Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Imagen que contiene Patrón de fondo

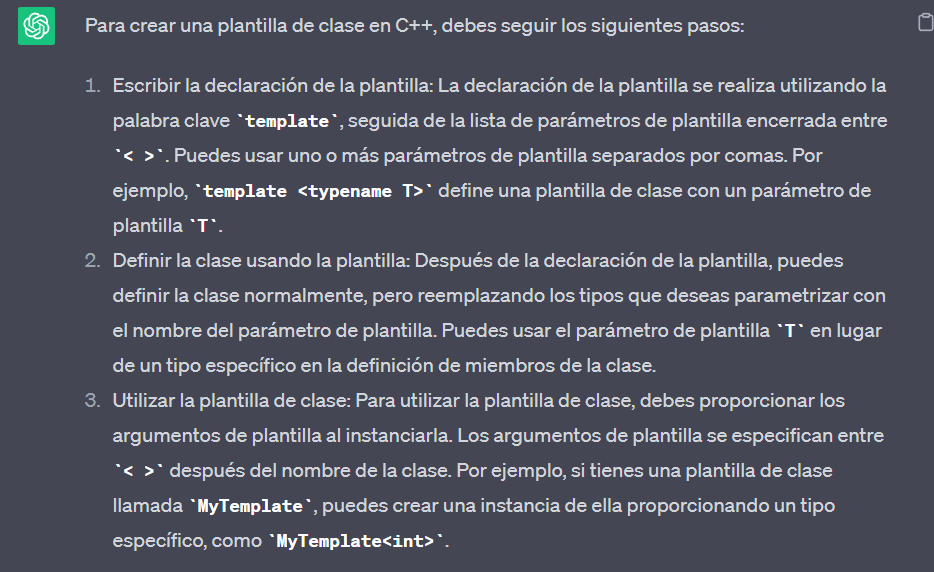
Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



Texto

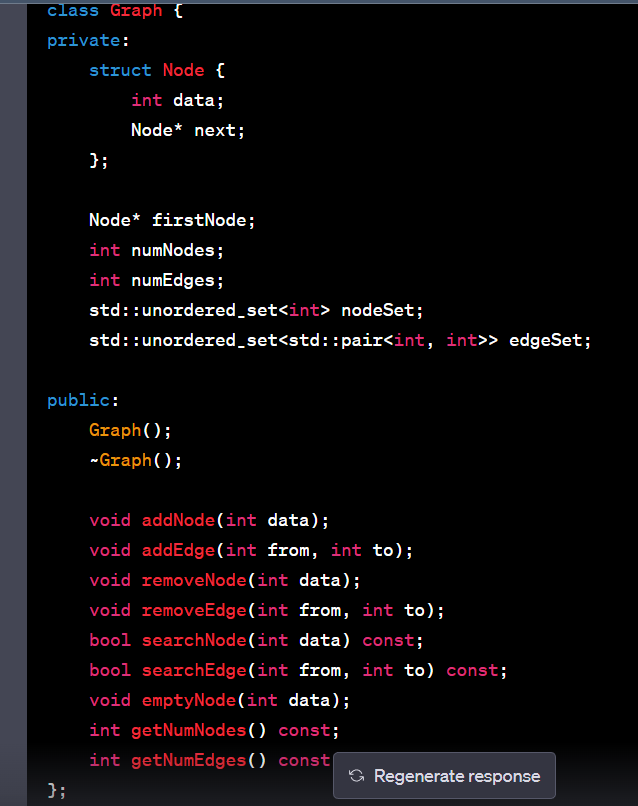
Descripción generada automáticamente

Texto

Descripción generada automáticamente

Implementación grafica:

.h:



Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Código:

#ifndef GRAPH\_H

#define GRAPH\_H

#include <unordered\_set>

class Graph {

private:

struct Node {

int data;

Node\* next;

};

Node\* firstNode;

int numNodes;

int numEdges;

std::unordered\_set<int> nodeSet;

std::unordered\_set<std::pair<int, int>> edgeSet;

public:

Graph();

