UNIVERSIDAD AUTÓNOMA DE CAMPECHE

FACULTAD DE INGENIERIA

INGÉNIERIA EN SISTEMAS COMPUTACIONALES

UNIDAD DE APRENDIZAJE: **INTELIGENCIA ARTIFICIAL** (6ER. SEMESTRE GRUPO B)

ALUMNO(A): **HERNANDEZ CRISANTO, BRAYAN RAFAEEL**

NOMBRE DEL TRABAJO: ***Distintos caminos para ir de A a F***

FECHA DE ENTREGA: 14 DE ENERO 2022

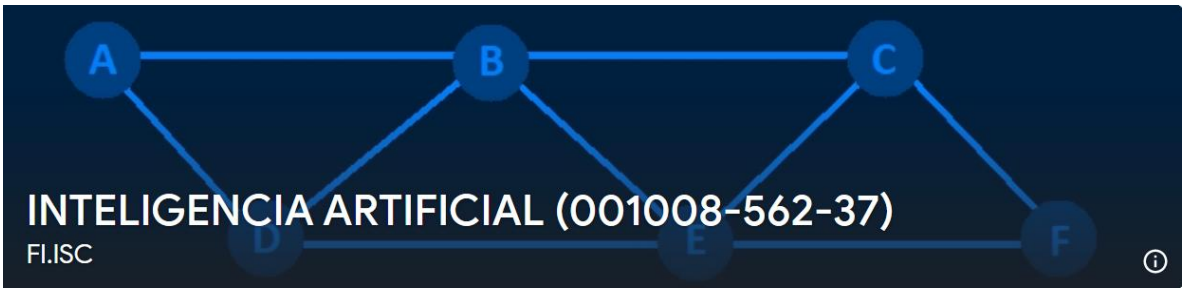
PROFESOR(A): JUAN J MONCADA BOLON

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

FACULTAD DE INGENIERIA

INGENIERIA EN SISTEMAS COMPUTACIONALES

Ejemplo tarea:




```
[Running] python -u "c:\Users\Odin17\Desktop\python\Imgen_Rafael Hernandez.py"
Caminos no c◆clicos posibles desde "A" a "F" usando busqueda en profundidad:
[['A', 'B', 'C', 'E', 'F'], ['A', 'B', 'C', 'F'], ['A', 'B', 'D', 'E', 'C', 'F'], ['A', 'B', 'D', 'E', 'F'], ['A', 'B', 'E', 'C', 'F'], ['A', 'B', 'E', 'F'], ['A', 'D'

Caminos no c◆clicos posibles desde "A" a "F" usando busqueda en en anchura:
[['A', 'B', 'C', 'F'], ['A', 'B', 'C', 'E', 'F'], ['A', 'B', 'E', 'F'], ['A', 'B', 'E', 'C', 'F'], ['A', 'B', 'D', 'E', 'F'], ['A', 'B', 'D', 'E', 'C', 'F'], ['A', 'D'
```

Caminos no cíclicos posibles desde "A" a "F" usando búsqueda en profundidad:

[['A', 'B', 'C', 'E', 'F'], ['A', 'B', 'C', 'F'], ['A', 'B', 'D', 'E', 'C', 'F'], ['A', 'B', 'D', 'E', 'F'], ['A', 'B', 'E', 'C', 'F'], ['A', 'B', 'E', 'F'], ['A', 'D', 'B', 'C', 'E', 'F'], ['A', 'D', 'B', 'C', 'F'], ['A', 'D', 'B', 'E', 'C', 'F'], ['A', 'D', 'B', 'E', 'F'], ['A', 'D', 'E', 'B', 'C', 'F'], ['A', 'D', 'E', 'C', 'F'], ['A', 'D', 'E', 'F']]

Caminos no cíclicos posibles desde "A" a "F" usando búsqueda en anchura:

