Un GRAPH es una estructura de datos que se usa para representar relaciones entre datos o objetos.

Tiene NODOS que son los VÉRTICES y BORDES que se llaman ARISTAS, estas últimas conectan los nodos.

Un dibujo de una persona

Descripción generada automáticamente con confianza media

***GRAPH UNDIRECTED***

es cuando las aristas no tienen una dirección definida. Si no que es la relación es simétrica y bidimensional.

Un ejemplo de grafo no dirigido puede ser una red de amigos en una red social, donde cada nodo representa a un usuario y las aristas representan la amistad entre ellos.

Diagrama

Descripción generada automáticamente

***GRAPH DIRECTED***

Es cuando las aristas tienen una relación definida, ejemplo, el ***NODO A*** tiene una relación con el ***NODO B,*** pero no viceversa.

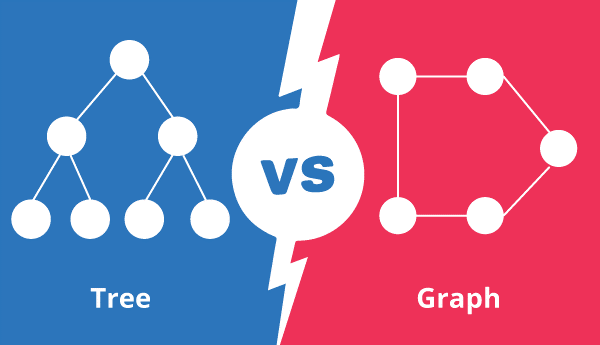
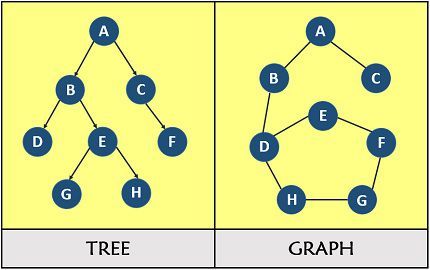
la redes de transporte, la planificación de tareas y la representación de flujos de información. Un ejemplo de grafo dirigido puede ser una red de carreteras, donde cada nodo representa una intersección y las aristas representan el tráfico que fluye desde una intersección a otra.

Diagrama

Descripción generada automáticamente

***¿Cuál Es La Diferencia Entre Graph Y Tree En Programación?***

1. Los Graph pueden tener cualquier numero de aristas y bordes, y pueden tener cualquier dirección, mientras que un Tree siempre tiene un único nodo raíz y cada nodo tiene un numero finito de hojas
2. Los tree no tienen caminos cerrados, por lo que cada nodo solo se alcanza por un único camino. Por esto son buenos para organización y búsqueda de datos.
3. Los graph no tienen un nodo raíz.
4. Los graph no pueden ser iterados con loops.

***WEIGHTED GRAPH***

Es un grafo en el que cada arista tiene un valor asignado. Este valor puede representar: [distancia, costo, tiempo, ect…]

Diagrama

Descripción generada automáticamente Diagrama

Descripción generada automáticamente con confianza baja

EN ESTE CASO HEMOS IMPLEMENTADO UN GRAPH DIRECTO DE CIUDADES.

class Graph():

    def \_\_init\_\_(self, edge):

        self.edge = edge

        self.graph\_dict = {}

        for start, end in self.edge:

            if start in self.graph\_dict:

                self.graph\_dict[start].append(end)

            else:

                self.graph\_dict[start] = [end]

        print("graph dict: ", self.graph\_dict)

