**Operaciones básicas de Grafos**

**1. Introducción**

Los grafos son estructuras fundamentales en el ámbito de la teoría de grafos y son ampliamente utilizados en diversas áreas, desde la informática hasta la logística, pasando por la biología y la ingeniería. Un grafo es una representación abstracta de un conjunto de objetos interconectados, donde estos objetos se denominan nodos o vértices, y las conexiones entre ellos se llaman aristas.

El estudio de operaciones con grafos es esencial para comprender y resolver una amplia gama de problemas, desde encontrar la ruta más corta entre dos puntos en una red de transporte hasta modelar relaciones sociales en redes sociales. Este manual tiene como objetivo proporcionar una guía completa para entender y utilizar operaciones básicas y avanzadas en grafos.

**2. Definición de las operaciones**

Un grafo consta de un conjunto de objetos o elementos junto a sus relaciones. Es preciso definir las operaciones básicas para construir la estructura y, en general, modificar sus elementos. En definitiva, especificar el tipo abstracto de datos grafo.

Ahora se definen operaciones básicas, a partir de las cuales se construye el grafo. Su realización depende de la representación elegida (matiz de adyacencia, o listas de adyacencia).

Las operaciones con grafos son manipulaciones o transformaciones que se realizan sobre estructuras de datos de grafos. Un grafo es una estructura que consta de nodos (o vértices) y aristas (o bordes) que conectan estos nodos.

Las operaciones más básicas con grafos son la inserción de Nodos y Aristas, eliminación de nodos y aristas, recorridos, búsqueda e impresión, a continuación, se describen las operaciones básicas:

**1.** **Agregar un Nodo (Vértice):** Esta operación implica agregar un nuevo nodo al grafo. Un nodo representa una entidad individual en el grafo, como una ubicación en un mapa o una persona en una red social. Al agregar un nuevo nodo, se debe actualizar la estructura de datos del grafo para incluir el nodo agregado.

**2. Eliminar un Nodo (Vértice):** La eliminación de un nodo implica eliminar un nodo existente del grafo. Esto también implica eliminar todas las conexiones asociadas con ese nodo, es decir, eliminar todas las aristas que salen o entran en el nodo.

**3. Agregar una Arista:** Una arista es una conexión entre dos nodos en el grafo. Agregar una arista implica establecer una conexión entre dos nodos existentes en el grafo. Esta operación puede ser dirigida o no dirigida, dependiendo de si la conexión tiene una dirección específica o no.

**4. Eliminar una Arista:** La eliminación de una arista implica eliminar una conexión entre dos nodos existentes en el grafo. Al eliminar una arista, se rompe la conexión entre los nodos y ya no se consideran adyacentes entre sí.

Estas son algunas de las operaciones básicas que se realizan comúnmente en grafos. Cada una de estas operaciones tiene aplicaciones prácticas en una variedad de campos, incluidas las redes sociales, la logística, la optimización de rutas, la planificación de proyectos y mucho más.

**3. Codificación de las operaciones básicas**

En esta sección, exploraremos los algoritmos y la codificación de estas operaciones básicas en grafos. Desde la implementación de estructuras de datos adecuadas hasta la escritura de algoritmos eficientes, nos sumergiremos en el mundo práctico de trabajar con grafos a través de ejemplos detallados y código explicativo.

***GrafoNoDirigido.h***

#ifndef GRAFONODIRIGIDO\_H

#define GRAFONODIRIGIDO\_H

#include <vector>

#include <stack>

**class** **GrafoNoDirigido** {

**private:**

**int** numNodos;

std::vector<std::vector<**int**>> adyacencia;

**public:**

GrafoNoDirigido(**int** n);

**void** **agregarNodo**();

**void** **eliminarNodo**(**int** nodo);

**void** **agregarArista**(**int** nodoA, **int** nodoB);

**void** **eliminarArista**(**int** nodoA, **int** nodoB);

**void** **imprimirGrafo**();

**void** **busquedaDFS**(**int** nodoInicial);

};

#endif // GRAFONODIRIGIDO\_H

***GrafoNoDirigido.cpp***

#include "GrafoNoDirigido.h"

#include <iostream>

#include <algorithm> // Necesario para std::find

GrafoNoDirigido::GrafoNoDirigido(**int** n) : numNodos(n), adyacencia(n, std::vector<**int**>()) {}

**void** GrafoNoDirigido::agregarNodo() {

++numNodos;

adyacencia.resize(numNodos);

}

**void** GrafoNoDirigido::eliminarNodo(**int** nodo) {

nodo--;

**if** (nodo >= **0** && nodo < numNodos) {

// Creamos un nuevo vector de adyacencia sin las conexiones asociadas al nodo que queremos eliminar

std::vector<std::vector<**int**>> nuevoAdyacencia(numNodos - **1**, std::vector<**int**>());

// Copiamos las conexiones del vector de adyacencia original al nuevo vector, excluyendo las conexiones asociadas al nodo que queremos eliminar

