📘 **Tema: Exploración y Análisis del Grafo Urbano**
📅 **Semana 5 — Proyecto Integrador Avance 3**

Alumno: Christian Gael Ortiz Ramirez
Docente: Eligardo Cruz Sanchez

Introducción

En esta quinta semana del proyecto integrador se trabajó con dos de los algoritmos fundamentales para la exploración de grafos: BFS (Breadth-First Search) y DFS (Depth-First Search), también conocidos como búsqueda en amplitud y búsqueda en profundidad, respectivamente.
Estos algoritmos constituyen la base de múltiples técnicas utilizadas en áreas como la inteligencia artificial, las redes de comunicación, el análisis de rutas, la minería de datos, los videojuegos y la planificación de recorridos óptimos.
El propósito principal de esta práctica fue comprender el funcionamiento interno de ambos algoritmos, identificar sus diferencias operativas, y aplicar cada uno a la exploración de un grafo representativo ya sea de tipo urbano, académico o de red de conexiones. El algoritmo BFS se basa en una estrategia de búsqueda por niveles, explorando primero todos los nodos vecinos del punto inicial antes de avanzar a niveles más profundos. Gracias a esta característica, BFS garantiza la obtención del camino más corto en grafos no ponderados, lo cual resulta especialmente útil para determinar rutas mínimas o distancias entre puntos.
Por otro lado, el algoritmo DFS emplea un enfoque recursivo o basado en pilas (LIFO), avanzando a lo largo de un camino hasta que ya no es posible continuar, para luego retroceder y explorar las demás ramas del grafo. Este método resulta más adecuado para detectar componentes conectados, ciclos y estructuras jerárquicas.

Desarrollo

El proyecto "**Exploración y Análisis del Grafo Urbano**" se compone de varios archivos en Python que trabajan de manera modular para representar, analizar y visualizar un grafo.

- **graph_traversal.py**

Contiene la clase GraphTraversal, que implementa los algoritmos **BFS**, **DFS**, el **camino más corto** y la **detección de componentes conectadas**.
Es el **núcleo del proyecto**, ya que los demás módulos dependen de sus funciones.

- **pathfinder.py**

Usa GraphTraversal para simular **rutas de transporte**.
Encuentra la mejor ruta, genera alternativas y calcula el **tiempo estimado de viaje**.
Representa la **aplicación práctica** de los algoritmos.

- **visualize_bfs.py**

Genera la **visualización del grafo** con matplotlib y networkx, mostrando cómo BFS recorre los nodos por niveles.
Sirve para representar gráficamente los resultados.

- **test_graph.py**

Incluye **8 pruebas automáticas** que verifican el correcto funcionamiento de los algoritmos y módulos usando pytest.
Garantiza la **fiabilidad y consistencia** del código.

- **main.py**

Es el **archivo principal**.
Integra todo: crea el grafo, ejecuta BFS y DFS, busca caminos, detecta componentes, simula rutas y genera la visualización final.

RESULTADOS:
Ejecución de Main el programa muestra correctamente los resultados de los algoritmos implementados.
El sistema crea un grafo con los nodos **A, B, C, D, E y F**, aplicando los recorridos **BFS** y **DFS**, la búsqueda del **camino más corto**, la **detección de componentes**

conectadas y la **simulación de rutas de transporte**.

```
=== Proyecto Semana 5: BFS y DFS ===

1. BFS desde A:
Orden BFS: ['A', 'B', 'C', 'D', 'E', 'F']

2. DFS desde A:
Orden DFS: ['A', 'B', 'D', 'E', 'F', 'C']

3. Camino más corto entre A y F:
Camino más corto A→F: ['A', 'C', 'F']

4. Componentes conectadas:
[['A', 'B', 'D', 'E', 'F', 'C']]

5. Planificación de rutas de transporte:
Ruta óptima: ['A', 'C', 'F']
Alternativas: []
Tiempo estimado (min): 20

Generando visualización BFS...
```

Al ejecutar las pruebas con **pytest test_graph.py**

```
collected 8 items

test_graph.py::test_bfs_order PASSED                                     [ 12%]
test_graph.py::test_dfs_order PASSED                                     [ 25%]
test_graph.py::test_shortest_path PASSED                                 [ 37%]
test_graph.py::test_connected_components PASSED                          [ 50%]
test_graph.py::test_pathfinder_route PASSED                              [ 62%]
test_graph.py::test_add_edge_bidirectional PASSED                        [ 75%]
test_graph.py::test_bfs_empty_graph PASSED                               [ 87%]
test_graph.py::test_no_path_found PASSED                                 [100%]

============================================ 8 passed in 0.01s ============================================
christianortiz@192 Semana5 %
```

Conclusiones

Durante el desarrollo de este proyecto comprendí cómo las estructuras de datos tipo grafo permiten representar y analizar conexiones entre elementos, como rutas, redes o relaciones.

Aprendí a implementar y aplicar los algoritmos BFS (Breadth-First Search) y DFS (Depth-First Search), entendiendo sus diferencias principales:

- BFS recorre el grafo por niveles y siempre encuentra el camino más corto en grafos no ponderados.

- DFS explora en profundidad y resulta útil para detectar componentes, ciclos o estructuras jerárquicas.

Además, logré integrar todos los módulos en un sistema funcional, conectando la teoría con la práctica mediante el análisis, la simulación y la visualización de un grafo urbano.

El uso de pruebas automáticas con pytest me permitió validar el correcto funcionamiento del código y asegurar su calidad.

En general, este proyecto me ayudó a fortalecer mi comprensión sobre recorridos en grafos, modularidad de código y validación de algoritmos, aplicando conceptos clave de la materia *Estructura de Datos* en un contexto real y profesional.

Prompt #1 – Análisis Comparativo Personalizado

Escenario	Problema	Algoritmo (BFS o DFS)	Justificación de elección	Desventaja del otro algoritmo
1. Rutas GPS	Encontrar la ruta más	BFS	BFS garantiza encontrar el	DFS podría perderse en caminos largos o sin salida antes de hallar la ruta óptima.
2. Redes Sociale	Encontrar usuarios a u	BFS	Ideal porque explora prime	DFS se profundiza en una sola rama, lo que no refleja niveles de conexión reales.
3. Laberintos o v	Buscar una salida o te	DFS	DFS permite explorar rutas	BFS usa más memoria y puede ser lento si el laberinto es muy grande.
4. Compiladores	Recorrer estructuras d	DFS	Permite recorrer árboles de	BFS podría consumir más recursos sin aportar ventaja en este tipo de jerarquías.
5. Detección de	Identificar grupos de n	BFS	Recorre todos los nodos al	DFS puede hacerlo, pero con mayor riesgo de stack overflow si hay demasiados nodos o ciclos.

Reflexión Post-IA

- ¿Los escenarios generados por la IA coinciden con mi intuición?
  Sí. Coinciden con mi comprensión de los algoritmos: BFS es ideal para caminos cortos y búsqueda por niveles, mientras que DFS es mejor para exploraciones profundas o jerárquicas.

- ¿Hay casos donde ambos algoritmos son igualmente válidos?
  Sí. En problemas de detección de componentes o grafos pequeños, tanto BFS como DFS pueden usarse con resultados equivalentes, variando solo el orden del recorrido.

- ¿Qué criterio adicional considero importante?
  Además de la complejidad temporal, considero esencial analizar la memoria utilizada y la estructura del grafo (si es denso, disperso o tiene ciclos). Estos factores determinan si BFS o DFS resulta más eficiente en la práctica.