INFORME PARCIAL 2

INFORMÁTICA II

Mateo Echavarria Perez

Marcos Restrepo Molina

UNIVERSIDAD DE ANTIOQUIA

2024

1. ANÁLISIS DEL PROBLEMA Y CONSIDERACIONES INICIALES

Para dar solución al problema es necesario tener en cuenta algunas consideraciones:

1.1 RED METRO

La red metro tiene características que hace que sea necesaria realizar una clase a partir de la cual podemos instanciar objetos de este tipo. Dichas características no solo se basan en facilidades para la versión actual del programa, sino que también se puede pensar como una buena alternativa para futuras versiones en las cuales se permita simular varias redes desde la misma sesión, donde cada una de estas redes tiene sus propios atributos. Dichas características son las que nos permiten guardar y mostrar los detalles de la simulación cuando sea necesario.

cada una de estas redes tendrá un miembro de tipo puntero, el cual será simplemente una dirección a la primera de las líneas a partir de la cual se irán conectando más de estas de forma unidireccional así:

1.2 LÍNEA

Cada una de las líneas pertenecientes a una red metro debe ser una clase. Ya que sus instancias deben saber su número de estaciones, nombre, siguiente línea, entre otros. En adición a esto, el atributo más importante de cada uno de estos objetos será un puntero hacia una estación del borde de la línea, dichas estaciones, como veremos en el siguiente punto, serán objetos que pueden apuntar hacia sus estaciones vecinas. Además, los métodos o acciones que se pueden implementar dentro de la clase línea serán de gran utilidad para dar una solución óptima y eficiente a acciones como inserción de estaciones o recorrido de las mismas.

Siguiendo esta lógica, las estaciones están conectadas entre sí de la siguiente manera:

1.3 ESTACIÓN

Al igual que los anteriores, debe existir una clase para instanciar objetos de tipo estación. cada uno de estos objetos están enlazados entre sí con punteros que hacen parte se sus atributos (de forma análoga a como se enlazan los elementos en las listas STL), lo que permitirá recorrer las líneas, por medio de sus estaciones, bidireccionalmente. Además facilita y optimiza enormemente la inserción de nuevas estaciones dentro de cada línea. Por otro lado, los objetos de tipo estación contendrán dentro de sus atributos variables para su nombre, tiempos, entre otros.

