

National University of Computer and Emerging Sciences

CL 461 Artificial Intelligence

Lab Manual 04

Trees and Graphs in Python

TREE

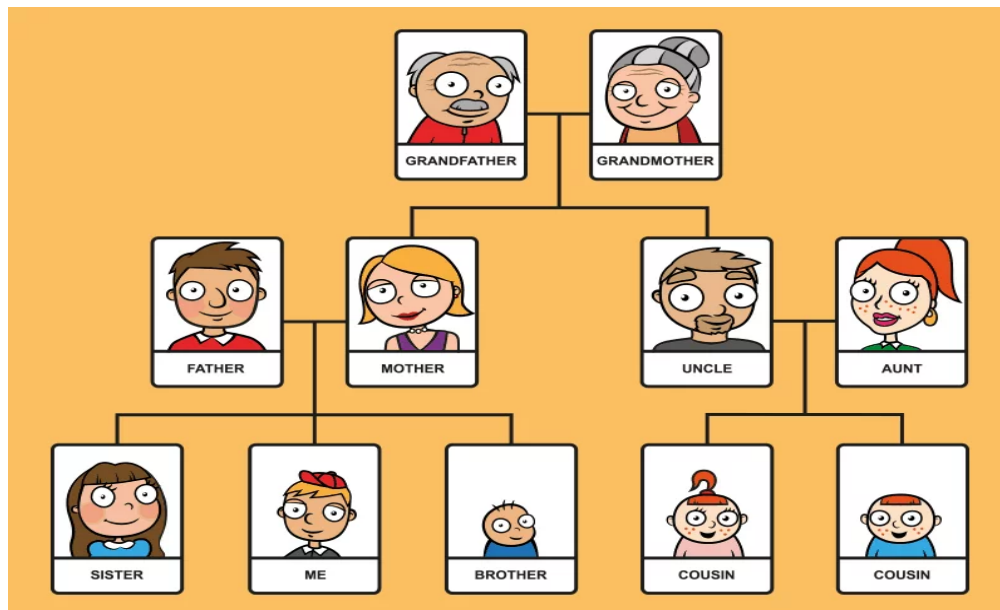
Definition:

Trees are well-known as a non-linear data structure. They don't store data in a linear way. They organize data hierarchically.

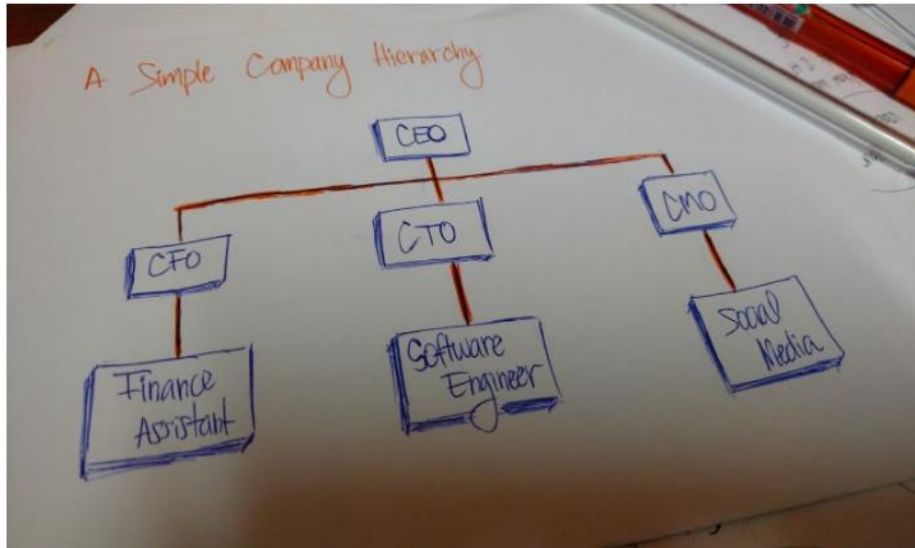
Let's dive into real life examples!

What do I mean when I say in a hierarchical way?