routers = [

    ("Mumbai","Paris"),

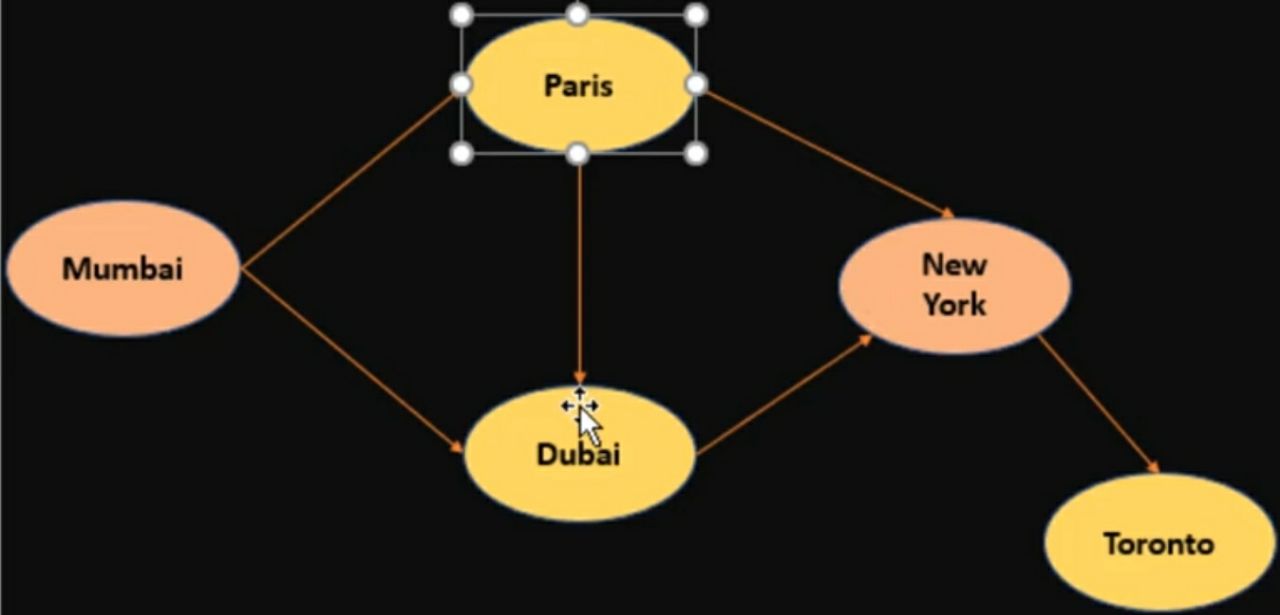
    ("Mumbai","Dubai"),

    ("Paris","Dubai"),

    ("Paris","New York"),

    ("Dubai","New York"),

    ("New York","Toronto")]



route\_graph = Graph(routers)



BUSCAR UNA RUTA

La recursividad de este código es que el END no cambia, siempre es el mismo, lo único que cambia es el START por el NODO, hasta que el NODO sea el mismo que el END, este código no deja de ejecutarse.

Una vez se ejecuta el segundo LOOP, lo que hace es agregar todos los valores que están en NEW\_PATHS los cuales se fueron agregando en cada recursión del código, en la lista que fue creada arriba del primer LOOP.

    def get\_paths(self, start, end, path=[]):

        path = path + [start]

        if start == end:

            return [path]

        if start not in self.graph\_dict:

            return []

        paths = []

        for node in self.graph\_dict[start]:

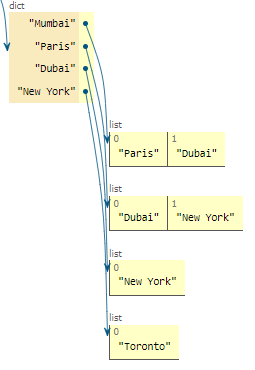
            if node not in path:

                new\_paths = self.get\_paths(node, end, path)

                for p in new\_paths:

                    paths.append(p)

        return paths

Es el mismo código, pero para que la ruta sea más corta:

    def get\_shorted\_paths(self, start, end, path=[]):

        path = path + [start]

        if start == end:

            return path

        if start not in self.graph\_dict:

            return None

        shortest\_paths = None

        for node in self.graph\_dict[start]:

            if node not in path:

                sp = self.get\_shorted\_paths(node, end, path)

                if sp:

                    if shortest\_paths is None or len(sp) < len(shortest\_paths):

                        shortest\_paths = sp

        return shortest\_paths