ALGO 2

GRAFOS

ESTUDIANTE: JOAQUIN VILLEGAS

1-

```python
class GraphNode:
  vertex = None
  conectList = None
  #aca esta el cambio que decia: ¿Y si solo lo guardara en la primera posicion?
  EdgesListofGraph = None

  #Para BFSTree y DFSTree:
  color = None
  distance = None
  parent = None

  #especiales para DFSTree:
  timeD = None
  timeF = None


def createGraph(LV,LA):
  listAdyacencia = []
  for i in range(len(LV)):
    Node = GraphNode()
    Node.vertex = LV[i]
    #Node.conectList = []
    listAdyacencia.append(Node)

  for i in range(len(LV)):
    listAdyacencia[i].conectList = []
    for j in range(len(LA)):
      if LV[i] == LA[j][0]:
        listAdyacencia[i].conectList.append(LA[j][1])
      elif LV[i] == LA[j][1]:
        listAdyacencia[i].conectList.append(LA[j][0])
  listAdyacencia[0].EdgesListofGraph = LA
  return listAdyacencia

```

2-

```python
#Cada dia se aprende algo nuevo, voy a hacer uso del bloque try-except, veamos si funciona. Tengo entendido que no puedo realizar comparaciones con valueError, pero si crear una funcion que con try except (funciona como un if-else) y asi poder retornar lo que yo necesite.

def busquedaElemento(lista, elemento):
  try:
    indice = lista.index(elemento)
    return indice
  except ValueError:
    return None

#La idea de esta funcion es recibe un Nodo (primero seguramente que es el que tiene a v1),vamos a agregarlo a la lista de los vertices pasados (para asi no formar un bucle), me fijo si en su lista de conexiones ESTA el v2. Si no es asi, recorro toda la lista de conexiones de la siguiente forma, voy comprobando si esos vertices estan en la lista de vertices pasados, si no estan, voy a buscarlos en mi lista Grafo para asi realizar recursividad. si en el Search me devuelve un true, es porque lo encontro.
def existPathR(verticesPasados, Grafo, Nodo, v2):
  verticesPasados.append(Nodo.vertex)
  if busquedaElemento(Nodo.conectList,v2) != None:
    return True
  for verticesConectados in Nodo.conectList:
    if busquedaElemento(verticesPasados, verticesConectados) == None:
      siguienteVertice = verticesConectados
      for nuevoVertice in Grafo:
        if nuevoVertice.vertex == siguienteVertice:
          search = existPathR(verticesPasados, Grafo, nuevoVertice, v2)

          if search == True:
            return True
  return False

```

```python
def existPath(Grafo, v1, v2):
  myNode = None
  #aca al principio unos casos base, si el grafo no tiene vertices o si v1 y v2 son el mismo vertice.
  if Grafo == None or len(Grafo) == 0:
    return False

  if v1 == v2:
    return True
  #aca lo que hacemos es buscar el v1 en Grafo, si lo encontro defino una variable llamada myNode para asi luego, si es None retorna falso(porque nunca encontro a v1), y si no es asi sigue.
  for recorridoVertices in Grafo:
    if recorridoVertices.vertex == v1:
      myNode = recorridoVertices
      break
  if myNode == None:
    return False
  #aca tambien, podemos decir un caso base. si Justo en la lista de vertices conectados a v1 llega a estar v2, se retorna true.
  if busquedaElemento(myNode.conectList,v2) != None:
    return True

  listaVerticesPasados = []
  return existPathR(listaVerticesPasados, Grafo, myNode, v2)

print(existPath(graph, 1, 6))
```

3-

```python
def isConnected(Grafo):
  if Grafo == None or len(Grafo) == 0:
    return False

  for vertices in Grafo:
    for vertices2 in Grafo:
      if existPath(Grafo, vertices.vertex, vertices2.vertex) == False:
        return False
  return True
```

4-

```python
def isTree(Grafo):
  if Grafo == None or len(Grafo) == 0:
    return False

  if len(Grafo[0].EdgesListofGraph) == len(Grafo)-1 and isConnected(Grafo) == True:
    return True
  else:
    return False
print(isTree(graph))

```

5-

6-

```python
def convertTree(Grafo):
  Lista = []
  #voy a recorrer lo que seria mi Grafo con doble bucle, es decir que por ejemplo, para v1 voy a pasar por v2, v3, v4, v5 y comparar sus conexiones. Luego el siguiente seria comparar las conexiones de v2 con v1, v3, v4, v5, y asi sucesivamente.
  for vertice1 in Grafo:
    for vertice2 in Grafo:
      cont = 0
      subLista = []
      if vertice1 != vertice2:
        for conexionesv1 in vertice1.conectList:
          #el caso de donde justo es el vertice2.vertex y no dentro de su lista de conexiones
          if conexionesv1 == vertice2.vertex:
            cont += 1
            subLista.append((vertice1.vertex, conexionesv1))
          #el caso donde aca si hay que entrar a la lista de conexiones, aca debo de agregar dos duplas, uno con vertice1.vertex y el vertice2.vertex
          if busquedaElemento(vertice2.conectList, conexionesv1) != None:
            cont += 1
            subLista.append((vertice2.vertex, conexionesv1))
            subLista.append((vertice1.vertex, conexionesv1))
        #si el contador es mayor o igual a 2, es que estamos en caso de un ciclo
        if cont >= 2:
          Lista.append(subLista)
  return Lista
```

