

Teoría de grafos en python

Para estos ejercicios, necesitarás instalar la librería networkx y opcionalmente matplotlib para la visualización:

Comando a utilizar: **pip install networkx matplotlib**

Ejercicio 1: Creación y Propiedades Básicas de un Grafo

Objetivo: Crear un grafo no dirigido y calcular algunas de sus propiedades fundamentales.

1. **Crea un grafo .**
 2. **Añade los nodos** 'A', 'B', 'C', 'D', 'E'.
 3. **Añade las aristas** para formar el siguiente conjunto de conexiones:
 - ('A', 'B')
 - ('A', 'C')
 - ('B', 'D')
 - ('C', 'E')
 - ('D', 'E')
 - ('E', 'A')
 4. **Calcula e imprime:**
 - El número total de nodos.
 - El número total de aristas.
 - El **grado** del nodo 'A' (número de aristas conectadas a 'A').
 - Una lista de los **vecinos** del nodo 'E'.
 5. **Visualiza** el grafo.
-

Ejercicio 2: Búsqueda y Caminos en un Grafo Dirigido

Objetivo: Trabajar con un grafo dirigido y aplicar algoritmos de búsqueda básicos (BFS o DFS) o de camino más corto.

Imagina un sistema de rutas de una sola dirección entre ciudades:

1. **Crea un grafo dirigido** (usa `nx.DiGraph()`).
 2. **Añade aristas dirigidas** que representen las siguientes rutas:
 - Madrid Barcelona
 - Madrid Sevilla
 - Barcelona Valencia
 - Sevilla Málaga
 - Valencia Madrid
 - Málaga Barcelona
 - Málaga Granada
 3. **Encuentra e imprime el camino más corto** desde **Madrid** hasta **Granada**.
 4. **Verifica** si existe un camino desde **Granada** hasta **Sevilla** e imprime el resultado.
-

Ejercicio 3: Grafos Ponderados y el Algoritmo de Dijkstra

Objetivo: Utilizar grafos con pesos en las aristas y aplicar el algoritmo del camino más corto (Dijkstra) en un contexto práctico.

Considera un mapa de carreteras con distancias (pesos) entre ciudades:

1. **Crea un grafo ponderado** (un `nx.Graph()` simple funciona).
 2. **Añade las siguientes aristas ponderadas** (Nodo1, Nodo2, Peso/Distancia):
 - ('O', 'A', 5)
 - ('O', 'B', 1)
 - ('A', 'C', 2)
 - ('B', 'A', 2)
 - ('B', 'D', 3)
 - ('C', 'D', 1)
 - ('D', 'E', 4)
 3. **Aplica el algoritmo de Dijkstra** para encontrar el **camino más corto** y la **distancia total mínima** desde el nodo de **Origen ('O')** hasta el nodo de **Destino ('E')**.
 4. **Imprime** el camino encontrado y su costo.
-