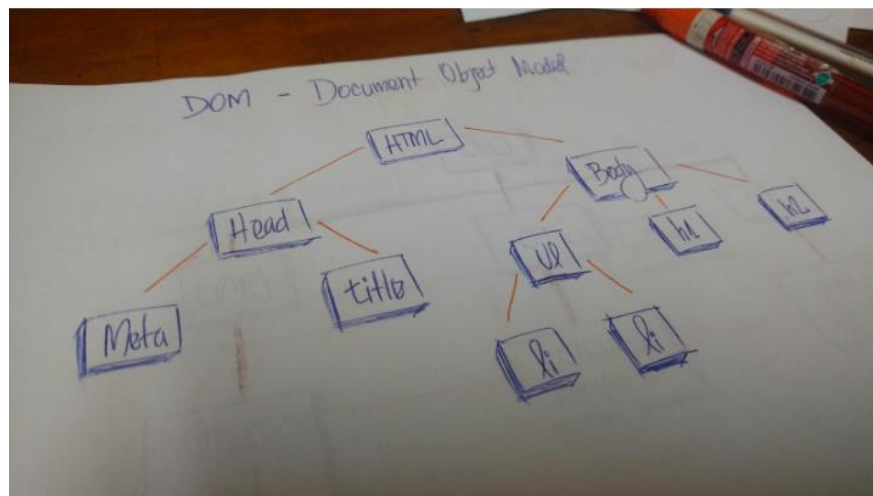
Imagine a family tree with relationships from all generations: grandparents, parents, children, siblings, etc. We commonly organize family trees hierarchically.



An organization's structure is another example of a hierarchy.



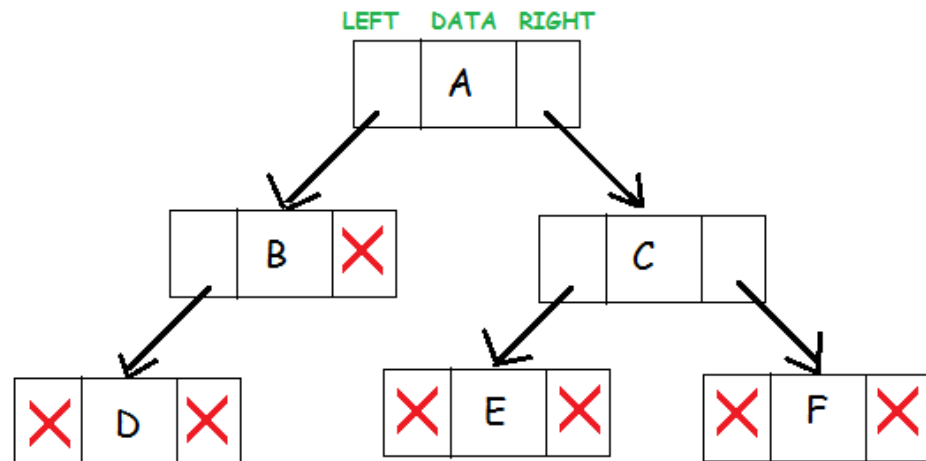
In HTML, the Document Object Model (DOM) works as a tree.



The **HTML** tag contains other tags. We have a **head** tag and a **body** tag. Those tags contain specific elements. The **head** tag has **meta** and **title** tags. The body tag has elements that show in the user interface, for example, **h1**, **a**, **li**, etc.

A technical definition

A tree is a collection of entities called nodes. Nodes are connected by edges. Each node contains a value or data, and it may or may not have a child node .



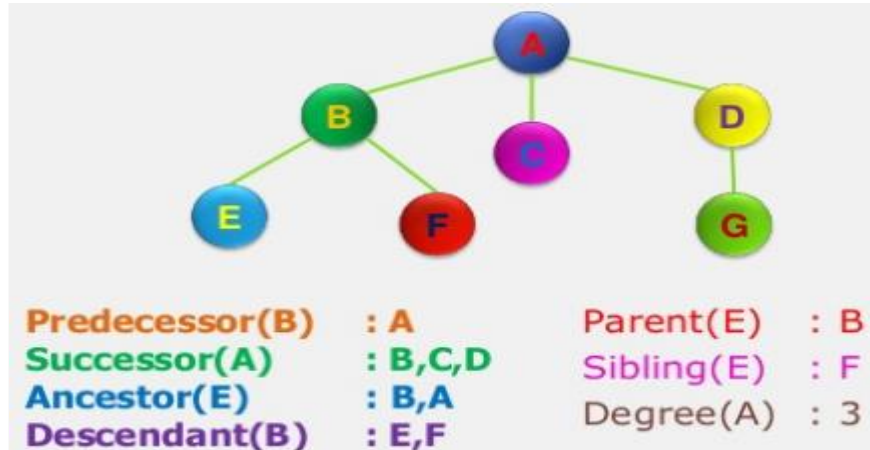
A binary tree node consists of the following components:

1. Data
2. Pointer to Left Child
3. Pointer to Right Child

Terminology summary

Below are some key terminologies related to a tree.

1. **Node:** The most elementary unit of a binary tree.
2. **Root:** The root of a tree is the topmost element.
3. **Edge:** The edge is the link between two nodes.
4. **Parent:** The parent of a node is the node that is one level upward of the node.
5. **Child:** The children of a node are the nodes that are one level downward of the node.
6. **Leaf:** The leaves of a binary tree are the nodes which have no children.
7. **Level:** The level is the generation of the respective node. The root has level 0, the children of the root node are at level 1, the grandchildren of the root node are at level 2 and so on.
8. **Height:** is the length of the longest path of a leaf.
9. **Depth:** is the length of the path to its root.
10. **Predecessor:** node that is above a certain node.
11. **Successor:** node that is below a certain node.
12. **Ancestor:** all nodes that are before a certain node and in the same path.
13. **Descendant:** all nodes that are after a certain node and in the same path.
14. **Sibling:** nodes that have the same parent.
15. **Degree:** number of children in one node.



Types of Tree

1. Binary Tree
2. Binary Search Tree

Application

1. Directory structure of a file store.
2. Structure of arithmetic expressions.
3. Used in almost every 3D video game to determine what objects need to be rendered.
4. Used in almost every high-bandwidth router for storing router-tables.
5. Used in compression algorithms, such as those used by the .jpeg and .mp3 file- formats.

FOR Further detail [Click Here](#)

Tree Traversal

1. inorder,
2. preorder and
3. postorder

Inorder traversal

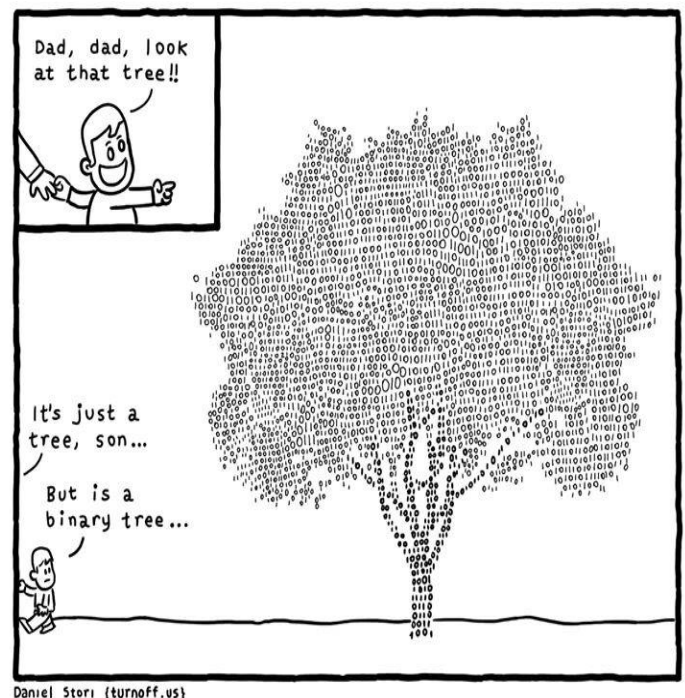
1. First, visit all the nodes in the left subtree
2. Then the root node
3. Visit all the nodes in the right subtree

Preorder traversal

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

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



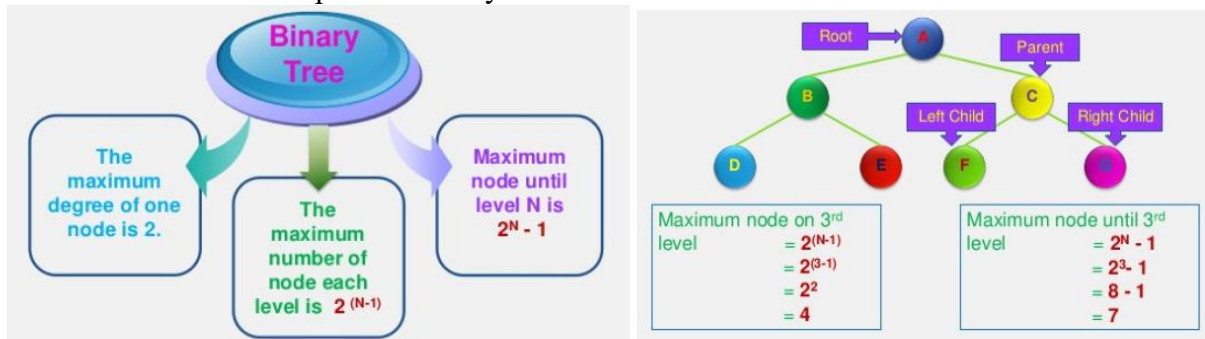
Binary trees

Now we will discuss a specific type of tree. We call it the **binary tree**.