[['A', 'B', 'C', 'F'], ['A', 'B', 'C', 'E', 'F'], ['A', 'B', 'E', 'F'], ['A', 'B', 'E', 'C', 'F'], ['A', 'B', 'D', 'E', 'F'], ['A', 'B', 'D', 'E', 'C', 'F'], ['A', 'D', 'E', 'F'], ['A', 'D', 'E', 'C', 'F'], ['A', 'D', 'E', 'B', 'C', 'F'], ['A', 'D', 'B', 'C', 'F'], ['A', 'D', 'B', 'C', 'E', 'F'], ['A', 'D', 'B', 'E', 'F'], ['A', 'D', 'B', 'E', 'C', 'F']]

Sol en Python:

```python
import collections
class Vertice:

    def __init__(self, n):
        self.nombre = n
        self.vecinos = list()
        self.distancia = 9999
        self.color = 'white'
        self.pred = -1

    def agregarVecino(self, v):
        if v not in self.vecinos:
            self.vecinos.append(v)
            self.vecinos.sort()
```

```
class Grafo:
    def __init__(self):
        self.vertices = dict()

    def agregarVertices(self, vertices):
        for v in vertices:
            n = Vertice(v)
            self.agregarVertice(n)

    def agregarAristas(self, aristas):
        for arista in aristas:
            self.agregarArista(arista[0], arista[1])

    def agregarVertice(self, vertice):
        if isinstance(vertice, Vertice) and vertice.nombre not in
self.vertices:
            self.vertices[vertice.nombre] = vertice
            return True
        else:
            return False

    def agregarArista(self, u, v):
        if u in self.vertices and v in self.vertices:
            for key, value in self.vertices.items():
                if key == u:
                    value.agregarVecino(v)
                if key == v:
                    value.agregarVecino(u)
            return True
        else:
            return False

    def bfs(self, vert):
        vert = self.vertices[vert]
        vert.distancia = 0
        vert.color = 'gray'
        vert.pred = -1
        q = list()

        q.append(vert.nombre)
```

```
        while len(q) > 0:

            u = q.pop()
            node_u  = self.vertices[u]
            for v in node_u.vecinos:
                node_v = self.vertices[v]
                if node_v.color == 'white':
                    node_v.color = 'gray'
                    node_v.distancia = node_u.distancia + 1
                    node_v.pred = node_u.nombre
                    q.append(v)
            self.vertices[u].color = 'black'

    def imprimeGrafo (self):
        for key in sorted(list(self.vertices.keys())):
            print ("Vertice " + key + " sus vecinos son "+
str(self.vertices[key].vecinos) )
            print("La distancia de A a " + key + " es: "+
str(self.vertices[key].distancia))
            print()

    def dfs_paths(self, vertice_inicial, vertice_final, camino=None):
        if camino == None:
            camino = [vertice_inicial]
        if vertice_inicial == vertice_final:
            yield camino
        for vertice in (v for v in self.vertices[vertice_inicial].vecinos if
v not in set(camino)):
            yield from self.dfs_paths(vertice, vertice_final, camino +
[vertice])


    def bfs_paths(self, vertice_inicial, vertice_final):
        cola = collections.deque()
        cola.append((vertice_inicial, [vertice_inicial]))
        while cola:
            (v, camino) = cola.pop()
            for vertice in set(self.vertices[v].vecinos) - set(camino):
                if vertice == vertice_final:
                    yield camino + [vertice]
                else:
                    cola.append((vertice, camino + [vertice]))
```

```
#Creamos un grafo como en su ejemplo profe:
vertices = [chr(i) for i in range(ord('A'), ord('K'))]
edges = ['AB','AD','BD','BE','BC','DB','DE','EC','EB','EF','CE','CF']
g1 = Grafo()
g1.agregarVertices(vertices)
g1.agregarAristas(edges)
g1.bfs('A')

print('Caminos no cíclicos posibles desde "A" a "F" usando busqueda en
profundidad:')
caminos = g1.dfs_paths('A','F')
print(list(caminos))
print('\nCaminos no cíclicos posibles desde "A" a "F" usando busqueda en en
anchura:')
caminos = g1.bfs_paths('A','F')
print(list(caminos))
```