INFORME PARCIAL 2

INFORMÁTICA II

Mateo Echavarria Perez

Marcos Restrepo Molina

UNIVERSIDAD DE ANTIOQUIA

2024

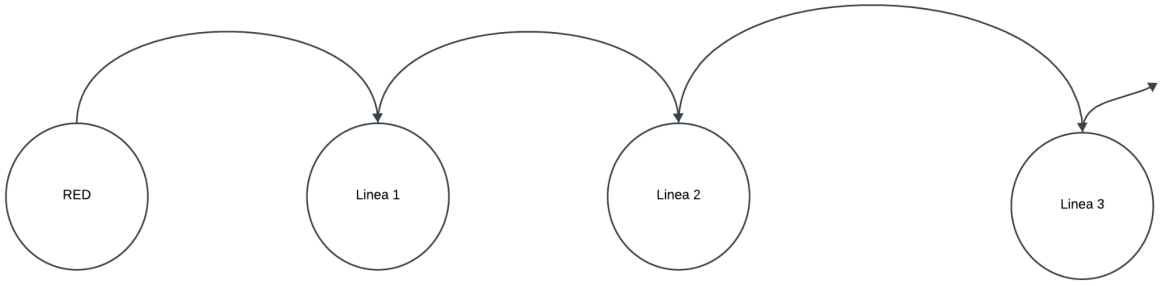
1. ANÁLISIS DEL PROBLEMA Y CONSIDERACIONES INICIALES

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.

Esta red 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:



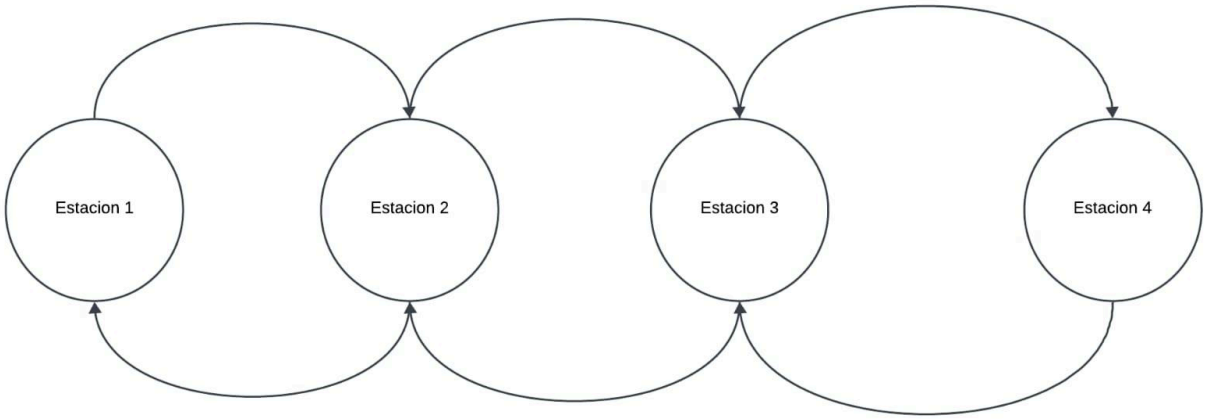
1.2 LÍNEA

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.

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 bidireccionalmente por medio de sus estaciones. 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.