“In computer science, a binary tree is a tree data structure in which each node has at the most two children, which are referred to as the left child and the right child.”—Wikipedia

“A Binary Search Tree is sometimes called ordered or sorted binary trees, and it keeps its values in sorted order, so that lookup and other operations can use the principle of binary search”—Wikipedia

So let's look at an example of a binary tree.



Types of Binary Tree

1. 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.

2. 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.

3. Complete Binary Tree

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

- Every level must be completely filled
- All the leaf elements must lean towards the left.
- 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.

4. Degenerate or Pathological Tree

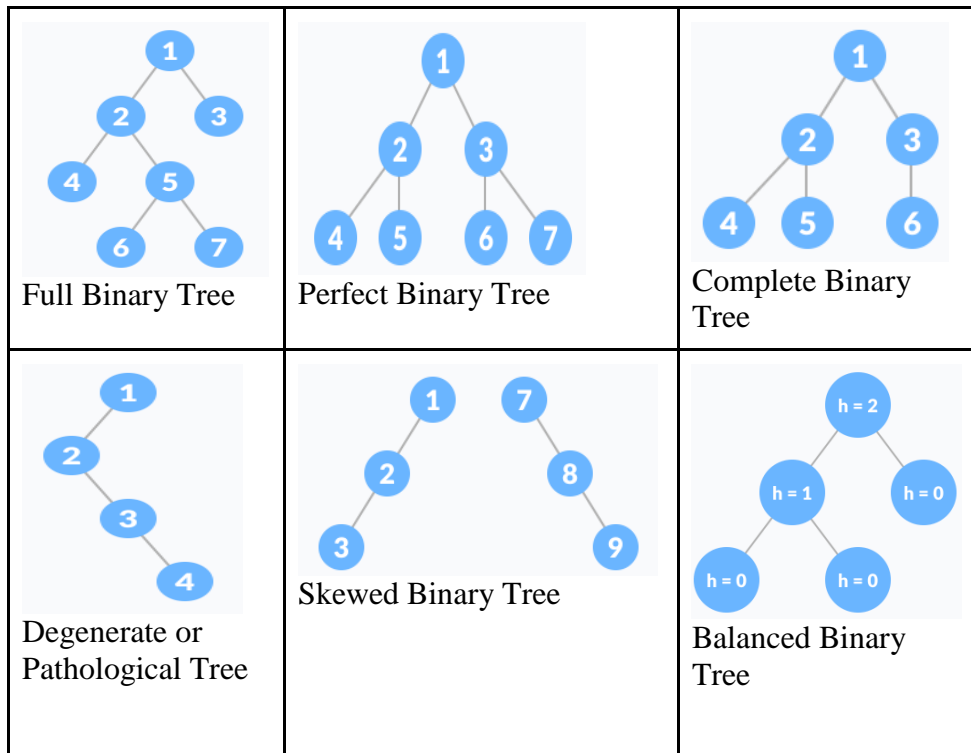
A degenerate or pathological tree is the tree having a single child either left or right.

5. 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.

6. 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.



Let's code a binary tree

The first thing we need to keep in mind when we implement a **binary search tree** is that it is a collection of nodes. Each node has three attributes: **value**, **left_child**, and **right_child**.

How do we implement a simple binary tree that initializes with above three properties?

We will see the binary tree and their implementation with all types through a notebook file present in google drive named as “**Week4_BT.ipynb**”.

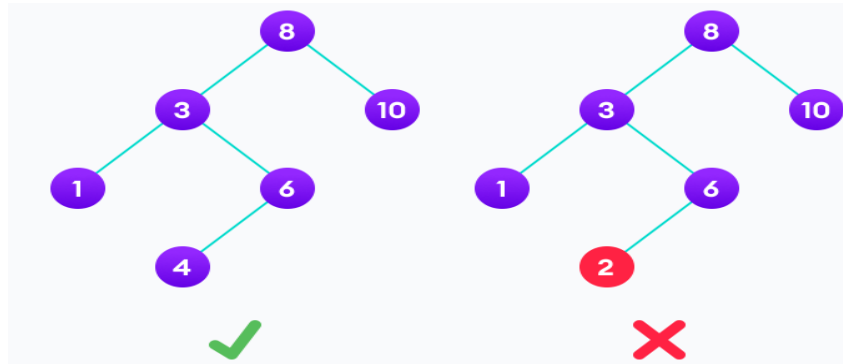
Binary Search Tree

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

1. It is called a binary tree because each tree node has a maximum of two children.
2. 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 separate a binary search tree from a regular binary tree is

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



There are two basic operations that you can perform on a binary search tree:

1. Search Operation

The algorithm depends on the property of BST that if each left subtree has values below root and each right subtree has values above the root.

If the value is below the root, we can say for sure that the value is not in the right subtree; we need to only search in the left subtree and if the value is above the root, we can say for sure that the value is not in the left subtree; we need to only search in the right subtree.

Algorithm:

```
If root == NULL
    return NULL;
If number == root->data
    return root->data;
If number < root->data
    return search(root->left)
If number > root->data
    return search(root->right)
```

2. Insert Operation

Inserting a value in the correct position is similar to searching because we try to maintain the rule that the left subtree is lesser than root and the right subtree is larger than root.

We keep going to either right subtree or left subtree depending on the value and when we reach a point left or right subtree is null, we put the new node there.

Algorithm:

```
If node == NULL
    return createNode(data)
if (data < node->data)
    node->left = insert(node->left, data);
else if (data > node->data)
    node->right = insert(node->right, data);
return node;
```


3. Delete Operation

There are three cases for deleting a node from a binary search tree.

Case I

In the first case, the node to be deleted is the leaf node. In such a case, simply delete the node from the tree.

Case II

In the second case, the node to be deleted has a single child node. In such a case follow the steps below:

1. Replace that node with its child node.
2. Remove the child node from its original position.

Case III

In the third case, the node to be deleted has two children. In such a case follow the steps below:

1. Get the inorder successor of that node.
2. Replace the node with the inorder successor.
3. Remove the inorder successor from its original position.

Let's code a binary search tree

We will see the BST and their implementation through a notebook file present in google drive named as “Week4_BT.ipynb”.

Binary tree Module in Python

In Python, a binary tree can be represented in different ways with different data structures (dictionary, list) and class representation for a node. However, binarytree library helps to directly implement a binary tree. It also supports heap and binary search tree(BST). This module does not come pre-installed with Python's standard utility module. To install it type the below command in the terminal.

Install via pip:

```
pip install binarytree
```

For conda users:

```
conda install binarytree -c conda-forge
```

Let's explore module from [here](#)

.

Let's code a binary tree using Library

We will see the BST and their implementation through a notebook file present in google drive named as “Week4_BT.ipynb”.

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.

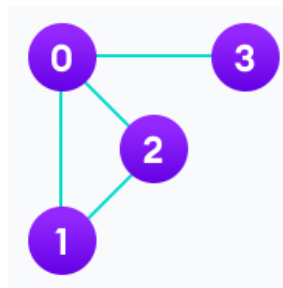
On facebook, everything is a node. That includes User, Photo, Album, Event, Group, Page, Comment, Story, Video, Link, Note...anything that has data is a node.

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.



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

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



In the graph,

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

Graph Terminology

1. **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.
2. **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.
3. **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.
4. **Undirected graph:** is a graph in which the edges do not point in any direction (ie. the edges are bidirectional).
5. **Connected graph:** is a graph in which there is always a path from a vertex to any other vertex.

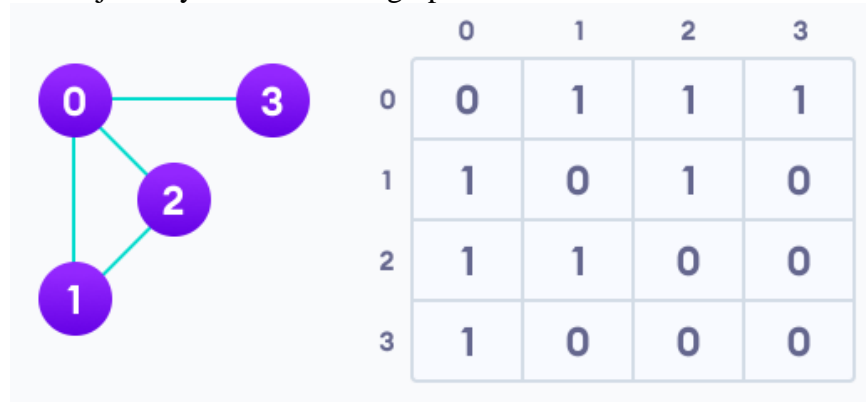
Graph Representation

Graphs are commonly represented in two ways:

1. Adjacency Matrix:

An adjacency matrix is a 2D array of $V \times V$ 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 .