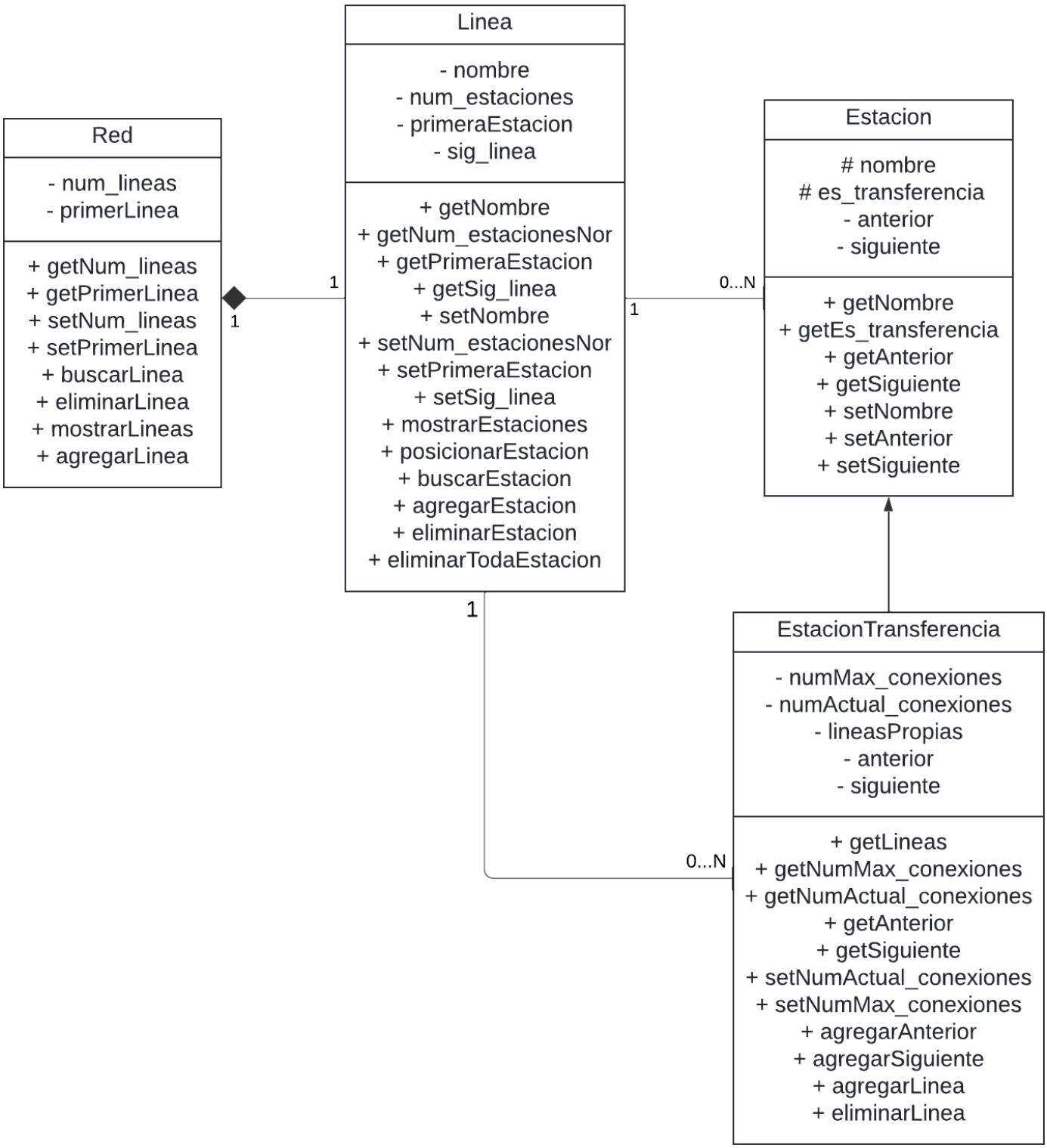
Ahora bien, las estaciones de transferencia son un tipo especial de estas estaciones, donde se puede tener una cantidad variable de líneas a las que pertenece, tiempos para cada línea, etc. Sin embargo el enlazamiento entre ellas debe ser el mismo, y comparten la mayoría de métodos. Es por ello que se toma la decisión de heredar de la clase estación una clase una para las estación de transferencia.

Por lo tanto una representación de alto nivel para este sistema de conexiones por medio de punteros entre las distintos objetos es el siguiente:

1.6 CÁLCULO DEl TIEMPO ENTRE ESTACIONES

Para calcular el tiempo entre estaciones es necesario que cada uno de los objetos de este tipo tengan un valor el cual indique tiempos para la estación siguiente y anterior. Ahora bien, para las estaciones de tipo transferencia es necesario guardar de forma separada estos datos dependiendo de la línea que se está recorriendo.

2. DIAGRAMA DE CLASES



3. DESCRIPCIÓN DEL DIAGRAMA DE CLASES Y LA LÓGICA DEL PROGRAMA

Para nuestra implementación todos los atributos de las clases serán de carácter privado, exceptuando algunos atributos dentro de la clase Estación los cuales son de carácter protegido, visto que son atributos que se comparten.

Por otro lado, los métodos que serán implementados son de carácter público, ya que se necesita poder acceder a ellos dentro de cualquier parte del programa para dar una buena implementación de acuerdo a la lógica que veremos a continuación:

3.1 CLASE RED

Como se vio anteriormente en el punto 1.1, existe un atributo dentro de la clase Red el cual direcciona a la primera instancia de tipo Línea, lo que permite crear una cantidad variable de Líneas interconectadas entre sí sin la necesidad de pedir al usuario la cantidad que va a tener la red.

Dentro del método “buscarLinea” se recibirá el nombre del objeto buscado, luego se recorren haciendo uso del puntero “primerLínea” y de los atributos existentes dentro de cada una de estos objetos llamado “sig_línea”. Una vez se encuentre el objeto con el nombre buscado, se retorna su dirección.