PARTE 2:

7-

```python
def countConnections(Grafo):
  #un caso base, si el grafo es conexo entonces solamente hay 1 componente
  if isConnected(Grafo) == True:
    return 1
  listaCC = []
  #bucles anidados para formar listas de componentes conexas
  for vertice1 in Grafo:
    subLista = []
    subLista.append(vertice1.vertex)
    for vertice2 in Grafo:
      if vertice1 != vertice2:
        if existPath(Grafo, vertice1.vertex, vertice2.vertex) == True:
          subLista.append(vertice2.vertex)
    #Ordeno la lista, para asi luego evitar repeticiones y tener bien contadas las componentes conexas
    subListaOrdenada = sorted(subLista)
    if busquedaElemento(listaCC, subListaOrdenada) == None:
      listaCC.append(subListaOrdenada)

  return len(listaCC)
```

8-

```python
def convertToBFSTree(Grafo, v):
  for vertice in Grafo:
    vertice.color = "Blanco"
    vertice.distance = -1
    vertice.parent = None

  v.color = "Gris"
  v.distance = 0
  v.parent = None
  colaVertices = []
  #para simular una cola, uso append y pop(0)
  colaVertices.append(v)
  while len(colaVertices) != 0:
    #RESOLVER ESTE PROBLEMA: Yo solamente en mi conectlist tengo guardados los vertices, pero no clase graphnode donde tengo los atributos a dar.
    verticeaUsar = colaVertices.pop(0)
    for vertice in verticeaUsar.conectList:
      for verticeConexo in Grafo:
        if verticeConexo.vertex == vertice:
          break

      if verticeConexo.color == "Blanco":
        verticeConexo.color = "Gris"
        verticeConexo.distance = verticeaUsar.distance + 1
        verticeConexo.parent = verticeaUsar
        colaVertices.append(verticeConexo)
    verticeaUsar.color = "Negro"

```

9-

```python
#Modificaciones que realice es que DFSVisit me retorna un int, que es el tiempo.
def convertToDFSTree(Grafo, v):
  for vertice in Grafo:
    vertice.color = "Blanco"
    vertice.parent = None

  tiempo = 0
  DFSVisit(Grafo, v, tiempo)
  #En caso de que hayan quedado Vertices en blanco, CREO que es por esta razon la que hace un bucle en las siguientes lineas
  for vertice in Grafo:
    if vertice.color == "Blanco":
      DFSVisit(Grafo, vertice, tiempo)
#Funcion Recursiva
def DFSVisit(Grafo, v, tiempo):
  tiempo += 1
  v.timeD = tiempo
  v.color = "Gris"
  for vertice in v.conectList:
    flag = False
    for vertice2 in Grafo:
      if (vertice2.vertex == vertice) and (vertice2.color == "Blanco"):
        flag = True
        break
    if vertice2.color == "Blanco" and flag == True:
      vertice2.parent = v
      tiempo = DFSVisit(Grafo, vertice2, tiempo)
  v.color = "Negro"
  tiempo += 1
  v.timeF = tiempo
  return tiempo

```

10-

```python
def bestRoad(Grafo, v1, v2):
  #Caso base, nos aseguramos si existe un camino entre v1 y v2
  if existPath(Grafo,v1, v2) == False:
    return []

  Lista = []
  for vertice in Grafo:
    if vertice.vertex == v1:
      verticeaUsar = vertice
  #Otro caso base, donde justo en los vertices adyacentes a v1, se encuentra el v2
  if busquedaElemento(verticeaUsar.conectList, v2) != None:
    Lista.append(v1)
    Lista.append(v2)
    return Lista

  #Hago la recursividad por la cantidad de vertices que son adyacentes a v1(?
  for vertices in verticeaUsar.conectList:
    listaFinal = bestRoadR(Grafo, verticeaUsar, v2, [], Lista,[])

  #Aca de todas los caminos posibles, tomo el mas pequeño:
  menor = 100000
  for posibleCamino in listaFinal:
    if len(posibleCamino) < menor:
      menor = len(posibleCamino)
      caminoMasCorto = posibleCamino
  return caminoMasCorto
```

```python
def bestRoadR(Grafo, verticeEnUso, v2, subLista, Lista, listaVerticesPasados):
  listaVerticesPasados.append(verticeEnUso.vertex)
  subLista.append(verticeEnUso.vertex)
  if busquedaElemento(verticeEnUso.conectList, v2) != None:
    subLista.append(v2)
    Lista.append(subLista)
    return Lista

  for numVertice in verticeEnUso.conectList:
    for verticeNodo in Grafo:
      if verticeNodo.vertex == numVertice and busquedaElemento(listaVerticesPasados, verticeNodo.vertex) == None:
        bestRoadR(Grafo, verticeNodo, v2, list(subLista), Lista, list(listaVerticesPasados))

  return Lista
```