binary tree is also known as a strict binary tree. The tree can only be considered as the full binary tree if each node must contain either 0 or 2 children. The full binary tree can also be defined as the tree in which each node must contain 2 children except the leaf nodes.

**Let's look at the simple example of the Full Binary tree.**



In the above tree, we can observe that each node is either containing zero or two children; therefore, it is a Full Binary tree.

**Properties of Full Binary Tree**

* The number of leaf nodes is equal to the number of internal nodes plus 1. In the above example, the number of internal nodes is 5; therefore, the number of leaf nodes is equal to 6.
* The maximum number of nodes is the same as the number of nodes in the binary tree, i.e., 2h+1 -1.
* The minimum number of nodes in the full binary tree is 2\*h-1.
* The minimum height of the full binary tree is **log2(n+1) - 1.**
* The maximum height of the full binary tree can be computed as:

n= 2\*h - 1

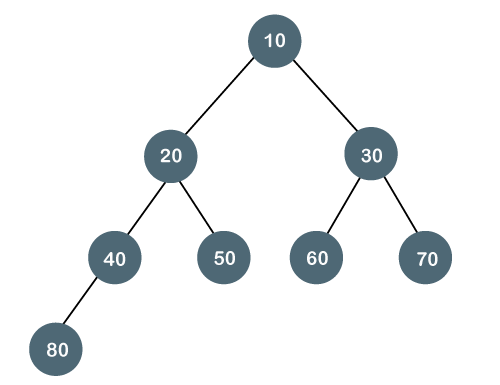
n+1 = 2\*h

**h = n+1/2**

**Complete Binary Tree**

The complete binary tree is a tree in which all the nodes are completely filled except the last level. In the last level, all the nodes must be as left as possible. In a complete binary tree, the nodes should be added from the left.

Let's create a complete binary tree.



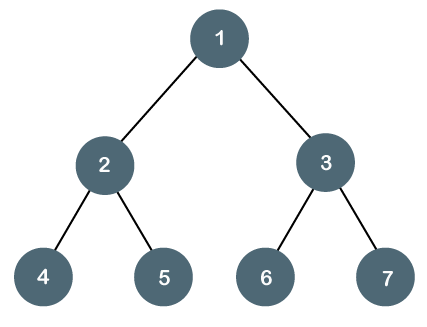
The above tree is a complete binary tree because all the nodes are completely filled, and all the nodes in the last level are added at the left first.

**Properties of Complete Binary Tree**

* The maximum number of nodes in complete binary tree is 2h+1 - 1.
* The minimum number of nodes in complete binary tree is 2h.
* The minimum height of a complete binary tree is **log2(n+1) - 1.**
* The maximum height of a complete binary tree is

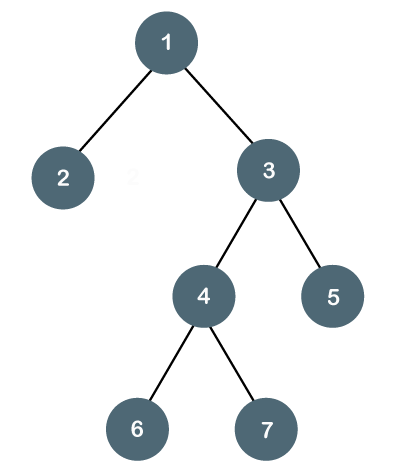
**Perfect Binary Tree**

A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.



**Let's look at a simple example of a perfect binary tree.**

The below tree is not a perfect binary tree because all the leaf nodes are not at the same level.

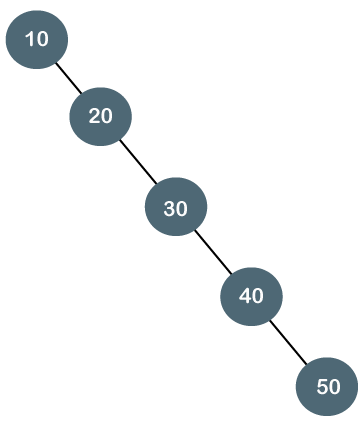


#### Note: All the perfect binary trees are the complete binary trees as well as the full binary tree, but vice versa is not true, i.e., all complete binary trees and full binary trees are the perfect binary trees.

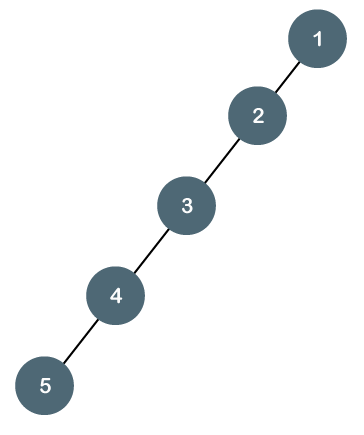
### Degenerate Binary Tree

The degenerate binary tree is a tree in which all the internal nodes have only one children.