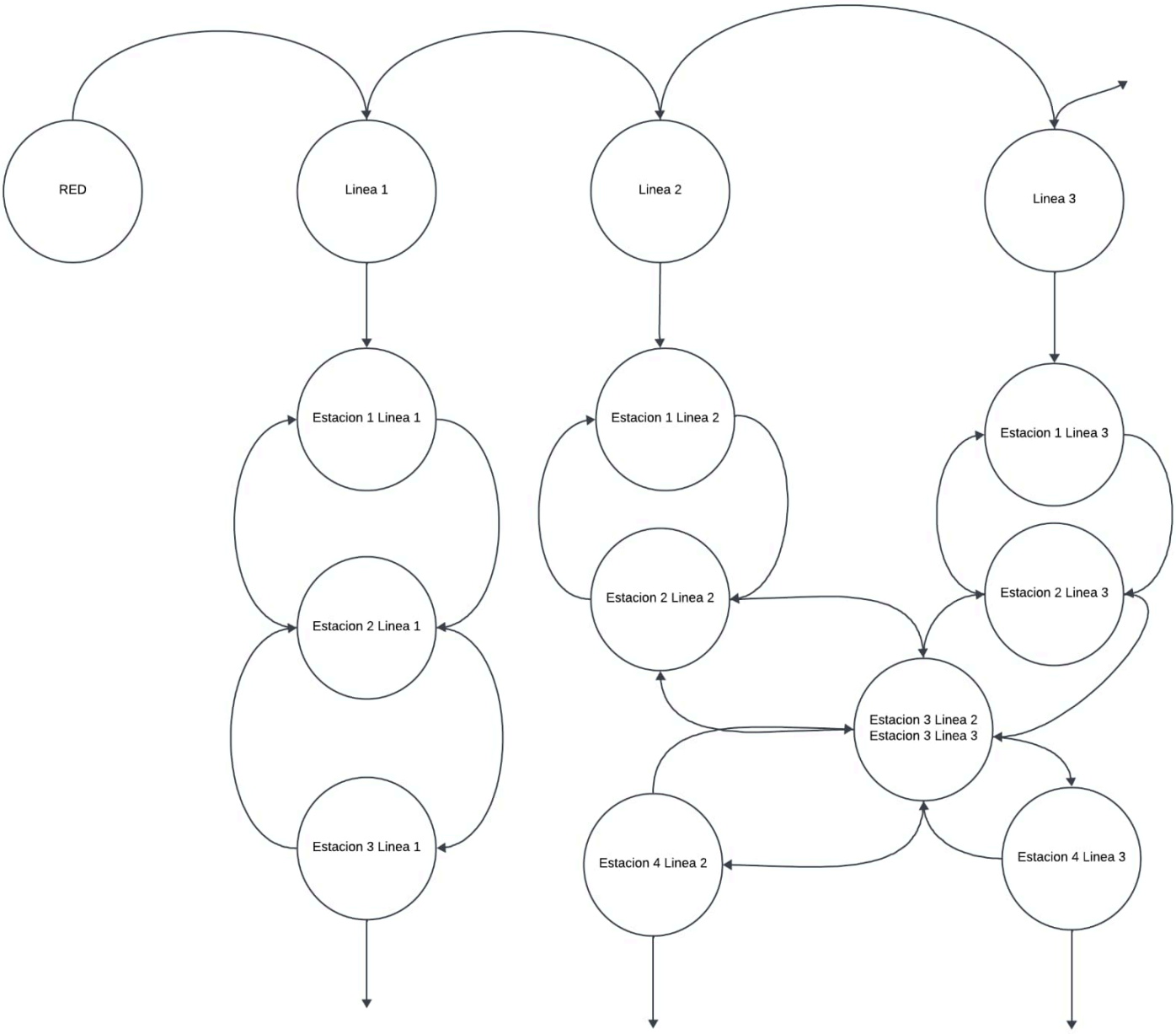
Siguiendo esta lógica, las estaciones están conectadas entre sí de la siguiente manera:



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.

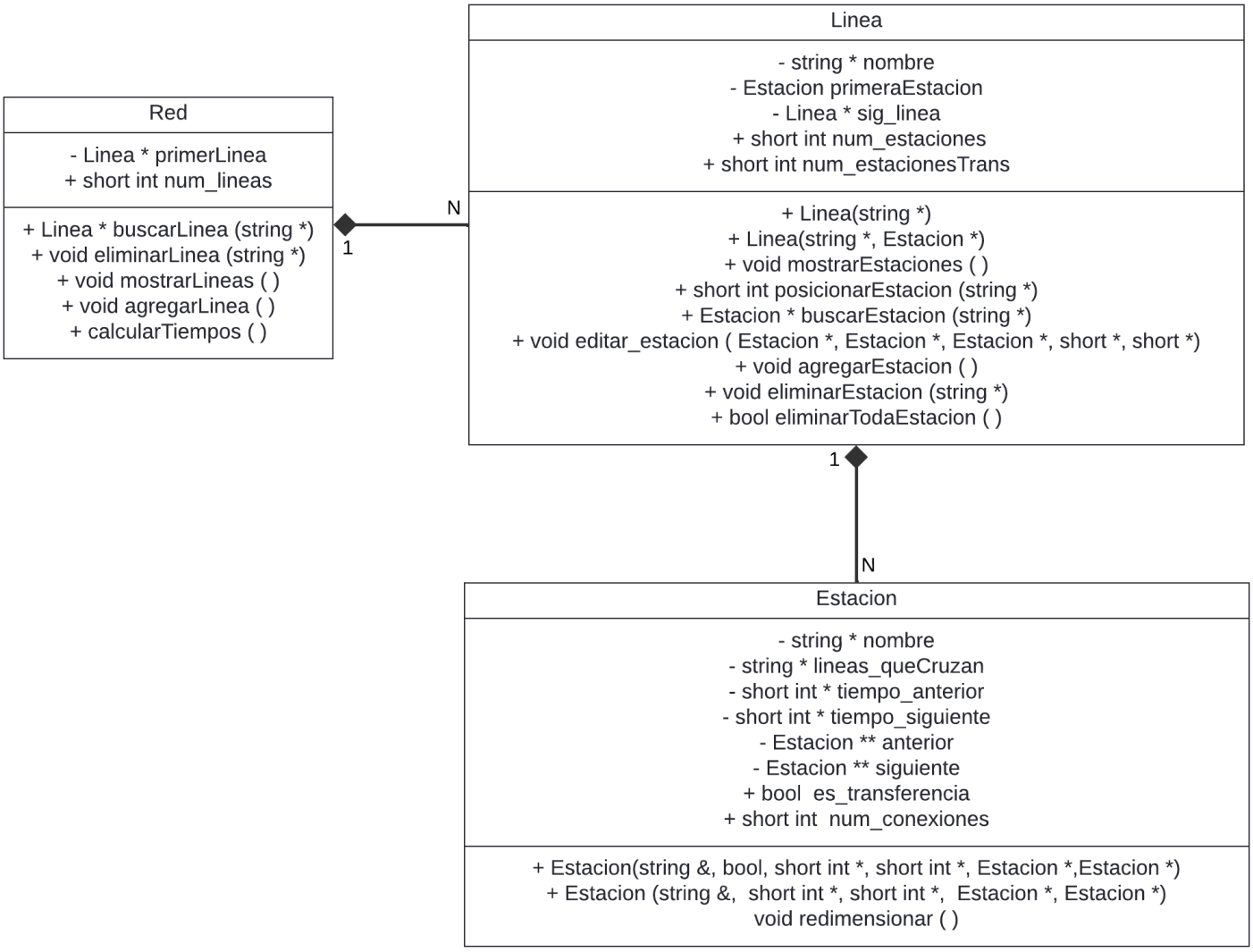
Para finalizar, al momento de calcular el tiempo entre estaciones es necesario que cada uno de los objetos de este tipo tengan un valor el cual indique tiempos para la estación siguiente y anterior. Ahora bien, para las estaciones de tipo transferencia es necesario guardar de forma separada estos datos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