**int** nuevoIndice = **0**;

**for** (**int** i = **0**; i < numNodos; ++i) {

**if** (i != nodo) {

**for** (**int** vecino : adyacencia[i]) {

**if** (vecino != nodo + **1**) {

nuevoAdyacencia[nuevoIndice].push\_back(vecino > nodo + **1** ? vecino - **1** : vecino);

}

}

nuevoIndice++;

}

}

// Actualizamos el vector de adyacencia con el nuevo vector creado

adyacencia = nuevoAdyacencia;

numNodos--;

std::cout << "**\n**Nodo " << nodo + **1** << " eliminado correctamente.**\n**" << std::endl;

} **else** {

std::cerr << "**\n**Error: El nodo a eliminar está fuera del rango del grafo.**\n**" << std::endl;

}

}

**void** GrafoNoDirigido::agregarArista(**int** nodoA, **int** nodoB) {

nodoA--;

nodoB--;

**if** (nodoA >= **0** && nodoA < numNodos && nodoB >= **0** && nodoB < numNodos) {

// Verificamos si los nodos existen en el grafo

**if** (adyacencia.size() >= nodoA && adyacencia.size() >= nodoB) {

adyacencia[nodoA].push\_back(nodoB);

adyacencia[nodoB].push\_back(nodoA);

std::cout << "**\n**Arista entre " << nodoA + **1** << " y " << nodoB + **1** << " agregada correctamente.**\n**" << std::endl;

} **else** {

std::cerr << "**\n**Error: Al menos uno de los nodos ingresados no existe en el grafo.**\n**" << std::endl;

}

} **else** {

std::cerr << "**\n**Error: Al menos uno de los nodos ingresados esta fuera del rango del grafo. **\n**" << std::endl;

}

}

**void** GrafoNoDirigido::eliminarArista(**int** nodoA, **int** nodoB) {

nodoA--;

nodoB--;

**if** (nodoA >= **0** && nodoA < numNodos && nodoB >= **0** && nodoB < numNodos) {

// Verificamos si los nodos existen en el grafo

**if** (adyacencia.size() >= nodoA && adyacencia.size() >= nodoB) {

// Verificamos si hay una arista entre los nodos ingresados

**auto** itA = std::find(adyacencia[nodoA].begin(), adyacencia[nodoA].end(), nodoB);

**auto** itB = std::find(adyacencia[nodoB].begin(), adyacencia[nodoB].end(), nodoA);

**if** (itA != adyacencia[nodoA].end() && itB != adyacencia[nodoB].end()) {

adyacencia[nodoA].erase(itA);

adyacencia[nodoB].erase(itB);

std::cout << "**\n**Arista entre " << nodoA + **1** << " y " << nodoB + **1** << " eliminada correctamente.**\n**" << std::endl;

} **else** {

std::cerr << "**\n**Error: No hay una arista entre los nodos ingresados.**\n**" << std::endl;

}

} **else** {

std::cerr << "**\n**Error: Al menos uno de los nodos ingresados no existe en el grafo.**\n**" << std::endl;

}

} **else** {

std::cerr << "**\n**Error: Al menos uno de los nodos ingresados esta fuera del rango del grafo.**\n**" << std::endl;

}

}

**void** GrafoNoDirigido::imprimirGrafo() {

std::cout << "Lista de adyacencia del grafo:" << std::endl;

**for** (**int** i = **0**; i < numNodos; ++i) {

std::cout << "Nodo " << i + **1** << " esta conectado con:";

**for** (**int** vecino : adyacencia[i]) {

std::cout << " " << vecino + **1**;

}

std::cout << std::endl;

}

std::cout << std::endl;

}

**void** GrafoNoDirigido::busquedaDFS(**int** nodoInicial) {

std::vector<**bool**> visitado(numNodos, false);

std::stack<**int**> pila;

pila.push(nodoInicial);

std::cout << "Recorrido DFS desde el nodo " << nodoInicial << ": ";

**while** (!pila.empty()) {

**int** nodoActual = pila.top();

pila.pop();

**if** (!visitado[nodoActual]) {

std::cout << nodoActual << " ";

visitado[nodoActual] = true;

**for** (**int** vecino : adyacencia[nodoActual]) {

**if** (!visitado[vecino]) {

pila.push(vecino);

}

}

}

}

std::cout << std::endl;

}

**6. Conclusiones**

Las operaciones básicas con grafos, como agregar y eliminar nodos y aristas, son fundamentales para manipular la estructura y la conectividad del grafo, afectando directamente su capacidad para representar relaciones entre entidades. Estas operaciones son la base sobre la cual se construyen algoritmos más complejos y aplicaciones prácticas en una variedad de campos, desde la optimización de rutas hasta el análisis de redes sociales. La eficiencia en la implementación de estas operaciones es crucial para garantizar un rendimiento óptimo en grafos de gran escala.

**7. Referencias**

[1] Estructura de datos en C++, Luis Joyanes e Ignacio Zahonero.

[2] Data Structures and Algorithms in Java, Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser.