Laboratorio Grafos

Estructura de datos

Juan Sebastián Rodríguez Pabon
Julian Pérez Gomes
David Rodríguez Jurado

Dirigido a John Corredor

6 de Noviembre de 2024

Bogotá D.C

Introducción

En el ámbito de la informática los grafos son estructuras fundamentales que permiten modelar una amplia variedad de problemas en diversas disciplinas, desde la optimización de rutas en redes de transporte hasta la representación de relaciones en redes sociales. Un grafo se compone de un conjunto de nodos (o vértices) y un conjunto de aristas (o bordes) que conectan estos nodos, lo que permite representar relaciones y conexiones de manera efectiva.

Este documento presenta la documentación detallada del código desarrollado para un laboratorio enfocado en la implementación de grafos en el lenguaje de programación C++. El objetivo principal de este programa es construir un grafo dirigido y ponderado, y calcular el camino más corto desde un nodo de origen a otros nodos utilizando el algoritmo de Dijkstra, uno de los algoritmos más conocidos y utilizados para resolver problemas de caminos más cortos en grafos.

El algoritmo de Dijkstra, propuesto por el matemático Edsger W. Dijkstra en 1956, es un método eficiente que permite encontrar el camino más corto desde un nodo inicial a todos los demás nodos en un grafo con pesos no negativos. Su aplicación es fundamental en áreas como la navegación GPS, la planificación de rutas y la optimización de redes, lo que resalta la relevancia de comprender y aplicar este algoritmo en la práctica.

A lo largo de este documento, se describirán las estructuras de datos utilizadas para representar el grafo, así como la implementación del algoritmo de Dijkstra. Se incluirán ejemplos de entrada y salida, así como una explicación detallada de cada parte del código, lo que facilitará la comprensión del funcionamiento del programa. Además, se discutirán las posibles extensiones y mejoras que se pueden realizar en el futuro, así como las limitaciones del algoritmo en ciertos contextos.

Este laboratorio no solo tiene como objetivo la implementación técnica del algoritmo, sino también proporcionar una comprensión más profunda de los conceptos teóricos subyacentes en la teoría de grafos y su aplicación en problemas del mundo real. Al finalizar, se espera que el lector tenga una visión clara de cómo los grafos y los algoritmos asociados pueden ser utilizados para resolver problemas complejos de manera eficiente.

Estructura del Código

El código desarrollado para este laboratorio está diseñado para representar un grafo no dirigido con pesos en sus aristas, lo que permite modelar situaciones en las que las conexiones entre nodos tienen un costo o distancia asociada. La estructura del código se organiza de manera modular, facilitando la comprensión y el mantenimiento del mismo. A continuación, se describen los componentes clave de la implementación:

Definición de la Estructura de Datos Edge: La estructura Edge es fundamental para representar las aristas del grafo. Cada instancia de esta estructura contiene información sobre los nodos que conecta, así como el peso asociado a la conexión. Esto permite que el grafo almacene no solo la relación entre los nodos, sino también el costo de viajar de uno a otro.

Funciones para Agregar Aristas: Se implementan funciones específicas que permiten agregar aristas al grafo de manera eficiente. Estas funciones se encargan de actualizar la estructura de datos del grafo, asegurando que las conexiones se establezcan correctamente y que los pesos se registren de forma adecuada. La modularidad de estas funciones facilita la adición de nuevas aristas y la modificación de las existentes.

Implementación del Algoritmo de Dijkstra: El núcleo del código se centra en la implementación del algoritmo de Dijkstra, que se utiliza para calcular el camino más corto desde un nodo de origen a todos los demás nodos del grafo. Este algoritmo se basa en una estrategia de búsqueda que explora los nodos de manera sistemática, actualizando las distancias mínimas a medida que avanza. La implementación incluye el uso de una cola de prioridad para seleccionar el siguiente nodo a explorar, lo que optimiza el rendimiento del algoritmo.