En este orden de ideas, una representación de alto nivel para este sistema de conexiones por medio de punteros entre las distintos objetos es el siguiente:



NOTA: en el anterior diagrama existe una estación de transferencia que comunica las líneas 2 y 3. Sin embargo, no se visualiza ninguna para la línea 1. Esto no significa que este objeto no tiene estación de transferencia, simplemente no se ha representado en este gráfico para mejorar su visibilidad.

2. DIAGRAMA DE CLASES



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

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. Dicha cantidad de líneas se guarda en “num_ lineas”.

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. 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 puede observar en el diagrama, esta clase tiene dos constructores sobrecargados. Uno de ellos se usa únicamente para la creación de la primera línea, y el otro funciona cuando la Red ya tiene más de una línea. Ahora bien, la razón de esto es que ambas situaciones son totalmente distintas entre sí, por lo tanto era necesario pensar en una forma aislada para la creación de la primera de las líneas.

La forma en que se recorren las estaciones de una línea 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 recibir como parámetro el nombre de la estación buscada. Para su funcionamiento, ambas hacen uso del atributo “primeraEstación”. 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 conecta este objeto con 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” , posteriormente se redireccionan 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 cambian sus punteros “anterior” y “siguiente” donde a este último se le asigna null, ya que de esta forma se valida el final de la línea.

El método “editarEstacion” es necesario para llevar a cabo el proceso de inserción anteriormente mencionado. Este método es el que asigna los nuevos punteros a las estaciones vecinas dependiendo los casos.

Dentro del método “eliminarEstacion” se recorre la línea y se destruye el objeto deseado (validando que no sea de transferencia) y teniendo en cuenta la liberación de la memoria dinámica correspondiente. Por otro lado, el método “eliminarTodaEstacion” se usa cuando se quiere eliminar todas las estaciones existentes dentro de la línea (necesario al momento de eliminar por completo una línea). Dentro de este método, primero se valida la NO existencia de estaciones de transferencia dentro de la línea, luego se procede a eliminar una por una.

3.3 CLASE ESTACIÓN

Las instancias de esta clase se pueden separar en dos tipos, para ello usamos el atributo público “es_transferencia”. Cuando una estación es de transferencia, el valor de “num_conexiones” puede variar durante la ejecución del programa dependiendo del número de líneas que atraviesen por esta estación. Ahora bien, en caso de no ser de transferencia este valor se mantendrá en 1.

Las estaciones de transferencia tienen su propio constructor sobrecargado, esto debido a que la forma en que se inicializan los atributos tipo arreglo es totalmente diferente y se deben realizar por separado.

