Informe Desafío II

Integrantes: Juan José Balvin Torres – Juan Pablo González Blandón

Análisis del problema y consideraciones para la alternativa de la solución propuesta

El desafío propuesto consiste en modelar una red metro con ciertas características expuestas en el PDF del desafío, teniendo en cuenta estas mismas y analizando como podemos llevar a cabo el desafío, la solución planteada es la siguiente: Pudimos identificar dos puntos clave para su desarrollo, estas son una vista general la cual llamaremos "red" y una vista un poco más reducida la cual llamaremos "línea" ahora bien por que decidimos llevar nuestro enfoque a esas dos clases principales, lo pensamos de la siguiente manera: Necesitamos modelar un comportamiento total sobre la red metro y realizar ciertos cambios y operaciones sobre esta que influirán no solo en una línea sino en toda la red en general así que para ello es necesario modelar la vista general, sin embargo entre las operaciones que se deben de realizar sobre esta, no se incluyen aspectos más internos de la red como las estaciones que tienen una pertenencia directa a las líneas de la red, para ello se modela la clase línea, para llevar control sobre estas y sus tiempos.

Para implementar la solución la dividiremos en una estructura bidimensional, cuyo esquema es el siguiente:

	Objetos line
Línea 1	Nombre objeto1
Línea 2	Nombre objeto2
. . .	. . .
Línea N	Nombre objetoN

Este será el atributo principal de la clase red encargada del modelamiento de toda de la red.

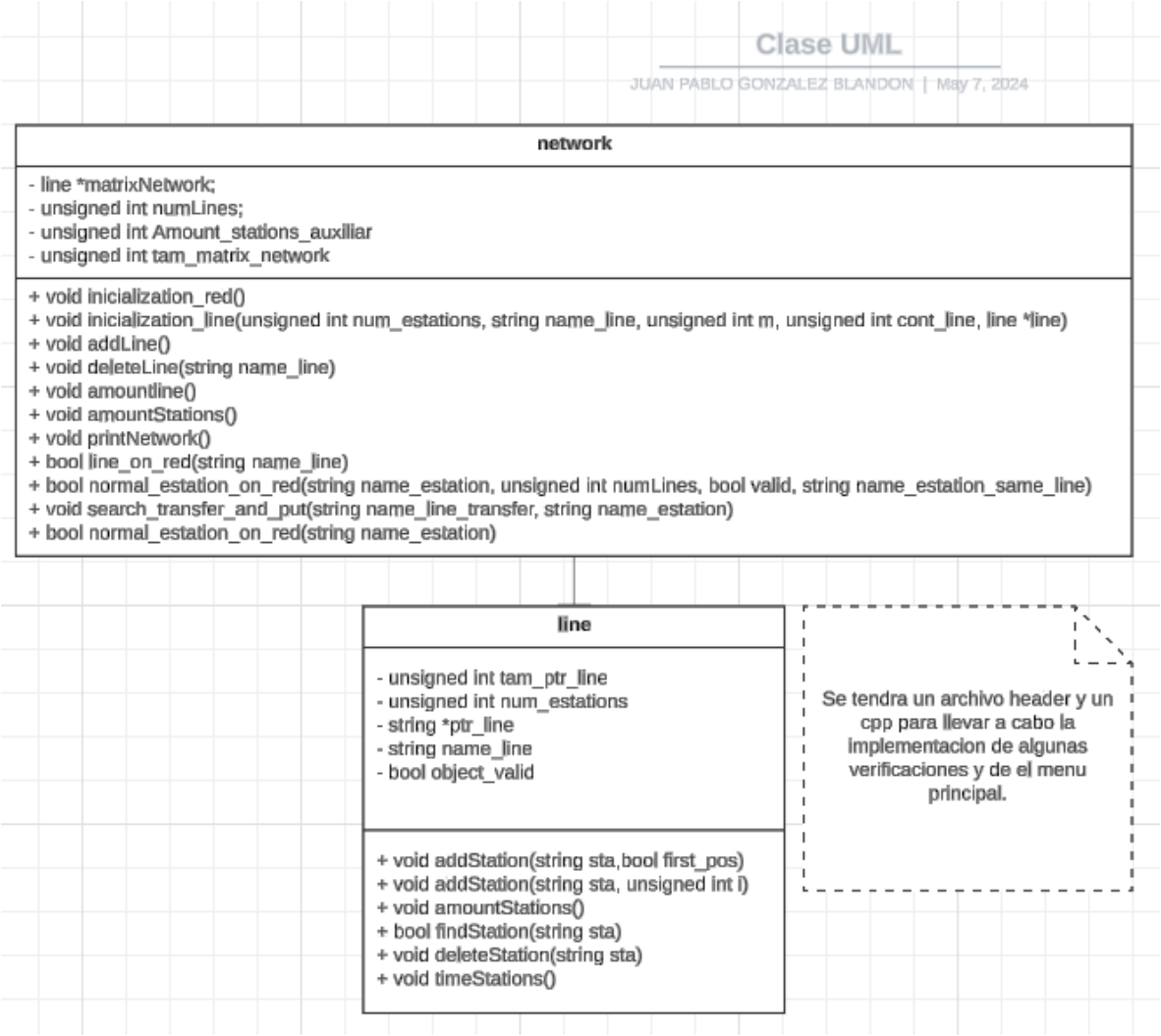
Ahora bien, para la clase línea se tendrá una estructura unidimensional extraída de la estructura bidimensional la cual será encargada del modelamiento de una línea en específico y realizar los cambios y operaciones que quiera sobre esta, su estructura será la siguiente:

	Estación1	Tiempo	Estación2	Tiempo	EstaciónN
Nombre Objeto	Nombre	Tiempo estación 1-2	Nombre	Tiempo estación (N-1)- N	Nombre

Este será el atributo principal de la clase línea en la cual tendremos almacenado los tiempos de llegada de una estación a otra, ya que no tenemos la posibilidad de usar mapas, las posiciones de serán relacionadas de la siguiente forma, para obtener el tiempo de llegada de una estación a otra

simplemente necesitaras pararte en donde este ubicado el nombre de la estación y devolverte una posición para conocer el tiempo a la previa estación o adelantarte una para conocer el tiempo de la siguiente estación.