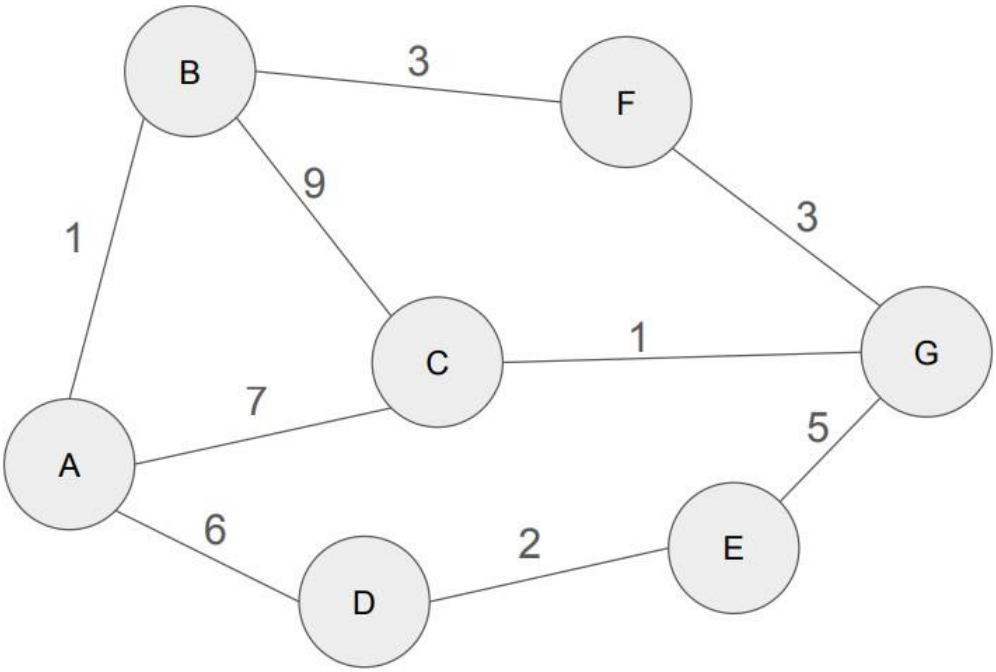
A través de esta estructura, el código no solo cumple con su objetivo funcional, sino que también se presenta de manera clara y organizada, lo que facilita su comprensión y posible extensión en el futuro.

Representación del Grafo

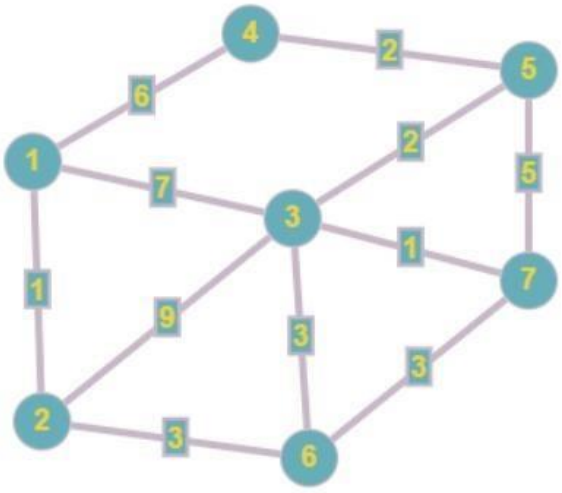
La representación visual del grafo es crucial para entender la estructura y las relaciones entre los nodos. A continuación, se describe el grafo utilizado como ejemplo en este laboratorio:

- **Nodos**: El grafo está compuesto por un conjunto de nodos, que en este caso son representados por las letras A, B, C, D, E, F y G. Cada nodo actúa como un punto de conexión, donde se pueden iniciar o finalizar rutas.
- **Aristas y Pesos**: Las aristas que conectan los nodos tienen un peso asociado, que representa la distancia o el costo de viajar entre ellos. Estos pesos son fundamentales para el cálculo de la distancia mínima, ya que el algoritmo de Dijkstra utiliza esta información para determinar el camino más corto. Por ejemplo, si la arista entre A y B tiene un peso de 5, esto indica que el costo de viajar de A a B es 5 unidades.
- **Grafo No Dirigido**: Es importante destacar que este grafo es no dirigido, lo que significa que las conexiones entre los nodos funcionan en ambas direcciones. Por lo tanto, si hay una arista entre A y B, se puede viajar de A a B y de B a A con el mismo costo.

El objetivo principal de este laboratorio es aplicar el algoritmo de Dijkstra para calcular el camino más corto desde un nodo de origen, como el nodo A, a todos los demás nodos del grafo. Este proceso implica considerar los pesos de las aristas y determinar la ruta más eficiente en términos de costo total. A través de este ejercicio, se busca no solo implementar el algoritmo, sino también comprender su funcionamiento y su aplicabilidad en problemas del mundo real.



2 -> 3 -> 5 -> 7 -> 6 -> 3 - >5 -> 4 -> 1 -> 3 -> 2 -> 1



Estructura Edge

La estructura Edge es un componente esencial en la representación de un grafo, ya que se utiliza para modelar las aristas que conectan los nodos. Cada arista no solo establece una conexión entre dos nodos, sino que también incluye un peso asociado que representa la "distancia" o "costo" de moverse entre esos nodos. Esta representación es crucial para el funcionamiento del algoritmo de Dijkstra, que se basa en la evaluación de estos pesos para determinar el camino más corto.
En el contexto de este programa, la estructura Edge se define con los siguientes atributos:

- **dest:** Este atributo indica el nodo de destino de la arista. Es un identificador que permite al programa saber a qué nodo se conecta la arista desde el nodo de origen. Este enfoque es fundamental para construir la lista de adyacencia del grafo, donde cada nodo puede acceder fácilmente a sus vecinos.

- **peso:** Este atributo almacena el peso de la arista, que puede interpretarse como la distancia o el costo de la conexión entre los nodos. El peso es un valor numérico que se utiliza en los cálculos del algoritmo de Dijkstra para determinar la ruta más eficiente.