Por otro lado, tenemos las variables “anterior”, “siguiente” y sus respectivos tiempos. las cuales pueden ser vistas como un arreglo de direcciones de objetos de tipo estación. La cantidad de elementos que conforman este arreglo depende del número de la variable “num_conexiones”, por lo que se puede inferir que para estaciones que no son de transferencia solo existe una estación anterior y siguiente. Sin embargo, este arreglo cobra sentido para los casos en los que se tenga una estación de tipo transferencia, ya que puede haber multiplicidad de estaciones anteriores y siguientes.

Cabe aclarar que los arreglos inicializados para cada estación de transferencia inician con un tamaño de 10 y una vez se llega a este tope se aumenta su capacidad. Teniendo en cuenta lo anterior, es necesario mencionar la funcionalidad del método “redimensionar”. Se hace una llamada a este método una vez la variable “num_conexiones” alcanza un tope el cual está siempre dado por múltiplos de 10 y funciona de manera similar a la propiedad “capacity” de los arreglos tipo vector. Este método copia los elementos anteriores y los inserta en un arreglo con el siguiente tamaño. Es claro el gasto computacional que conlleva, sin embargo se ha implementado de la mejor manera para evitar un gasto excesivo.

Ahora bien, para el caso de las estaciones de transferencia se necesita saber cual es la estación anterior y siguiente dependiendo de la línea que se esté recorriendo. Es por ello que existe una variable llamada “lineas_queCruzan” la cual es un arreglo que, como su nombre lo indica, guarda los nombres de las líneas que cruzan por una estación de transferencia (en caso de no ser una estación de transferencia este arreglo toma un tamaño 0 desde el constructor sobrecargado). Esta variable tiene como objetivo que, al momento de estar recorriendo una línea específica, se indexe su nombre dentro de este

arreglo, y esa posición coincidirá con las posiciones de sus respectivas estaciones anteriores y siguientes que se guardaron dentro de los arreglos anteriormente mencionados.

3.4 CÁLCULO DEL TIEMPO ENTRE ESTACIONES

Dentro de la clase red existe un método el cual nos servirá para calcular el tiempo entre estaciones de una línea específica. Dentro de este método se le pedirá al usuario la línea a recorrer, estación de inicio y estación final. Luego, haciendo uso del correspondiente puntero “primeraEstación” se hace una búsqueda de ambas estaciones ingresadas. Una vez validada toda esa parte, se inicia un proceso en el que hacemos uso de los atributos pertenecientes a cada estación los cuales conectan a cada una con sus estaciones vecinas. Además, cada estación tiene atributos de tipo entero llamados “tiempo_siguiente” y “tiempo_anterior” los cuales se suman de forma acumulada para así dar un total una vez se recorre la linea por las estaciones deseadas.

Ahora bien, es importante señalar en esta parte la importancia de indexar los nombres de las líneas a las cuales pertenece una estación de tipo transferencia. Ya que, como se mencionó anteriormente en el punto 3.3, las estaciones de transferencia pueden seguir varios caminos dependiendo de la línea recorrida, y como se puede intuir, los tiempos van a cambiar dependiendo del camino que se tome. Cuando se indexa la línea actual, esta posición coincide con la respectiva posición de los tiempos y de las estaciones vecinas las cuales están guardadas dentro de los arreglos de la estación de transferencia.

4. 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.

5. PROBLEMAS DE DESARROLLO Y EVOLUCIÓN DE LA SOLUCIÓN

- Al momento de pensar en las estaciones de transferencia, presentamos problemas al definir cómo sus arreglos guardan sus conexiones. Sin embargo, luego de un largo análisis decidimos que la mejor solución era crear arreglos dinámicos los cuales guarden en un orden específico los datos correspondientes a sus estaciones vecinas. Este orden fue importante al momento de recorrer las líneas.
- Debido al indefinido número de conexiones que puede tener una estación de transferencia, se podía llegar a un límite de memoria en estos arreglos, fue por ello que tomamos la decisión de implementar funcionalidades para redimensionar estos arreglos por medio de copias en arreglos más grandes para luego liberar los originales. Esta implementación se hizo teniendo en cuenta todos los criterios de eficiencia posibles.
- Al momento de querer crear objetos de tipo línea, nos percatamos que la inicialización de estos objetos cambia radicalmente entre la creación de la primera línea de una red y la creación de una línea nueva. Es por ello que se tomó la decisión de implementar constructores sobrecargados para afrontar estos dos casos. Esto debido a que, a diferencia de una línea nueva, la primera de las líneas no recibe como parámetro ninguna estación de transferencia ya existente con la cual conectarse.
- Dentro de la clase Estación presentamos un problema similar al anterior pero esta vez entre constructores destinados a estaciones de transferencia y estaciones normales. Esto debido a que las estaciones de transferencia tienen una inicialización en la que sus atributos necesitan ser arreglos de los cuales se pueda indexar, sin embargo, este no era el caso para estaciones normales, es por esto que se tomó la decisión de separar ambas inicializaciones.