Para el método “eliminarLínea” se usará la función anterior para encontrar el objeto a eliminar, y una vez obtenida su dirección se procede a buscar estaciones de transferencia dentro de ella, y si no se encuentra ninguna su posición de memoria será liberada teniendo en cuenta las posibles causas de fuga de memoria.

El método “mostrarLineas” será necesario para enseñar al usuario las líneas disponibles. Estos objetos se recorren de la misma manera que en el método “buscarLinea” y se imprime uno a uno su atributo “nombre”.

Por último, dentro del método “agregarLinea” se recorre todos los punteros hasta llegar al último el cual estará apuntando a null y el cual se redirecciona al nuevo objeto creado con anterioridad.

3.2 CLASE LINEA

Como se vio en el punto 1.2, dentro de esta clase existe un atributo de tipo puntero llamado “primeraEstacion” junto con otro de tipo entero llamado “num_estaciones” el cual puede variar con el tiempo.

La forma en la que se recorre estas estaciones es muy similar a como se hace con las líneas dentro de la clase Red. Sin embargo, la peculiaridad más importante es que las estaciones pueden ser recorridas bidireccionalmente.

Los métodos “posicionarEstacion” y “buscarEstacion” funcionarán de igual manera al momento de entregarles como parámetro el nombre de la estación buscada, sin embargo, su valor de retorno cambia. El primer método retorna un entero que hace referencia a la distancia desde el objeto en la posición “primeraEstacion” hasta la estación buscada, y el segundo retorna su dirección una vez que se encuentra.

Para el método “agregarEstacion” (el cual está inspirado en la forma en que se insertan elementos en las listas STL) se le pedirá al usuario la posición de la línea en la que se quiere insertar la nueva estación: al principio, entre alguna estación en específico o al final. Luego, si se quiere insertar al principio se redirecciona “primeraEstación” hacia el objeto nuevo y se direcciona este objeto hacia la siguiente estación. Por otro lado, si se quiere insertar entre estaciones se direcciona el nuevo objeto a sus nuevas estaciones vecinas, cambiando sus atributos llamados “anterior” y “siguiente” , y luego redireccionando los punteros de sus estaciones vecinas hacia el nuevo objeto. Por último, si se quiere agregar al final, se redirecciona la estación final a la nueva y se redireccionan sus punteros “anterior” y “siguiente” donde a este último se le asigna nulo, ya que de esta forma se valida el final de la línea.

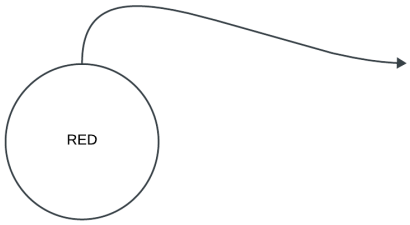
Para finalizar, dentro del método “eliminarEstacion” se recorre la línea y se destruye el objeto deseado teniendo en cuenta la liberación de la memoria dinámica. Por otro lado, este método es invocado repetidamente dentro de “eliminarTodaEstacion” donde se quiere eliminar todas las estaciones existentes dentro de la línea (necesario al momento de eliminar una línea por completo).

3.3 CLASE ESTACIÓN Y CLASE ESTACIÓN TRANSFERENCIA

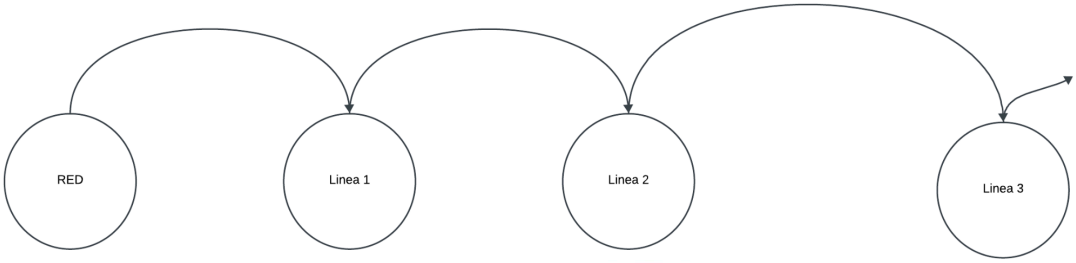
la relación de herencia entre ambas estaciones es clara, donde es necesario heredar algunos atributos de la clase base como “nombre” y “es_transferencia”

