Actividad Práctica Aplicada de Profundización

Implementación en Python de Algoritmos de Recorrido en Grafos: BFS y DFS

Elaborado por:
Edward Iván Becerra Niño

Estructura de Datos II

Docente:
Flavio Alexander Navarro Carmona

Fundación Universitaria Compensar
Bogotá, 27 de marzo de 2025

Resumen

El presente documento describe el desarrollo e implementación en Python de una estructura de datos tipo grafo mediante una lista de adyacencia, e integra dos algoritmos fundamentales de recorridos: la búsqueda en anchura (Breadth-First Search, BFS) y la búsqueda en profundidad (Depth-First Search, DFS). Se explica el funcionamiento del código, las mejores prácticas empleadas, y se analizan aspectos conceptuales y de implementación que sustentan el desarrollo de la solución. Además, se incluyen consideraciones de diseño de interfaz y validación de entrada para asegurar la robustez del programa. Se explican las decisiones técnicas, la estructura del código y su alineación con principios de programación robusta (Joyanes, 2007; Cormen et al., 2009).

Introducción

Los grafos son estructuras de datos dinámicas y no lineales ampliamente utilizadas para modelar relaciones entre entidades en diversos ámbitos, tales como redes de comunicación, planificación de tareas y análisis de rutas (Joyanes, 2007; Cormen et al., 2009). Los algoritmos de recorrido en grafos permiten explorar la totalidad o parte de los nodos de un grafo de manera sistemática. Entre los algoritmos más conocidos se encuentran la búsqueda en anchura (BFS) y la búsqueda en profundidad (DFS), los cuales son la base para la resolución de problemas complejos en áreas como la inteligencia artificial y la optimización de rutas (Sedgewick & Wayne, 2011).

El código presentado se estructura en torno a una clase Graph que utiliza una lista de adyacencia para representar el grafo. Se incorporan funciones específicas para agregar nodos y aristas, así como métodos para realizar los recorridos BFS y DFS. Adicionalmente, se implementa una interfaz de línea de comandos que permite al usuario seleccionar el algoritmo deseado y el nodo de inicio, mostrando el resultado de la exploración del grafo.

Desarrollo del Código

1. Clase Graph

Representa grafos mediante una lista de adyacencia (adj_list), donde cada nodo (genérico tipo T) tiene una lista de vecinos. Métodos clave:

- add_node y add_edge: Añaden nodos y aristas, con validación de bucles y opción bidireccional.
- bfs y dfs: Implementan los recorridos usando una cola (BFS) y recursión (DFS), respectivamente.
- Validación y visualización: Incluye métodos para verificar nodos (validate_node) y generar representaciones visuales (visualize_graph).

2. Algoritmos de Recorrido
   - BFS: Utiliza una cola (deque) para explorar nodos nivel por nivel, garantizando un orden óptimo para rutas más cortas (Cormen et al., 2009).
   - DFS: Emplea recursión para explorar ramas profundamente antes de retroceder, útil para detectar componentes conexos (Sedgewick & Wayne, 2011).
3. Interfaz de Usuario

   Usa la biblioteca colorama para una experiencia interactiva con menús coloridos. Los usuarios seleccionan algoritmos, nodos iniciales (A-G) y visualizan la estructura del grafo.

Metodología

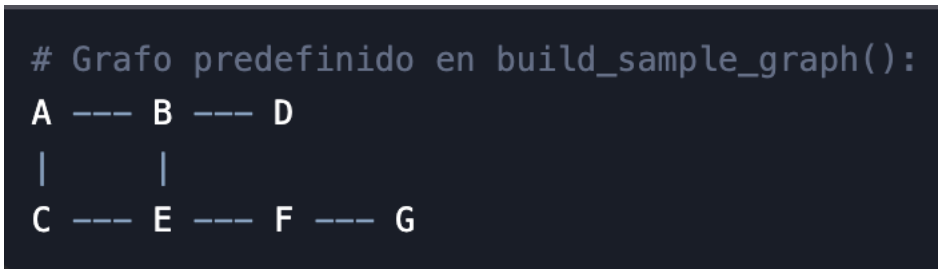
- Lenguaje y herramientas: Python 3.11, tipado estático (TypeVar), y bibliotecas como collections.deque para gestión eficiente de colas.
- Paradigma: Orientación a objetos, separando lógica de grafos, algoritmos e interfaz.
- Validación: El método has_path verifica conectividad usando BFS, mientras que edge_count calcula aristas para grafos no dirigidos.

Resultados y Discusión

La implementación descrita cumple con el objetivo de demostrar el uso de estructuras de datos dinámicas no lineales en Python mediante la manipulación de grafos. La utilización de BFS y DFS permite al usuario explorar el grafo de distintas maneras, lo cual es fundamental para resolver problemas de conectividad y búsqueda de caminos. La inclusión de una interfaz interactiva mejora la experiencia del usuario y posibilita una validación robusta de los datos de entrada.

Este enfoque se alinea con los principios teóricos descritos en la literatura, en donde los grafos y sus recorridos son esenciales para la resolución de problemas en ciencia de la computación (Cormen et al., 2009; Sedgewick & Wayne, 2011). Asimismo, el uso de Python y la aplicación de tipado y documentación detallada favorecen el mantenimiento y la escalabilidad del código, aspectos críticos en proyectos de desarrollo de software a nivel universitario y profesional.

Ejemplo de ejecución:



- Salida de BFS desde "A": A -> B -> C -> D -> E -> F -> G.
- Salida de DFS desde "A": A -> B -> D -> E -> F -> G -> C.

Limitaciones:

- La recursión en DFS puede causar desbordamiento de pila en grafos grandes.
- No soporta grafos ponderados, limitando aplicaciones en rutas con pesos (Heileman, 1998).

Eficiencia:

Ambos algoritmos tienen complejidad temporal O(V + E), óptima para grafos representados con listas de adyacencia (Joyanes, 2007).

Conclusión

El desarrollo del programa en Python demuestra la eficacia de utilizar estructuras de datos dinámicas como los grafos para modelar y resolver problemas complejos. La implementación de los algoritmos BFS y DFS se fundamenta en principios teóricos sólidos y se complementa con una interfaz de usuario que facilita la interacción y validación de los datos. Este trabajo no solo refuerza la comprensión de los algoritmos de recorrido en

grafos, sino que también destaca la importancia de aplicar buenas prácticas de programación y documentación en el desarrollo de software.

El código demuestra una implementación sólida de grafos y algoritmos de recorrido, siguiendo buenas prácticas de programación. Su modularidad permite extensiones futuras, como añadir pesos a las aristas o implementar algoritmos avanzados (Dijkstra, Prim). Este trabajo se alinea con aplicaciones reales en redes, inteligencia artificial y optimización.

Referencias

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms (3ª ed.). MIT Press.
- Joyanes, L. (2007). Estructura de datos en C++. McGraw-Hill.
- Joyanes Aguilar, L. (2006). Programación en C++: algoritmos, estructuras de datos y objetos (2ª ed.). McGraw-Hill España.
- Sedgewick, R., & Wayne, K. (2011). Algorithms (4ª ed.). Addison-Wesley Professional.
- Python Software Foundation. (2020). Python documentation. https://docs.python.org/3/

- Heileman, G. L. (1998). Estructuras de Datos, Algoritmos y Programación Orientada a Objetos. McGraw-Hill.