COMILLAS UNIVERSIDAD PONTIFICIA ICAI DEPARTAMENTO DE MATEMÁTICA APLICADA

Matemática Discreta - IMAT
Práctica de NetworkX

In [ ]:
```
import networkx as nx
import matplotlib.pyplot as plt
```

Grafos no dirigidos

In [ ]:
```
G=nx.Graph()
G.add_node(1)
G.add_node(2)
G.add_edge(1,2)
G.add_nodes_from([3,4,5])
G.add_edges_from([(1,4),(3,4),(2,4),(3,2),(3,5)])

print(G)
print(G.nodes)

print(G.edges)
```

```
Graph with 5 nodes and 6 edges
[1, 2, 3, 4, 5]
[(1, 2), (1, 4), (2, 4), (2, 3), (3, 4), (3, 5)]
```

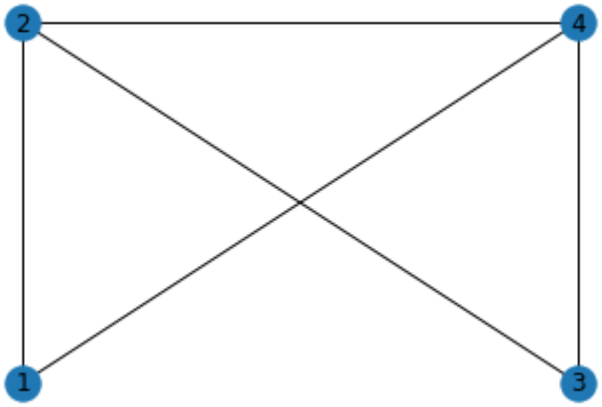
In [ ]:
```
#Eliminamos nodo
G.remove_node(5)
print(G)
print(G.nodes)

print(G.edges)
```

```
Graph with 4 nodes and 5 edges
[1, 2, 3, 4]
[(1, 2), (1, 4), (2, 4), (2, 3), (3, 4)]
```

In [ ]:
```
#Adyacencia
print(G[1])
print(G.adj[1])
```

```
{2: {}, 4: {}}
{2: {}, 4: {}}
```

In [ ]:
```
plot=plt.plot()
# nx.draw(G)
#nx.draw(G,with_labels=True)
nx.draw(G,with_labels=True,pos={1:(0,0),2:(0,1),3:(1,0),4:(1,1)})
plt.show()
```



Incluimos etiquetas y datos

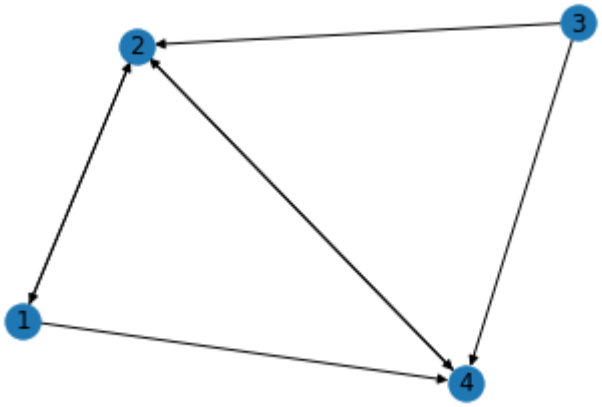
In [ ]:
```
G2=nx.Graph()
G2.add_nodes_from(["a","b",10,11,3.14])
G2.add_edge("a",10,object="Una cadena", weight=1.1)
G2.add_edges_from([("a",3.14,{"object":"Otra cadena","weight":1.2}),("b",11,{"obje

print(G2['a'])
#Contenido de la arista entre "a" y 10
print(G2.edges["a",10])
print(G2["a"][10])
print(G2.nodes)
print(G2.edges)

#Representamos
plot=plt.plot()
nx.draw(G2,with_labels=True)
plt.show()
```

```
{10: {'object': 'Una cadena', 'weight': 1.1}, 3.14: {'object': 'Otra cadena', 'wei
ght': 1.2}}
{'object': 'Una cadena', 'weight': 1.1}
{'object': 'Una cadena', 'weight': 1.1}
['a', 'b', 10, 11, 3.14]
[('a', 10), ('a', 3.14), ('b', 11)]
```

Grafos dirigidos

In [ ]:
```
Gd=nx.DiGraph()
Gd.add_nodes_from([1,2,3,4])
Gd.add_edges_from([(1,2),(2,1),(1,4),(3,4),(2,4),(4,2),(3,2)])
plot=plt.plot()
```

```
nx.draw(Gd,with_labels=True)
plt.show()
```



Funciones de análisis de grafos

Grado

In [ ]:
```
print(G.degree())
print(G2.degree())
print(Gd.in_degree())
print(Gd.out_degree())
#Grado de un vértice concreto
print(Gd.in_degree(1),Gd.out_degree(1))
#También se puede hacer con Gd.in_degree[i]
```

```
[(1, 2), (2, 3), (3, 2), (4, 3)]
[('a', 2), ('b', 1), (10, 1), (11, 1), (3.14, 1)]
[(1, 1), (2, 3), (3, 0), (4, 3)]
[(1, 2), (2, 2), (3, 2), (4, 1)]
1 2
```

Componentes conexas

In [ ]:
```
componentes=nx.connected_components(G2)
print(list(componentes))

componentes_conexas=nx.weakly_connected_components(Gd)
print(list(componentes_conexas))

componentes_fuertemente_conexas=nx.strongly_connected_components(Gd)
print(list(componentes_fuertemente_conexas))
```

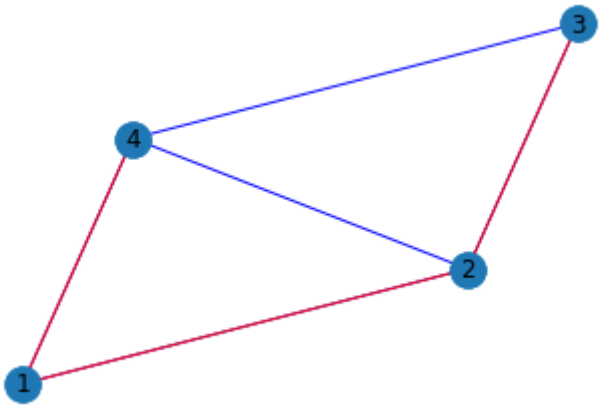
```
[{10, 3.14, 'a'}, {11, 'b'}]
[{1, 2, 3, 4}]
[{1, 2, 4}, {3}]
```

Árbol abarcador mínimo

In [ ]:
```
aam=nx.minimum_spanning_tree(G)
#se puede especificar algorithm=prim o algorithm=kruskal
#También hay otra forma de recuperar el AAM: minimum_spanning_edges
print(aam.nodes)
print(aam.edges)

#Representamos
plot=plt.plot()
```

```
pos=nx.spring_layout(G)
nx.draw(G,pos=pos,with_labels=True,edge_color='b')
nx.draw(aam,pos=pos,with_labels=False,edge_color='r')
plt.show()
#
```

```
[1, 2, 3, 4]
[(1, 2), (1, 4), (2, 3)]
```



Camino mínimo

In [ ]:

```
camino=nx.shortest_path(G,1,3)
print(camino)

plot=plt.plot()
nx.draw(G,pos=pos,with_labels=True,edge_color='b')
nx.draw_networkx_edges(G,pos=pos,arrows=True,edgelist=[(1,4),(4,3)],edge_color='r')
nx.draw_networkx_edge_labels(G,pos=pos,edge_labels={(1,4):"1",(4,3):"2"})
plt.show()
```

```
[1, 4, 3]
```