Laboratorio Grafos

Pontificia Universidad Javeriana

Estructura de datos

Juan Sebastian Rodriguez

Julian Perez Gomez

Profesor: John Corredor

6 de noviembre 2024

Introducción

Este documento contiene la documentación detallada del código correspondiente a un laboratorio sobre grafos, implementado en C++. El objetivo del programa es construir un grafo y calcular el camino más corto desde un nodo de origen a otros nodos utilizando el algoritmo de Dijkstra.

Estructura del Código

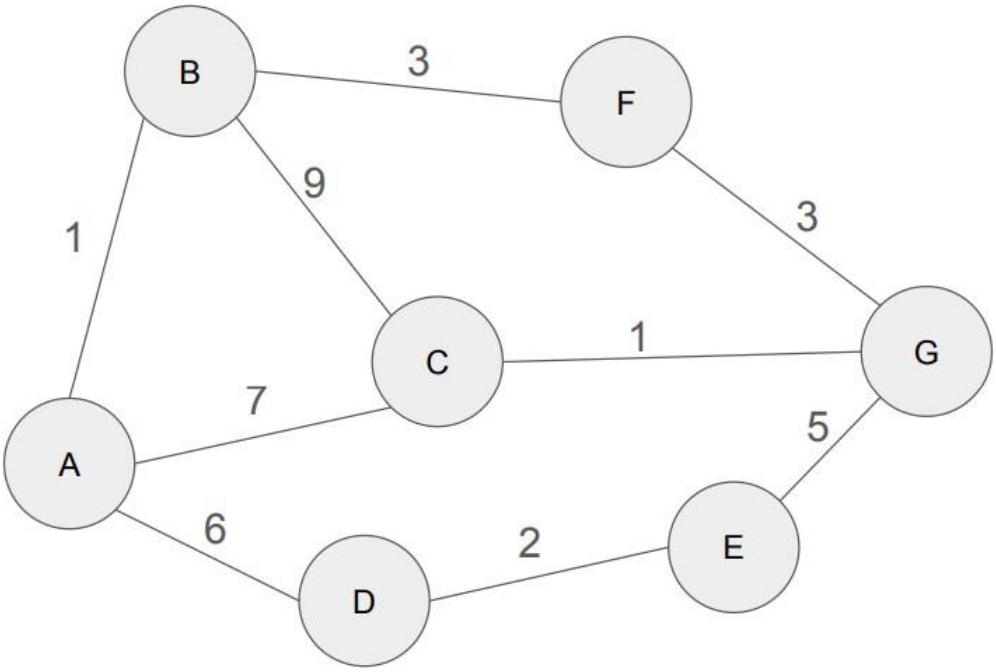
El código está diseñado para representar un grafo no dirigido con pesos en sus aristas y contiene la implementación de un algoritmo de búsqueda de caminos mínimos. La estructura del código incluye una definición de la estructura de datos `Edge`, funciones para
agregar aristas y la implementación del algoritmo de Dijkstra.

Representación del Grafo

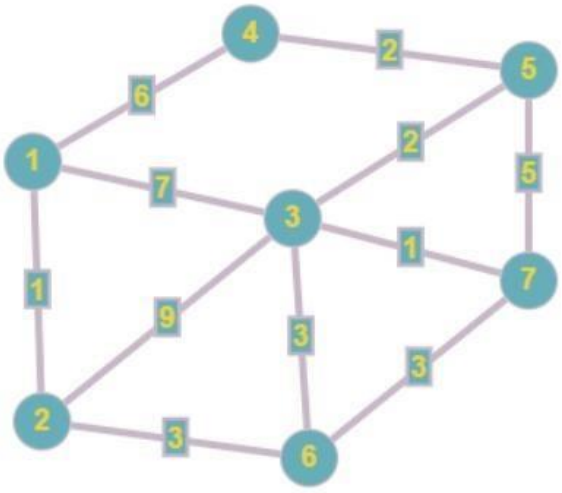
La imagen a continuación muestra el grafo que se utiliza como ejemplo en este laboratorio. Cada nodo representa un punto de conexión, y las aristas entre los nodos tienen un peso asociado que indica la distancia o costo de viajar entre ellos.

- Los nodos (A, B, C, D, E, F, G) representan los puntos de conexión.
- Los números junto a cada arista indican el peso de la conexión entre los nodos, lo que se utiliza en el cálculo de la distancia mínima.
- Este grafo es no dirigido, lo cual significa que la conexión entre los nodos funciona en ambas direcciones.

El objetivo del laboratorio es utilizar el algoritmo de Dijkstra para calcular el camino más corto desde un nodo de origen (por ejemplo, el nodo A) a los demás nodos del grafo, tomando en cuenta los pesos de las aristas.



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



Estructura Edge

La estructura `Edge` se utiliza para representar una arista en el grafo. Cada arista conecta dos nodos y tiene un peso asociado que representa "distancia" o "costo" de moverse entre esos nodos. En el contexto de este programa, `Edge` facilita la representación
de una arista entre dos nodos mediante los siguientes atributos:

- `dest`: Indica el nodo de destino de la arista.
- `peso`: Almacena el peso de la arista, el cual puede interpretarse como la distancia o costo de la conexión entre los nodos.

Esta estructura permite que el grafo sea modelado como una lista de adyacencia, donde cada nodo tiene una lista de aristas que representan sus conexiones a otros nodos.

Función addEdge

La función `addEdge` permite agregar una arista entre dos nodos en el grafo. Este grafo es no dirigido, lo cual significa que cada arista conecta dos nodos en ambas direcciones. Cuando se añade una arista entre el nodo `u` y el nodo `v`, ambos se convierten en vecinos entre sí, con el peso `peso` que representa la distancia entre ellos.

Parámetros de la función:
- `graph`: Referencia al grafo, el cual es una lista de adyacencia. Cada posición en esta lista contiene un conjunto de aristas
  que representan las conexiones de un nodo.
- `u`: Nodo de origen de la arista.
- `v`: Nodo de destino de la arista.
- `peso`: Peso o distancia de la arista entre los nodos `u` y `v`.

Esta función es fundamental para construir el grafo dinámicamente, permitiendo que los nodos se conecten mediante aristas ponderadas.

Algoritmo de Dijkstra

La función `dijkstra` implementa el algoritmo de Dijkstra, que permite calcular el camino más corto desde un nodo de origen a todos los demás nodos en el grafo. Este algoritmo es adecuado

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:
1. Inicializa la distancia de todos los nodos como infinita, excepto el nodo de origen que tiene una distancia de cero.
2. Utiliza una cola de prioridad para seleccionar el nodo con la menor distancia acumulada que aún no ha sido procesado.
3. Para cada nodo, verifica sus vecinos y actualiza la distancia acumulada si se encuentra un camino más corto.
4. 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:

```cpp
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.