Código usado para contar altura da árvore

class Node:

def \_\_init\_\_(self, content):

self.content = content

self.left = None

self.right = None

def \_\_str\_\_(self):

return self.content

class BinaryTree:

def \_\_init\_\_(self):

self.root = None

def getRoot(self):

return self.root

def inOrder(self, root):

if root is None:

return

self.inOrder(root.left)

print(root)

self.inOrder(root.right)

def preOrder(self, root):

if root is None:

return

print(root)

self.preOrder(root.left)

self.preOrder(root.right)

def postOrder(self, root):

if root is None:

return

self.postOrder(root.left)

self.postOrder(root.right)

print(root)

def insert(self, root, newContent):

if root is None:

return Node(newContent)

if newContent < root.content:

root.left = self.insert(root.left, newContent)

else:

root.right = self.insert(root.right, newContent)

return root # importante!

##ira conta o tamanho da arvore a cada root novo##

def Countroot(self, root):

if root is None:

return 0

else:

return 1 + self.Countroot(root.left) + self.Countroot(root.right)

def height(self, root):

if root is None:

return 0

else:

left\_height = self.height(root.left)

right\_height = self.height(root.right)

return 1 + max(left\_height, right\_height)

tree = BinaryTree()

tree.root = tree.insert(tree.getRoot(), "zelia")

tree.root = tree.insert(tree.getRoot(), "Laura")

tree.root = tree.insert(tree.getRoot(), "Bia")

tree.root = tree.insert(tree.getRoot(), "Carla")

tree.root = tree.insert(tree.getRoot(), "Victor")

tree.root = tree.insert(tree.getRoot(), "Douglas")

tree.root = tree.insert(tree.getRoot(), "Maria")

tree.root = tree.insert(tree.getRoot(), "Paulo")

tree.root = tree.insert(tree.getRoot(), "Nara")

tree.root = tree.insert(tree.getRoot(), "Otavio")

tree.preOrder(tree.getRoot())

# Exemplo de uso: altura da árvore após todas as inserções

print('Altura da árvore:', tree.height(tree.getRoot()))

apresentado grafo, ea diferencia entre arvore

O grafo pode-se ter: ser direcionado, acontece loops entre ele, podendo ser mais dinâmico caso o caminho seja alterado.

Tarefa para fazer a modelagem de um grafo em python.

Codigo final da aula:

class Edge:

def \_\_init\_\_(self, target, weight):

self.target = target

self.weight = weight

class Vertex:

def \_\_init\_\_(self, name):

self.name = name

self.adjacents = []

def addEdge(self, target, weight):

self.adjacents.append(Edge(target, weight))

class Graph:

def \_\_init\_\_(self):

self.vertices = {}

def addVertex(self, name):

if name not in self.vertices:

self.vertices[name] = Vertex(name)

def addEdge(self, source, target, weight):

if source not in self.vertices:

self.addVertex(source)

if target not in self.vertices:

self.addVertex(target)

self.vertices[source].addEdge(target, weight)

def findPath(self, graph, source, target, visited = set(), path = []):

if source in visited:

return

visited.add(source)

path.append(source)

if source == target:

return path.copy()

for edge in self.vertices[source].adjacents:

result = self.findPath(graph, edge.target, target, visited, path)

if result:

return result

path.pop()

graph = Graph()

graph.addEdge("Palmas", "Porto", 10)

graph.addEdge("Palmas", "Paraiso", 10)

graph.addEdge("Paraiso", "Porto", 30)

print(graph.findPath(graph, "Palmas", "Porto"))

**Tarefa fazer um por meio de uso Grafo ou Árvore algo que seja legal divertido de trabalhar e tem que ser interativo**