~Graph();

void addNode(int data);

void addEdge(int from, int to);

void removeNode(int data);

void removeEdge(int from, int to);

bool searchNode(int data) const;

bool searchEdge(int from, int to) const;

void emptyNode(int data);

int getNumNodes() const;

int getNumEdges() const;

};

#endif // GRAPH\_H

Cpp:

#include "graph.h"

#include <iostream>

Graph::Graph() : firstNode(nullptr), numNodes(0), numEdges(0) {}

Graph::~Graph() {

Node\* currentNode = firstNode;

while (currentNode != nullptr) {

Node\* nextNode = currentNode->next;

delete currentNode;

currentNode = nextNode;

}

}

void Graph::addNode(int data) {

if (nodeSet.count(data) == 0) {

Node\* newNode = new Node;

newNode->data = data;

newNode->next = nullptr;

if (firstNode == nullptr) {

firstNode = newNode;

} else {

Node\* currentNode = firstNode;

while (currentNode->next != nullptr) {

currentNode = currentNode->next;

}

currentNode->next = newNode;

}

nodeSet.insert(data);

numNodes++;

} else {

std::cout << "Node " << data << " already exists in the graph." << std::endl;

}

}

void Graph::addEdge(int from, int to) {

if (nodeSet.count(from) != 0 && nodeSet.count(to) != 0) {

std::pair<int, int> edge = std::make\_pair(from, to);

if (edgeSet.count(edge) == 0) {

edgeSet.insert(edge);

numEdges++;

} else {

std::cout << "Edge from " << from << " to " << to << " already exists in the graph." << std::endl;

}

} else {

std::cout << "One or both of the nodes (" << from << ", " << to << ") do not exist in the graph." << std::endl;

}

}

void Graph::removeNode(int data) {

if (nodeSet.count(data) != 0) {

for (auto it = edgeSet.begin(); it != edgeSet.end(); ) {

if (it->first == data || it->second == data) {

it = edgeSet.erase(it);

numEdges--;

} else {

++it;

}

}

Node\* currentNode = firstNode;

Node\* prevNode = nullptr;

while (currentNode != nullptr) {

if (currentNode->data == data) {

if (prevNode == nullptr) {

firstNode = currentNode->next;

} else {

prevNode->next = currentNode->next;

}

delete currentNode;

numNodes--;

nodeSet.erase(data);

break;

}

prevNode = currentNode;

currentNode = currentNode->next;

}

} else {

std::cout << "Node " << data << " does not exist in the graph." << std::endl;

}

}

void Graph::removeEdge(int from, int to) {

std::pair<int, int> edge = std::make\_pair(from, to);

if (edgeSet.count(edge) != 0) {

edgeSet.erase(edge);

numEdges--;

} else {

std::cout << "Edge from " << from << " to " << to << " does not exist in the graph." << std::endl;

}

}

bool Graph::searchNode(int data) const {

return nodeSet.count(data) != 0;

}

bool Graph::searchEdge(int from, int to) const {

std::pair<int, int> edge = std::make\_pair(from, to);

return edgeSet.count(edge) != 0;

}

void Graph::emptyNode(int data) {

if (nodeSet.count(data) != 0) {

for (auto it = edgeSet.begin(); it != edgeSet.end(); ) {

if (it->first == data || it->second == data) {

it = edgeSet.erase(it);

numEdges--;

} else {

++it;

}

}

} else {

std::cout << "Node " << data << " does not exist in the graph." << std::endl;

}

}

int Graph::getNumNodes() const {

return numNodes;

}

int Graph::getNumEdges() const {

return numEdges;

}

Main:

#include <iostream>

#include "graph.h"

int main() {

Graph myGraph;

myGraph.addNode(1);

myGraph.addNode(2);

myGraph.addNode(3);

myGraph.addEdge(1, 2);

myGraph.addEdge(2, 3);

std::cout << "Number of nodes: " << myGraph.getNumNodes() << std::endl;

std::cout << "Number of edges: " << myGraph.getNumEdges() << std::endl;

myGraph.emptyNode(2);

std::cout << "Number of nodes after emptying: " << myGraph.getNumNodes() << std::endl;

std::cout << "Number of edges after emptying: " << myGraph.getNumEdges() << std::endl;

return 0;

}