Diagrama de clases



Algoritmos implementados

- inicialization_red & inicialization_line

  Estos algoritmos implementados en la solución del problema fueron pensados como un mecanismo para inicializar la red con objetos line; los cuales son inicializados de forma individual por inicialization_line, en ambos algoritmos se realizan constantes verificaciones (métodos) sobre toda la red para no permitir la repetición de estaciones, líneas etc.

- add_line

  La siguiente sección del código reutiliza código de la parte de inicialization_red para pedir la línea a agregar y usa inicialization_line para poder crear esta misma y meterla en el arreglo de objetos.

- delete_line

  Este método se encargará de realizar una búsqueda sobre el arreglo de objetos y cuando concuerde con el nombre de la fila la cual desea eliminar buscara si tiene estaciones de transferencia, en caso de que tenga no dejara eliminar, cabe aclarar que siguiendo las reglas del Desafío II solo se debería poder eliminar una red que contenga solo una línea.

- amountline & amountStations

  estos dos algoritmos no tienen mucho de especial ya que tenemos atributos arraigados propiamente tanto a la cantidad de líneas como a la cantidad de estaciones de una línea en específico. El único que es un poco diferente es el de amountStations ya que se necesito recorrer el arreglo de objetos e ir sumando el atributo numestations, luego se le resta con las repeticiones de las estaciones de transferencias ya que estas solo deben de contar como una estación.

- print_network

  Este sencillo programa lo que hace es recorrer todo el arreglo de objetos incluyendo su contenido almacenado en el puntero ptr_line para poder generar la impresión de la red metro, además tiene una pequeña verificación para saber si la red metro esta vacía e imprimir dicho mensaje.

- addStation

  la sobrecarga de este método se empleo ya que necesitaba un caso en el cual no preguntara si deseaba agregarla al final o en el medio, simplemente la agregara de forma “contigua” digo contigua ya que en realidad se están agregando cada dos espacios porque en el medio está el tiempo que tarda de llegar de una estación a otra (se agrega a ptr_line).

- AmountStation & findStation

  amountStation es básicamente imprimir por pantalla el atributo de cada objeto el cual contiene el numero de estaciones de dicha línea, y findStation se encarga de recorrer ptr_line en busca de el nombre de la estación que se ingresa para retornarle al usuario si esa estación se encuentra allí o no.

- deleteStation

  básicamente recorre ptr_line si y solo si la estación que se ingresa no es de transferencia ya que recordemos que no se puede eliminar la estación de transferencia, en caso de que se pueda eliminar lo que se hará es dejar esa posición vacía y correr los elementos del arreglo.

- Time_station

  Este método es el encargado de preguntarle al usuario en que línea y cuales estaciones son las que desea calcular el tiempo, en el momento en que ambas estaciones concuerdan y se encuentran procederemos a buscar cual es primero que cual para saber qué tiempos debemos sumar, luego nos apoyaremos en la librería ctime para tomar la hora actual y esta misma sumarle lo correspondiente e imprimir la hora de llegada, cabe aclarar que los tiempos entre estaciones los estamos tomando en minutos.

- El resto de métodos no mencionados de forma explicita en el informe son verificaciones cuyo funcionamiento es bastante similar. A groso modo este itera sobre la red en busca de la verificación correspondiente.

  Problemas de desarrollo que afronto – consideraciones a tener en cuenta en la solución

  El primer problema evidenciado fue la hora de definir las clases ya que incluso llegamos a tener un problema gravísimo del diseño de estas mismas como lo es la dependencia circular, pero con un estudio de la problemática y un razonamiento sobre esta se pudo solventar de manera satisfactoria.

  El gestionamiento de los arreglos en el caso en el que se desee, eliminar o agregar un elemento que no se encuentre en los extremos del arreglo también presento no un problema sino una consideración del problema la cual se necesitó pensar bastante.

  Fue algo complejo definir bien como seria la distribución de la estructura de datos ya que teníamos como idea principal un triple puntero, pero al empezar a codificar nos dábamos cuenta de que era algo ineficiente.

  Se presento un problema de desarrollo de la solución cuando se necesitó la redimensión de nuestra estructura de datos ya que implica crear copia de esta cada vez que desee agregar tanto una línea como una estación y esto requiere cierto costo computacional, como solución se estudiaron redes de metro por Latinoamérica y se sacó un promedio de líneas y estaciones para obtener de esta forma una expectativa de crecimiento de nuestra red y evitar el costo computacional de crear copias

  Estaciones: 285.1 aproximadamente 285 estaciones
  Líneas: 15.7 aproximadamente 16 líneas

Debido a la abstracción del problema y tomando en cuenta de que es educativo reduciremos la cantidad de estaciones en un 95 por ciento es decir la expectativa de las estaciones será de 15 estaciones y las líneas la reduciremos un 35 por ciento, la expectativa será de 10 líneas. **Cabe aclarar que se agregara una funcionalidad para redimensionarla en caso de que esta expectativa se supere.**

Avance

El avance de la solución se puede evidenciar en el repositorio con los commits diarios realizados.