The adjacency matrix for the graph we created above is

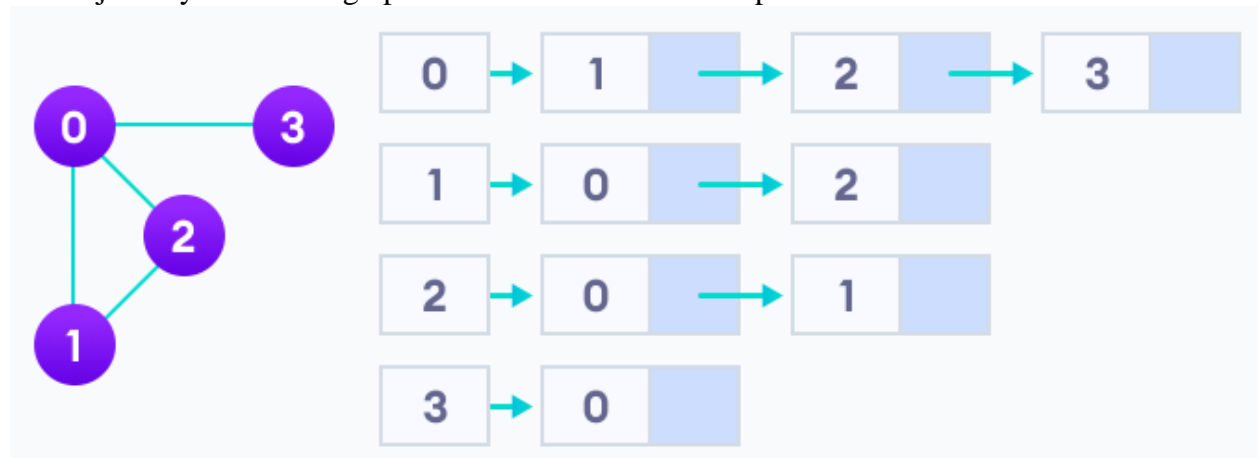


2. Adjacency List

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:



Graph Operations

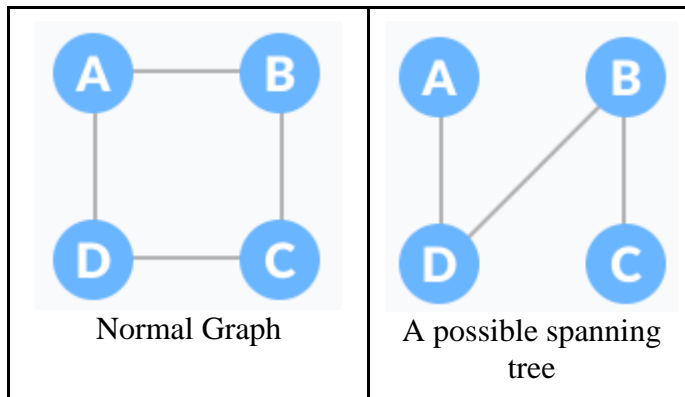
The most common graph operations are:

1. Check if the element is present in the graph
2. Graph Traversal

3. Add elements(vertex, edges) to graph
4. Finding the path from one vertex to another

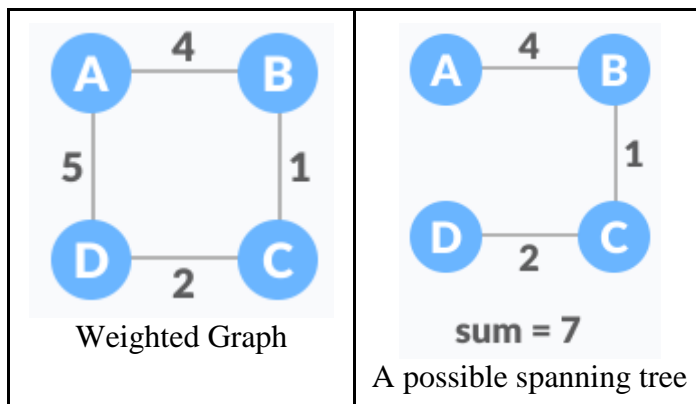
Spanning Tree

A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.