**Let's understand the Degenerate binary tree through examples.**



The above tree is a degenerate binary tree because all the nodes have only one child. It is also known as a right-skewed tree as all the nodes have a right child only.

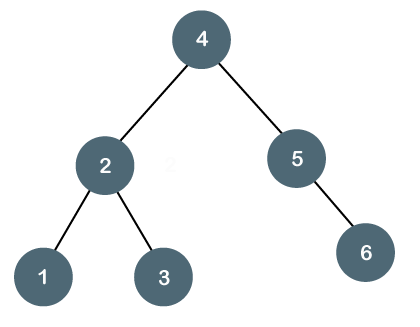


The above tree is also a degenerate binary tree because all the nodes have only one child. It is also known as a left-skewed tree as all the nodes have a left child only.

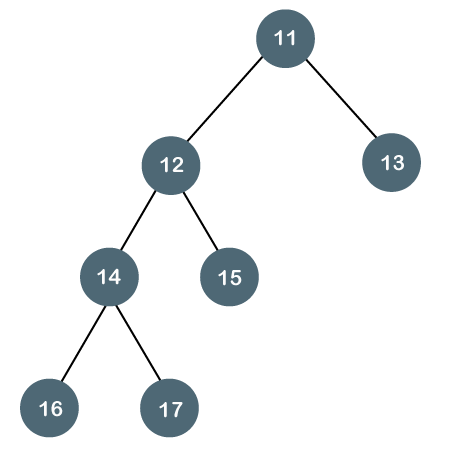
**Balanced Binary Tree**

The balanced binary tree is a tree in which both the left and right trees differ by atmost 1. For example, **AVL** and **Red-Black trees** are balanced binary tree.

**Let's understand the balanced binary tree through examples.**



The above tree is a balanced binary tree because the difference between the left subtree and right subtree is zero.



The above tree is not a balanced binary tree because the difference between the left subtree and the right subtree is greater than 1.

# What Is a Decision Tree?

A decision tree is a type of [supervised machine learning](https://www.mastersindatascience.org/learning/machine-learning-algorithms/) used to categorize or make predictions based on how a previous set of questions were answered. The model is a form of supervised learning, meaning that the model is trained and tested on a set of data that contains the desired categorization.

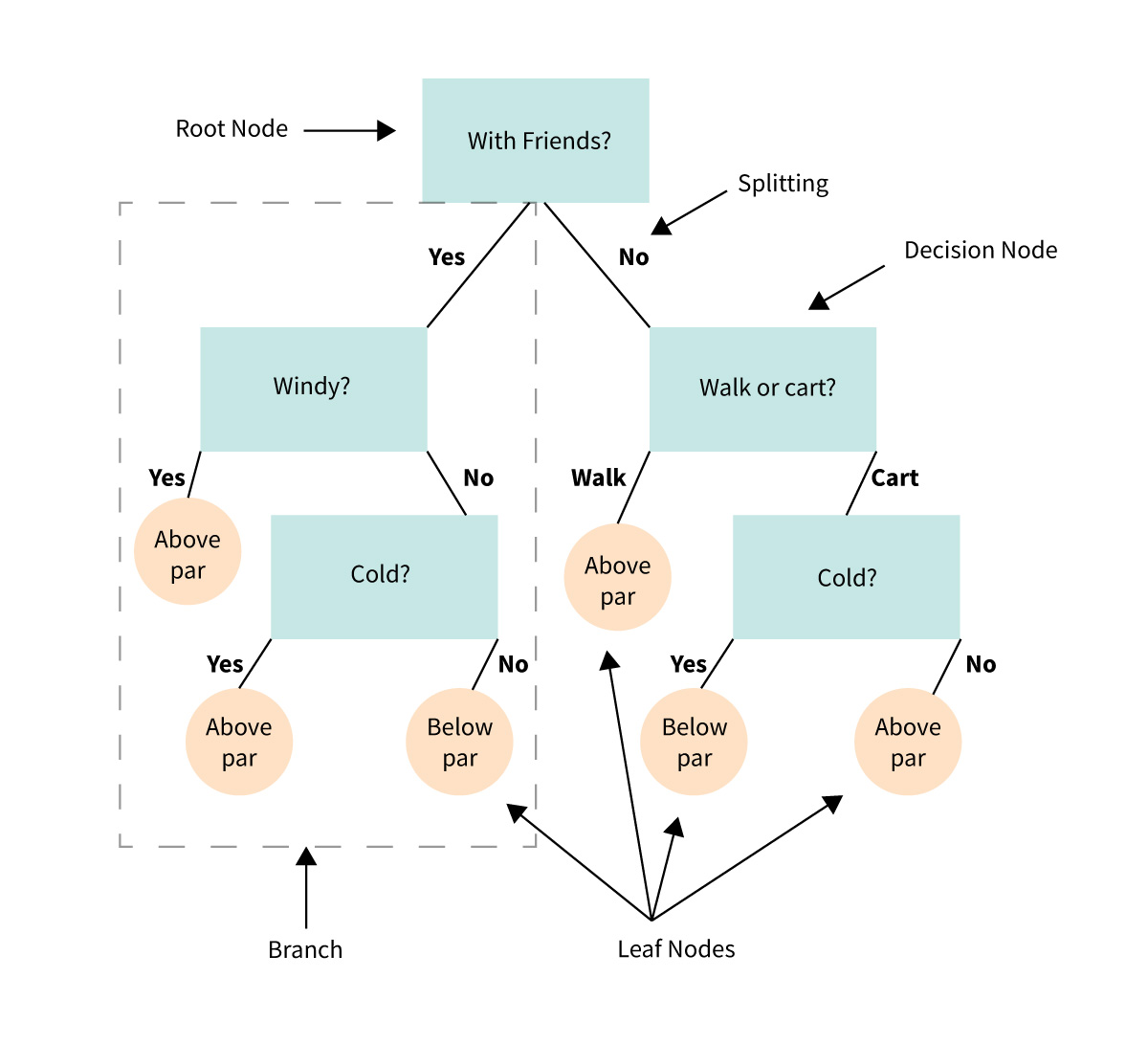
The decision tree may not always provide a clear-cut answer or decision. Instead, it may present options so the data scientist can make an informed decision on their own. Decision trees imitate human thinking, so it’s generally easy for data scientists to understand and interpret the results.