Almacenar el peso en la estructura Edge permite que el grafo sea flexible y adaptable a diferentes contextos, ya que los pesos pueden representar distancias físicas, costos monetarios, tiempos de viaje, entre otros.

La implementación de la estructura Edge permite que el grafo sea modelado como una lista de adyacencia. En este modelo, cada nodo tiene una lista de aristas que representan sus conexiones a otros nodos. Esta representación es eficiente en términos de espacio y permite realizar operaciones de búsqueda y actualización de manera rápida, lo que es fundamental para el rendimiento del algoritmo de Dijkstra.

Función addEdge

La función addEdge es una parte fundamental del código, ya que permite agregar aristas entre nodos en el grafo de manera dinámica. Dado que este grafo es no dirigido, cada arista conecta dos nodos en ambas direcciones. Esto significa que, al añadir una arista entre el nodo u y el nodo v, ambos nodos se convierten en vecinos entre sí, y el peso asociado a la arista representa la distancia o costo de la conexión.

Los parámetros de la función son los siguientes:

- **graph**: Este parámetro es una referencia al grafo, que se representa como una lista de adyacencia. Cada posición en esta lista contiene un conjunto de aristas que representan las conexiones de un nodo. Al pasar el grafo como referencia, se asegura que cualquier modificación realizada en la función afecte directamente al grafo original.
- **u**: Este parámetro representa el nodo de origen de la arista. Es el nodo desde el cual se establece la conexión hacia el nodo de destino.
- **v**: Este parámetro representa el nodo de destino de la arista. Es el nodo al que se conecta la arista desde el nodo de origen.
- **peso**: Este parámetro indica el peso o distancia de la arista entre los nodos u y v. Este valor es crucial para el cálculo de caminos más cortos, ya que el algoritmo de Dijkstra utilizará

este peso para evaluar las rutas posibles.

La función addEdge es fundamental para construir el grafo de manera dinámica, permitiendo que los nodos se conecten mediante aristas ponderadas. Al utilizar esta función, se puede modificar la estructura del grafo en tiempo de ejecución, lo que proporciona flexibilidad y adaptabilidad al programa. Esta capacidad de construir el grafo de forma dinámica es especialmente útil en aplicaciones donde la topología de la red puede cambiar o donde se requiere la adición de nuevos nodos y conexiones.

Algoritmo de Dijkstra

La función dijkstra implementa el algoritmo de Dijkstra, un método eficiente y ampliamente utilizado para calcular el camino más corto desde un nodo de origen a todos los demás nodos en un grafo ponderado. Este algoritmo es especialmente adecuado para grafos que no contienen aristas con pesos negativos, lo que garantiza que la solución encontrada sea óptima para grafos con pesos no negativos y busca encontrar la distancia mínima desde el nodo inicial a todos los otros nodos posibles.

Pasos generales del algoritmo:

- Inicializa la distancia de todos los nodos como infinita, excepto el nodo de origen que tiene una distancia de cero.
- Utiliza una cola de prioridad para seleccionar el nodo con la menor distancia acumulada que aún no ha sido procesado.
- Para cada nodo, verifica sus vecinos y actualiza la distancia acumulada si se encuentra un camino más corto.
- Repite el proceso hasta que se hayan calculado las distancias mínimas para todos los nodos.

Parámetros de la función:

- **graph:** Grafo representado como una lista de adyacencia, donde cada nodo tiene una lista de aristas que lo conectan a otros nodos.
- **src:** Nodo de origen desde el cual se calcularán las distancias mínimas.

La función devuelve un vector de distancias mínimas desde el nodo de origen a cada uno de los otros nodos. Esta información puede ser útil para aplicaciones donde se necesita conocer la ruta más económica o rápida entre diferentes puntos en un sistema.

Ejemplos de Uso

Para utilizar el código, se debe construir el grafo añadiendo las aristas correspondientes entre los nodos.
Luego, se llama a la función `dijkstra` para obtener las distancias mínimas desde un nodo de origen.
Ejemplo:

Graph graph(nodos);
// Añadir aristas addEdge(graph, 0, 1, 1);
addEdge(graph, 1, 2, 3);
// Calcular camino más corto desde el nodo 0 vector<int> distancias = dijkstra(graph, 0);

Este ejemplo crea un pequeño grafo y calcula las distancias mínimas desde el nodo 0 hacia otros nodos.

Conclusiones

Este código proporciona una implementación clara y eficiente de un grafo ponderado no dirigido junto con el algoritmo de Dijkstra
para encontrar caminos mínimos. Es útil para aplicaciones en sistemas de navegación, redes de comunicación, logística, y otros
campos que requieren encontrar rutas óptimas en un entorno de nodos conectados con pesos específicos.

Bibliografía

- Khan Academy. (n.d.). Graphs. Recuperado de https://www.khanacademy.org/computing/computer-science/algorithms/graphs
- Weiss, M. A. (2013). *Data structures and algorithm analysis in C++* (4th ed.). Pearson.