Minimum Spanning Tree

A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.



The minimum spanning tree from a graph is found using the following algorithms:

1. Prim's Algorithm
2. Kruskal's Algorithm

Prim's Algorithm

Prim's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

1. form a tree that includes every vertex
2. has the minimum sum of weights among all the trees that can be formed from the graph

How Prim's algorithm works

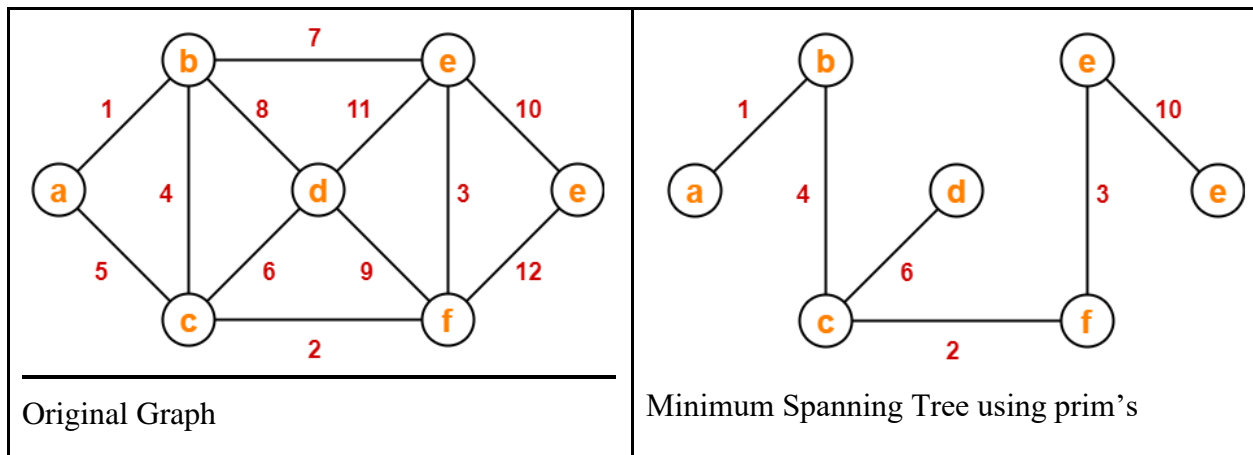
It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.

We start from one vertex and keep adding edges with the lowest weight until we reach our goal.

The steps for implementing Prim's algorithm are as follows:

1. Initialize the minimum spanning tree with a vertex chosen at random.
2. Find all the edges that connect the tree to new vertices, find the minimum and add it to the tree.
3. Keep repeating step 2 until we get a minimum spanning tree.

Example:



Kruskal's Algorithm

Kruskal's algorithm is a minimum spanning tree algorithm that takes a graph as input and finds the subset of the edges of that graph which

1. form a tree that includes every vertex
2. has the minimum sum of weights among all the trees that can be formed from the graph

How Kruskal's algorithm works

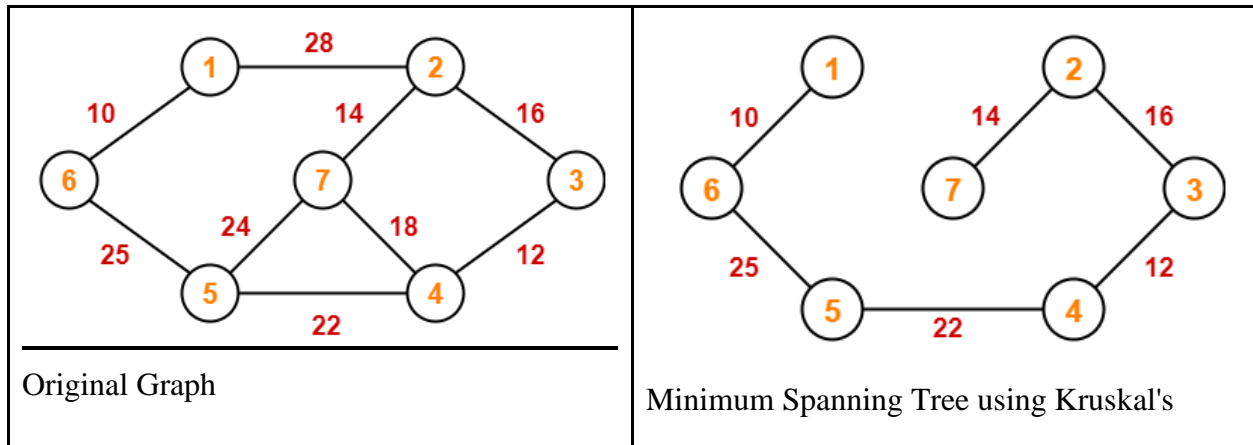
It falls under a class of algorithms called greedy algorithms that find the local optimum in the hopes of finding a global optimum.