4. Esquematización de la red metro:

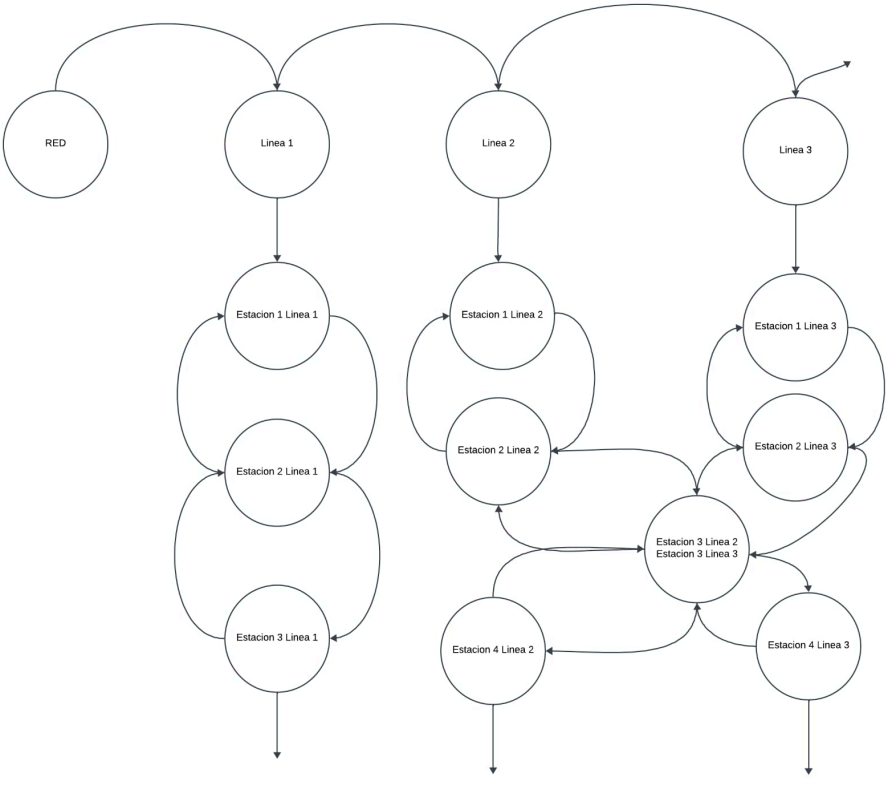
Como ya se nos ha explicado que es cada parte del sistema a continuación se nos mostrarán cómo se componen cada parte de las mismas :



Al Principio tendremos la red metro



De esta saldrian las diferentes líneas



Y de las líneas saldrán las diferentes estaciones, esto incluiría, las de transferencia

5. PROBLEMAS DE DESARROLLO

- Problemas al encontrar posibles casos de fuga de memoria.
- problemas de eficiencia al momento de eliminar objetos de tipo línea.

6. EVOLUCIÓN DE LA SOLUCIÓN

7. CONSIDERACIONES PARA TENER EN CUENTA EN LA IMPLEMENTACIÓN

La consideración más importante a tener en cuenta es el manejo de estaciones de transferencia. Ya que estas pueden tener una cantidad variable de líneas a las que pertenecen, implicando el manejo de la memoria dinámica para el almacenamiento de los distintos caminos que se puede seguir dependiendo de la línea que se esté recorriendo.

Asimismo, se debe tener en cuenta un buen manejo de la información al momento de ser liberada, ya que puede haber multiplicidad de causas para tener fuga de memoria con el modelo que estamos siguiendo. Por ejemplo, al momento de eliminar objetos de clase línea será totalmente necesario recorrer cada una de las estaciones pertenecientes a la misma y liberar toda la memoria dinámica reservada al momento de crear estas estaciones.