#include "Red.h"

#include "Linea.h"

#include "Estacion.h"

#include <limits>

#include <iostream>

Red::Red(std::string nombre) {

// Constructor de la clase Red

this->nombre = nombre;

this->cabezaLineas = nullptr;

this->numLineas = 0;

}

Red::~Red() {

// Destructor de la clase Red

liberarMemoria(cabezaLineas);

}

Linea\* Red::getCabezaLineas() const {

// Método para obtener un puntero a la cabeza de la lista de líneas

return cabezaLineas;

}

std::string Red::getNombre() const {

// Método para obtener el nombre de la red

return nombre;

}

int Red::getNumLineas() const {

// Método para obtener el número de líneas en la red

return numLineas;

}

void Red::liberarMemoria(Linea\* linea) {

// Función privada para liberar la memoria de las líneas

if (linea == nullptr) {

return;

}

Estacion\* temp = linea->getCabezaEstaciones();

while (temp != nullptr) {

Estacion\* siguiente = temp->getSiguienteEstacion();

delete temp;

temp = siguiente;

}

liberarMemoria(linea->getSiguienteLinea());

}

void Red::agregarLinea(Linea\* linea) {

// Método para agregar una nueva línea a la red

if (cabezaLineas == nullptr) {

cabezaLineas = linea;

} else {

Linea\* temp = cabezaLineas;

while (temp->getSiguienteLinea() != nullptr) {

temp = temp->getSiguienteLinea();

}

temp->enlazarSiguienteLinea(linea);

}

numLineas++;

}

bool Red::eliminarLinea(std::string nombreLinea) {

// Método para eliminar una línea de la red

if (cabezaLineas == nullptr) {

return false;

}

Linea\* temp = cabezaLineas;

if (temp->getNombre() == nombreLinea) {

Estacion\* estacion = temp->getCabezaEstaciones();

bool tieneTransferencia = false;

while (estacion != nullptr) {

if (estacion->getEsTransferencia()) {

tieneTransferencia = true;

break;

}

estacion = estacion->getSiguienteEstacion();

}

if (tieneTransferencia) {

std::cout << "La línea no se pudo eliminar ya que tiene una estación de transferencia.\n";

return false;

} else {

if (numLineas == 1) {

cabezaLineas = nullptr;

} else {

cabezaLineas = temp->getSiguienteLinea();

}

delete temp;

numLineas--;

return true;

}

}

while (temp->getSiguienteLinea() != nullptr) {

if (temp->getSiguienteLinea()->getNombre() == nombreLinea) {

Linea\* eliminar = temp->getSiguienteLinea();

Estacion\* estacion = eliminar->getCabezaEstaciones();

bool tieneTransferencia = false;

while (estacion != nullptr) {

if (estacion->getEsTransferencia()) {

tieneTransferencia = true;

break;

}

estacion = estacion->getSiguienteEstacion();

}

if (tieneTransferencia) {

std::cout << "La línea no se pudo eliminar ya que tiene una estación de transferencia.\n";

return false;

} else {

temp->enlazarSiguienteLinea(eliminar->getSiguienteLinea());

delete eliminar;

numLineas--;

return true;

}

}

temp = temp->getSiguienteLinea();

}

std::cout << "No se encontró la línea o no se pudo eliminar.\n";

return false;

}

int Red::contarEstaciones() const {

// Método para contar el número total de estaciones en la red

int totalEstaciones = 0;

Linea\* temp = cabezaLineas;

while (temp != nullptr) {

Estacion\* estacion = temp->getCabezaEstaciones();

int estacionesUnicas = 0;

while (estacion != nullptr) {

if (!estacion->getEsTransferencia()) {

estacionesUnicas++;

} else {

bool yaContada = false;

Linea\* otraLinea = cabezaLineas;

while (otraLinea != temp) {

Estacion\* otraEstacion = otraLinea->getCabezaEstaciones();

while (otraEstacion != nullptr) {

if (otraEstacion->getNombre() == estacion->getNombre()) {

yaContada = true;

break;

}

otraEstacion = otraEstacion->getSiguienteEstacion();

}

if (yaContada) {

break;

}

otraLinea = otraLinea->getSiguienteLinea();

}

if (!yaContada) {

estacionesUnicas++;

}

}

estacion = estacion->getSiguienteEstacion();

}

totalEstaciones += estacionesUnicas;

temp = temp->getSiguienteLinea();

}

return totalEstaciones;

}

int Red::calcularTiempoEntre(Estacion\* inicio, Estacion\* destino) const {

// Método para calcular el tiempo entre dos estaciones

int tiempo = 0;

Estacion\* temp = inicio;

while (temp != destino) {

tiempo += temp->getTiempoSiguiente();

temp = temp->getSiguienteEstacion();

}

return tiempo;

}

int Red::calcularTiempoLlegada(std::string estacionInicio, std::string estacionDestino, std::string horaInicio) {

// Método para calcular el tiempo de llegada entre dos estaciones

Estacion\* inicioEstacion = nullptr;

Estacion\* destinoEstacion = nullptr;

Linea\* linea = cabezaLineas;

while (linea != nullptr) {

Estacion\* temp = linea->getCabezaEstaciones();

while (temp != nullptr) {

if (temp->getNombre() == estacionInicio) {

inicioEstacion = temp;

}

if (temp->getNombre() == estacionDestino) {

destinoEstacion = temp;

}

temp = temp->getSiguienteEstacion();

}

linea = linea->getSiguienteLinea();

}

if (inicioEstacion == nullptr || destinoEstacion == nullptr) {

return -1; // Alguna de las estaciones no existe en la red

}

linea = cabezaLineas;

bool mismaLinea = false;

while (linea != nullptr) {

Estacion\* temp = linea->getCabezaEstaciones();

while (temp != nullptr) {

if (temp == inicioEstacion) {

mismaLinea = true;

break;

}

temp = temp->getSiguienteEstacion();

}

if (mismaLinea) {

break;

}

linea = linea->getSiguienteLinea();

}

if (!mismaLinea) {

return -2; // Las estaciones no están en la misma línea

}

int tiempoLlegada = calcularTiempoEntre(inicioEstacion, destinoEstacion);

// Aquí puedes implementar la lógica para convertir el tiempo en segundos a un formato de hora

// y sumarle la hora de inicio proporcionada.

return tiempoLlegada;

}