## How Does the Decision Tree Work?

key terms of a decision tree.

* **Root node:** The base of the decision tree.
* **Splitting:**The process of dividing a node into multiple sub-nodes.
* **Decision node:** When a sub-node is further split into additional sub-nodes.
* **Leaf node:** When a sub-node does not further split into additional sub-nodes; represents possible outcomes.
* **Pruning:** The process of removing sub-nodes of a decision tree.
* **Branch:**A subsection of the decision tree consisting of multiple nodes.

A decision tree resembles, well, a tree. The base of the tree is the root node. From the root node flows a series of decision nodes that depict decisions to be made. From the decision nodes are leaf nodes that represent the consequences of those decisions. Each decision node represents a question or split point, and the leaf nodes that stem from a decision node represent the possible answers. Leaf nodes sprout from decision nodes similar to how a leaf sprouts on a tree branch. This is why we call each subsection of a decision tree a “branch.” Let’s take a look at an example for this. You’re a golfer, and a consistent one at that. On any given day you want to predict where your score will be in two buckets: below par or over par.



While you are a consistent golfer, your score is dependent on a few sets of input variables. Wind speed, cloud cover and temperature all play a role. In addition, your score tends to deviate depending on whether or not you walk or ride a cart. And it deviates if you are golfing with friends or strangers.

In this example, there are two leaf nodes: below par or over par. Each of the input variables will determine decision nodes. Was it windy? Cold? Did you golf with friends? Did you walk or take a cart? With enough data on your golfing habits (and assuming you are a consistent golfer), a decision tree could help predict how you will do on the course on any given day.

**What is a Spanning Tree?**

Given an undirected and connected graph

, a spanning tree of the graph is a tree that spans (that is, it includes every vertex of ) and is a subgraph of (every edge in the tree belongs to

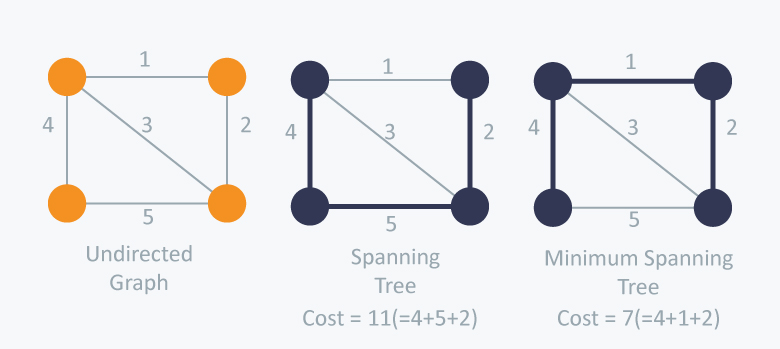
)

**What is a 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



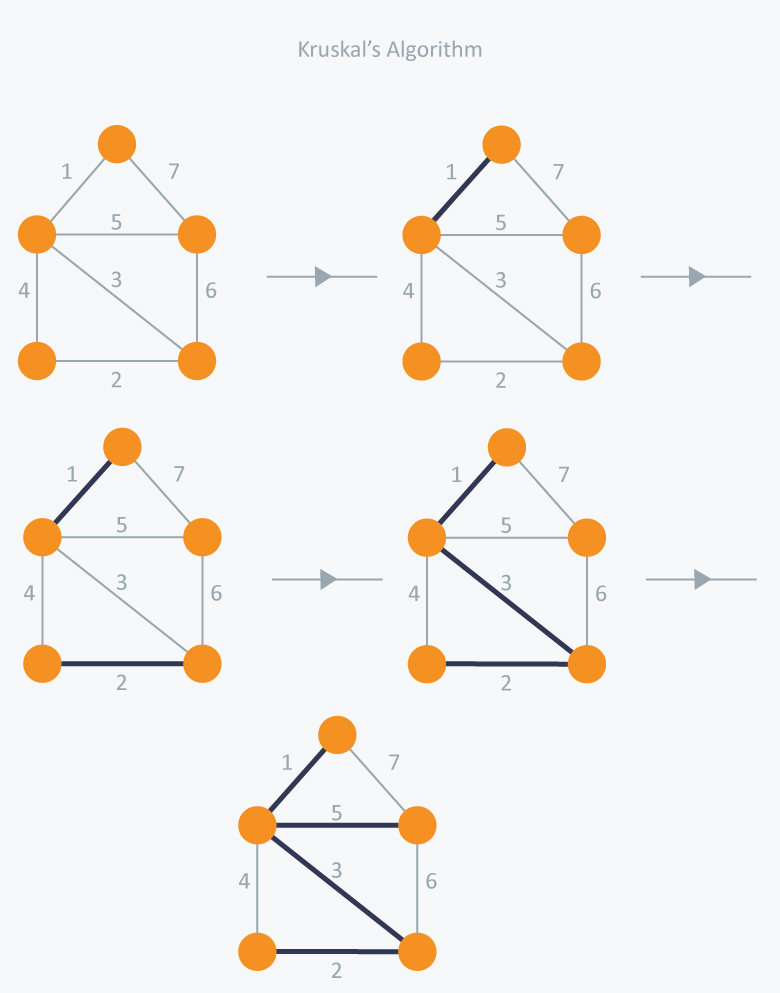
There are two famous algorithms for finding the Minimum Spanning Tree:

**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.

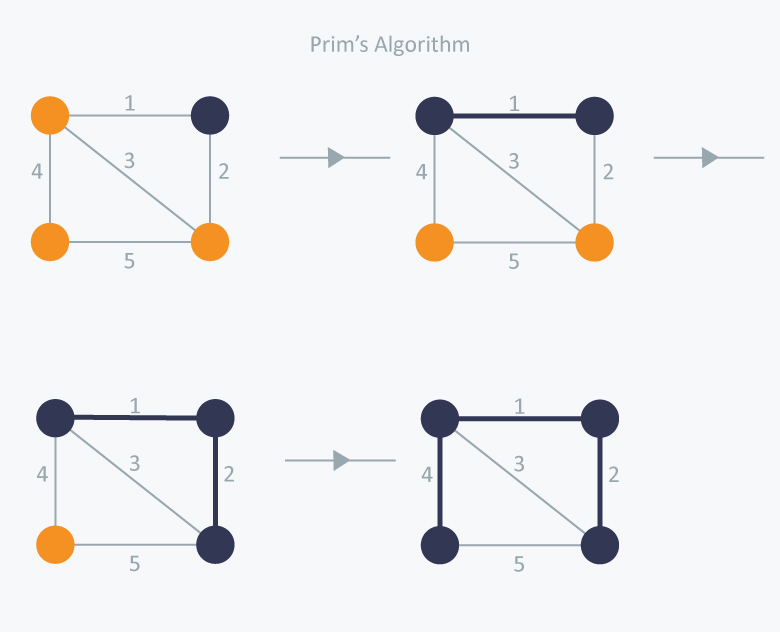


**Prim’s Algorithm**

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.



Lower Bound Theory(Decision Tree)Lower Bound Theory Concept is based upon the calculation of minimum time that is required to execute an algorithm is known as a lower bound theory or Base Bound Theory.Lower Bound Theory uses a number of methods/techniques to find out the lower bound.Aim:The main aim is to calculate a minimum number of comparisons required to execute an algorithm.Techniques:The techniques which are used by lower Bound Theory are:1.Comparisons Trees.2.Oracle and adversaryargument3.State Space Method