Desafío 2 Planteamiento

Integrante 1: Daniel Cruz

Integrante 2: Miguel Cacabelos.

Taller escogido: Red De Metro

Profesores de teoría:

Aníbal José Guerra.

Augusto Enrique Salazar.

**Problema**

En nuestro sistema de metro, cada estación tiene un nombre único y un valor de tiempo que indica la duración en segundos para llegar a la siguiente o anterior estación en la misma línea.

También hay estaciones de trasferencia, que son puntos de intersección entre dos líneas, esta estación de trasferencia no podrá ser eliminada sin tiene conexiones con dos estaciones entre ellas.

Tenemos tres clases de objetos: estación, línea y red, con la siguiente jerarquía:

Una línea pertenece a una única red.

Una estación aparece solo una vez en una línea.

Una línea está presente solo una vez en una red.

Si una red tiene múltiples líneas, estas no pueden estar desconectadas.

Funcionalidades Requeridas

El programa debe incluir constructores, destructores, getters y setters, y ser capaz de:

A. Agregar una estación a una línea.

B. Eliminar una estación de una línea, con la excepción de las estaciones de trasbordo.

C. Determinar cuántas líneas tiene una red de metro.

D. Contar el número de estaciones en una línea específica.

E. Verificar si una estación dada pertenece a una línea específica.

F. Agregar una línea a la red de metro.

G. Eliminar una línea de la red de metro, solo si no tiene estaciones de trasbordo.

H. Calcular cuántas estaciones tiene una red de metro, con precaución respecto a las estaciones de trasbordo.

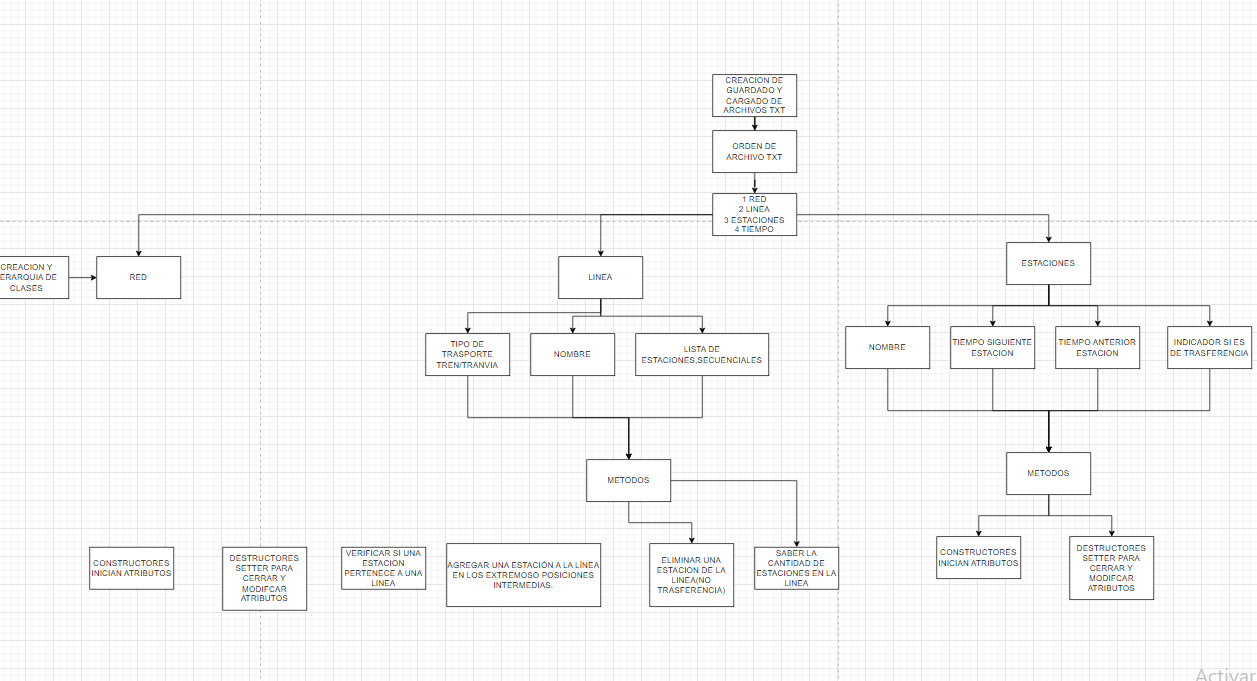
Además, se requiere un subprograma que simule el funcionamiento de la red, denominado "Cálculo del tiempo de llegada". Este programa toma la hora actual como el tiempo de salida de un tren desde una estación de la red y predice el tiempo que tardará ese tren en llegar a otra estación dentro de la misma línea. Esta simulación solo funciona para estaciones dentro de la misma línea.

**Diseño de Clases y Jerarquía:**

**ESTRUCTURA**

**Bitácora**: empezamos planeando en un diagrama de flujo la estructura de las clases

**Boceto1**



**PLANTEAMIENTO FINAL:**

Diagrama

Descripción generada automáticamente

**Clase Estación:**

Cada estación puede tener los siguientes atributos:

Nombre

Tiempo hacia la estación siguiente

Indicador de si es una estación de transferencia

Nombre de la línea a la que pertenece

Métodos:

Constructor para inicializar los atributos.

Getters para acceder a los atributos.

**Clase Nodo:**

Representa un nodo en la lista enlazada.

Atributos:

Puntero a una instancia de Estación.

Puntero al siguiente Nodo en la lista.

Métodos:

Constructor para inicializar el Nodo con una instancia de Estación.

Método para apuntar al siguiente Nodo.