We start from the edges with the lowest weight and keep adding edges until we reach our goal.

The steps for implementing Kruskal's algorithm are as follows:

1. Sort all the edges from low weight to high
2. Take the edge with the lowest weight and add it to the spanning tree. If adding the edge created a cycle, then reject this edge.
3. Keep adding edges until we reach all vertices.

Example:



Let's code a Prim's and Krukul's

We will see the Prim's and Krukul's and their implementation through a notebook file present in google drive named as **"Week4_Graph.ipynb"**.

MATPLOTLIB

What Is Python Matplotlib?

matplotlib.pyplot is a plotting library used for 2D graphics in python programming language. It can be used in python scripts, shell, web application servers and other graphical user interface toolkits.

What is Matplotlib used for?

Matplotlib is a Python Library used for plotting, this python library provides object oriented APIs for integrating plots into applications.

Is Matplotlib Included in Python?

Matplotlib is not a part of the Standard Libraries which is installed by default when Python, there are several toolkits which are available that extend python matplotlib functionality. Some of them are separate downloads, others can be shipped with the matplotlib source code but have external dependencies.

Types of Plots

There are various plots which can be created using python matplotlib. Some of them are listed below:



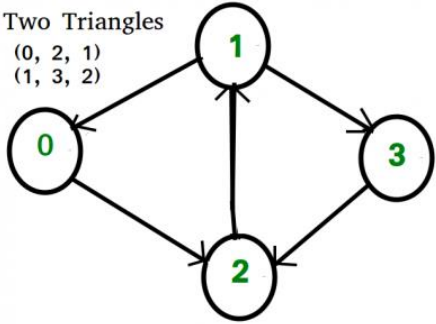
Let's explore Matplotlib

We will explore matplotlib through a notebook file present in google drive named as “matplotlib.ipynb”.

LAB EXERCISES

Question # 01:

Calculate Number of Triangles in Directed.

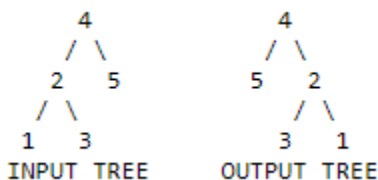
Give adjacency matrix	Give adjacency matrix represents following directed graph.
0: {0, 0, 1, 0}, 1: {1, 0, 0, 1}, 2: {0, 1, 0, 0}, 3: {0, 0, 1, 0}	<p>Two Triangles (0, 2, 1) (1, 3, 2)</p> 

Question # 02:

Write a recursive function `size(TreeNode* root)` that prints the number of elements in the tree.

Question # 03:

Write a recursive function `mirror` that swaps the left and right of the tree. Below trees provide an example input and output for `mirror()`



Question # 04:

This exercise contains five questions. Each question includes a specific Matplotlib.

month_number	facecream	facewash	toothpaste	bathingsoap	shampoo	moisturizer	total_units	total_profit
1	2500	1500	5200	9200	1200	1500	21100	211000
2	2630	1200	5100	6100	2100	1200	18330	183300
3	2140	1340	4550	9550	3550	1340	22470	224700
4	3400	1130	5870	8870	1870	1130	22270	222700
5	3600	1740	4560	7760	1560	1740	20960	209600
6	2760	1555	4890	7490	1890	1555	20140	201400
7	2980	1120	4780	8980	1780	1120	29550	295500
8	3700	1400	5860	9960	2860	1400	36140	361400
9	3540	1780	6100	8100	2100	1780	23400	234000
10	1990	1890	8300	10300	2300	1890	26670	266700
11	2340	2100	7300	13300	2400	2100	41280	412800
12	2900	1760	7400	14400	1800	1760	30020	300200

1. Read Total profit of all months and show it using a line plot.
2. Read all product sales data and show it using a multiline plot.
3. Read toothpaste sales data of each month and show it using a scatter plot.
4. Read face cream and facewash product sales data and show it using the bar chart.
5. Calculate total sale data for last year for each product and show it using a Pie chart.