NetworkX y nx.Graph()

En NetworkX, la clase `nx.Graph()` representa un grafo no dirigido. Es decir, no hace distinción entre la dirección de las aristas: si existe una arista entre u y v, se considera tanto de u a v como de v a u.

A continuación, veremos sus principales características, cómo podemos explorarlo y también cómo modificarlo.

1. ¿Qué es un nx.Graph()?

Un objeto de tipo `nx.Graph` es la estructura base para trabajar con grafos no dirigidos en la librería NetworkX. Internamente, el grafo:

- Guarda un conjunto de nodos.
- Guarda un conjunto de aristas que conectan esos nodos.
- Permite asociar atributos tanto a los nodos como a las aristas y al grafo en general.

Ejemplo de creación:

```
import networkx as nx

G = nx.Graph()
```

En este momento, G es un grafo vacío que no contiene nodos ni aristas.

2. ¿Qué contiene el grafo y cómo verlo?

2.1. Nodos

PROF

Los nodos en NetworkX pueden ser de cualquier tipo (enteros, cadenas, objetos, etc.) mientras sean hashables (que puedan usarse como claves en un diccionario de Python).

- Agregar un nodo: `G.add_node("A")`
- Agregar varios nodos: `G.add_nodes_from(["B", "C"])`
- Ver la lista de nodos: `G.nodes()`
    - Retorna un objeto especial, pero se puede convertir a lista usando `list(G.nodes())`
- Ver datos asociados al nodo: `G.nodes(data=True)`
    - Retorna no solo los nodos, sino también los atributos que cada nodo pudiera tener

2.2. Aristas

Las aristas en `nx.Graph` no tienen dirección. Conectan dos nodos sin importar el orden.

- Agregar una arista: `G.add_edge("A", "B")`
- Agregar varias aristas:

```
```

```
G.add_edges_from([
    ("A", "C"),
    ("B", "C")
])
```

- Ver la lista de aristas: `G.edges()`
- Ver datos de las aristas: `G.edges(data=True)`

2.3. Atributos

- Atributos de nodos: se pueden especificar al momento de agregarlos, por ejemplo `G.add_node("A", weight=5)`
- Atributos de aristas: de la misma forma, al añadir una arista, por ejemplo `G.add_edge("A", "B", distance=3)`
- Atributos del grafo: `G.graph` es simplemente un diccionario donde podemos guardar información general, por ejemplo:

```
G.graph["nombre"] = "MiGrafo"
G.graph["descripcion"] = "Este es un grafo para pruebas"
```

3. ¿Cómo explorar y usar el grafo?

Dado un grafo ya definido (por ejemplo, uno que ya tiene nodos y aristas añadidos), podemos explorarlo de varias maneras:

```
# Listar nodos y aristas
print("Nodos:", G.nodes())
print("Aristas:", G.edges())

# Obtener el grado de cada nodo
print("Grado de los nodos:", G.degree())
# o por nodo específico
print("Grado de A:", G.degree("A"))

# Ver los vecinos (adyacentes) de un nodo
list(G.neighbors("A"))
```

PROF

3.1. Algoritmos de grafos con NetworkX

Una de las ventajas de NetworkX es la cantidad de funciones y algoritmos que ofrece. Algunos ejemplos:

- Búsqueda en amplitud (BFS) o en profundidad (DFS)
- Cálculo de caminos más cortos: `nx.shortest_path(G, source, target)`

- Cálculo de centralidades, clústeres, componentes conectados, etc.

Por ejemplo, para hallar el camino más corto entre dos nodos:

```
import networkx as nx

path = nx.shortest_path(G, source="A", target="C")
print("Camino más corto de A a C:", path)
```

4. ¿Cómo modificar un grafo ya existente?

4.1. Agregar nodos/aristas

```
G.add_node("D")
G.add_edge("A", "D", atributo="nuevo")
```

4.2. Eliminar nodos/aristas

- Eliminar un nodo: `G.remove_node("D")`
- Eliminar varios nodos: `G.remove_nodes_from(["B", "C"])`
- Eliminar una arista: `G.remove_edge("A", "B")`
- Eliminar varias aristas:

```
G.remove_edges_from([
    ("A", "C"),
    ("B", "C")
])
```

PROF

4.3. Actualizar atributos

Si ya existe un nodo o arista, podemos añadirle o cambiarle atributos:

```
# Para una arista
G["A"]["B"]["weight"] = 10

# Para un nodo
G.nodes["A"]["color"] = "blue"
```

5. ¿Qué ejercicios podemos hacer?

- Contar nodos y aristas:

```
num_nodos = G.number_of_nodes()
num_aristas = G.number_of_edges()
```

- Visualizar la estructura:

```
import matplotlib.pyplot as plt
nx.draw(G, with_labels=True)
plt.show()
```

- Aplicar algoritmos:
    - Encuentra el camino más corto entre dos nodos
    - Calcula la centralidad de un nodo (ej. PageRank o Betweenness Centrality)
    - Encuentra componentes conectados (`nx.connected_components(G)`)

Conclusiones

- `nx.Graph()` es la estructura fundamental para crear y manejar grafos no dirigidos en NetworkX
- Nos permite almacenar nodos y aristas con atributos opcionales
- Podemos explorar de múltiples formas sus nodos y aristas (`G.nodes()`, `G.edges()`, `G.degree()`, etc.)
- Podemos modificar el grafo dinámicamente añadiendo o quitando nodos/aristas y editando sus atributos
- NetworkX ofrece un ecosistema de algoritmos de análisis de grafos (búsqueda, caminos, centralidad, etc.)