**Clase Intento\_vector:**

Clase contenedora para almacenar instancias de Estación utilizando una lista enlazada.

Atributos:

Puntero al primer Nodo en la lista.

Tamaño de la lista.

Métodos:

Constructor para inicializar la lista vacía.

Destructor para liberar la memoria.

Método para agregar una instancia de Estación al final de la lista.

Método para eliminar una instancia de Estación en una posición específica.

Sobrecarga del operador [] para acceder a instancias de Estación por índice.

Método para obtener el tamaño de la lista.

**Clase Línea:**

Cada línea puede tener los siguientes atributos:

Nombre

Lista de estaciones en orden secuencial (utilizando Intento\_vector)

Puntero a la siguiente línea en la red del metro

Métodos:

Agregar una estación a la línea al final.

Eliminar una estación de la línea (excepto estaciones de transferencia).

Obtener la cantidad de estaciones en la línea.

Verificar si una estación dada pertenece a la línea.

Constructor para inicializar los atributos.

Destructor para limpiar la memoria si es necesario.

**Clase RedMetro:**

La red de metro puede tener los siguientes atributos:

Lista de líneas que forman parte de la red (utilizando Intento\_vector).

Métodos:

Agregar una línea a la red de metro.

Eliminar una línea de la red de metro (si no tiene estaciones de transferencia).

Obtener la cantidad de líneas en la red de metro.

Obtener la cantidad total de estaciones en la red de metro (considerando estaciones de transferencia).

Constructor para inicializar la red de metro.

Destructor para limpiar la memoria si es necesario. Esta función solo realiza cálculos entre estaciones de la misma línea.

**ACTUALIZACION 27/04/2024**

Se empezó a desarrollar la clase estación, teniendo en mente que no se pueden usar contenedores, esto ha generado una gran problemática al tener varios intentos fallidos de cargado de información, lo cual no esta generado un reto para poder implementar el guardado de información, ya que sin esto es complicado avanzar en la otra implementación del código.

**Problemas de implementación 02/05/2024:**

1. Se tiene un primer problema al crear la función de implementar nuevas funciones estaciones a una línea.
2. Hay un segundo problema al crear la función 7 (Eliminar una línea de la red Metro (sólo puede eliminarse si no posee estaciones de transferencia)), el problema es que no se implementó que solo se puedan eliminar líneas que no tengan transferencias

***Cambio de planteamiento 03/05/2024***

**Diagrama Uml:**

Diagrama

Descripción generada automáticamente

**Descripción UML:**

La clase Estación está compuesta por Nodo, En el código proporcionado, la relación entre Estación y Nodo es que un (Nodo) contiene una instancia de una estación (Estación). Por ser parte de una estructura de datos en la que los nodos forman parte de una lista enlazada que contiene estaciones en cada nodo. Entonces, la relación de composición implica el objeto de tipo Estación está compuesto por un objeto de tipo Nodo.

Intento\_vector contiene instancias de Estación. En el código, Intento\_vector es una clase que es una implementación de un vector o lista que contiene instancias de Estacion. La relación dice que un objeto de tipo Intento\_vector contiene múltiples instancias de Estación.

Línea tiene una instancia de Intento\_vector. En el diseño, una línea tiene una lista de estaciones (Intento\_vector) que representan las estaciones en esa línea. Entonces, la relación muestra que un objeto de tipo Linea tiene exactamente una instancia de Intento\_vector para almacenar sus estaciones.

RedMetro contiene instancias de Línea. En la estructura de la red de metro, una red (RedMetro) contiene múltiples líneas. Lo que hace que un objeto de tipo RedMetro contenga múltiples instancias de Línea.

El Menú utiliza la clase RedMetro. En el código del menú, se crea un objeto de tipo RedMetro para manejar todas las operaciones relacionadas con la red de metro. El menú utiliza este objeto para llamar a sus métodos y operar en la red de metro. Aquí, la relación es que la clase Menú depende de la funcionalidad proporcionada por la clase RedMetro.

**Desarrollo del cambio de planteamiento:**

Al principio, la idea era generar y guardar datos en un archivo de texto (txt). Sin embargo, esta aproximación se descartó más tarde al generar bastantes errores de manejo y creación de datos.

Creación de la clase Intento\_vector: Para mejorar la manipulación de la información, se introdujo la clase Intento\_vector. Esta clase proporcionaba métodos para modificar la información contenida, facilitando así la gestión de los datos.

Se decidió cambiar el enfoque hacia la implementación de clases. A medida que se iba generando el código, se creaba información que podía ser buscada y mostrada utilizando punteros. Esto implicaba crear clases como Estación, Línea, Nodo, y RedMetro para manejar la información relacionada con las estaciones, líneas, nodos y la red del metro.

Como parte de la adaptación al uso de Intento\_vector, se modificó la clase RedMetro para que funcionara como una clase que recibía punteros de Intento\_vector. Esto permitía trabajar de manera más eficiente con la información de la red del metro.

**Eficiencia En el Programa:**

En el código se utilizan punteros para administrar la memoria dinámica y evitar copias innecesarias de objetos grandes como las instancias de Estación.

Se han utilizado estructuras de datos adecuadas para almacenar y manipular la información, como listas enlazadas para representar las estaciones en una línea y un método para almacenar las líneas en la red del metro. Estas estructuras son eficientes en términos de acceso y manipulación de datos. Manejo adecuado de la memoria mediante la liberación de recursos cuando ya no son necesarios, como en el destructor de las clases.

Uso de punteros como una forma de referenciar objetos, lo que puede ser eficiente en términos de rendimiento.