DESAFÍO II: SIMULACIÓN DE UNA RED METRO

Juan David Ortiz

Ramón Francisco Santacruz

Estudiantes Informática II

Universidad de Antioquia

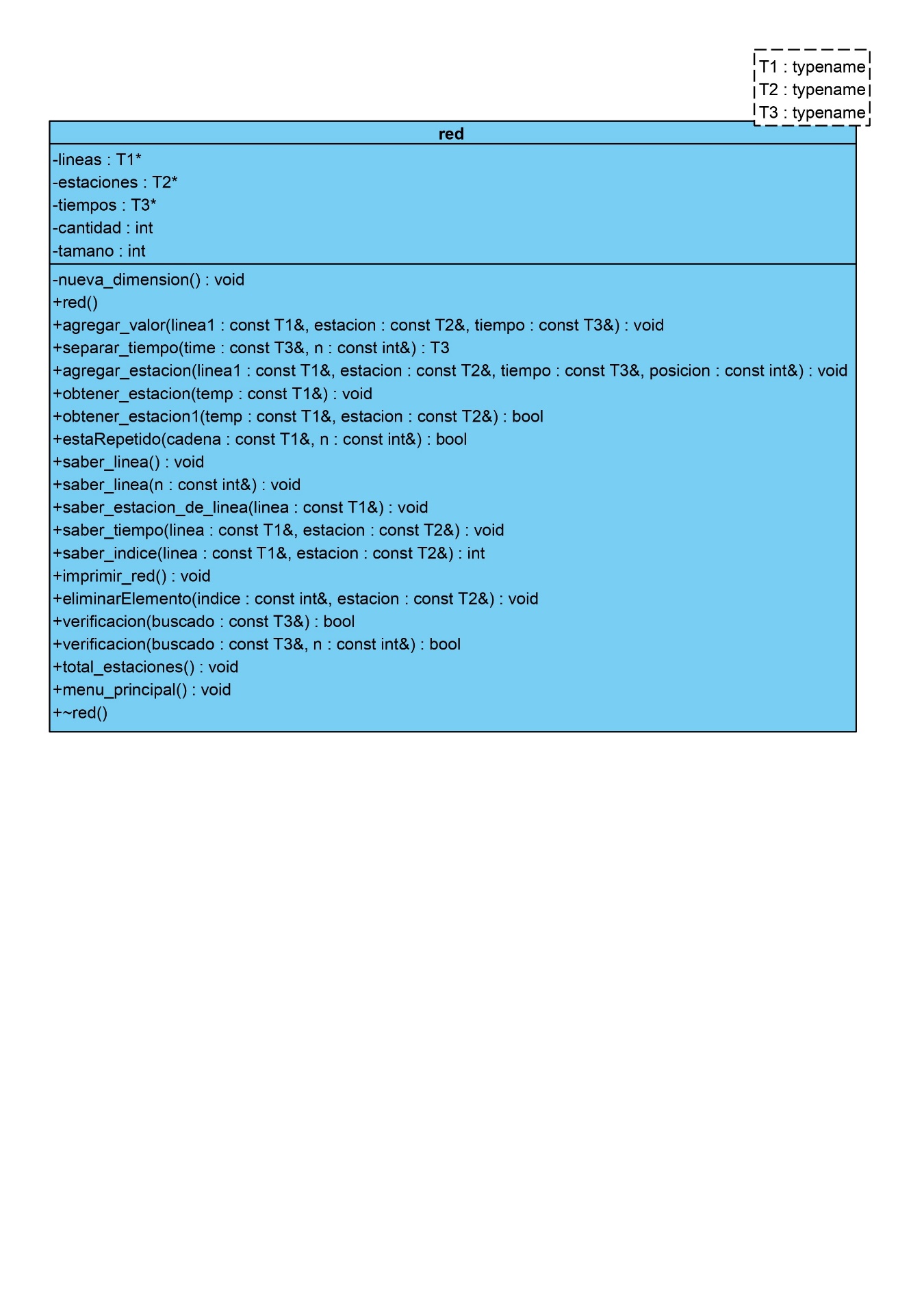
## a. Análisis del problema y consideraciones para la alternativa de solución propuesta.

Para abordar el problema, dedicamos tiempo a una lectura exhaustiva del archivo desafío 2. Nuestro objetivo fue comprender a fondo el problema antes de proceder con la elaboración de cualquier diagrama de flujo o estrategia de solución. Enfocamos nuestra atención en comprender las complejidades del problema y evaluar las posibles alternativas de solución de manera integral.

Una de las principales estrategias que adoptamos fue dividir todo el programa en funciones de la respectiva clase. Esto nos permitió realizar un trabajo colaborativo más efectivo y organizado.

Para la solución del desafío planteamos la creación de una plantilla que contiene las líneas, las estaciones de cada línea y los tiempos relacionados entre estaciones. Podemos crear y eliminar estaciones, agregar líneas y visualizar la red.

## b. Diagrama de clases.



c. Algoritmos implementados.

|  |  |
| --- | --- |
| **FUNCIÓN** | **DESCRIPCIÓN** |
|  | Este es el constructor de la clase “red” donde se inicializan las siguientes variables   1. **cantidad(10)**: Inicializa el miembro **cantidad** de la clase con el valor **10**. 2. **tamano(0)**: Inicializa el miembro **tamano** de la clase con el valor **0**. 3. **líneas=new T1[cantidad]**: Crea un arreglo dinámico de tipo **T1** con tamaño **cantidad** y asigna su dirección de memoria al puntero **líneas**. 4. **estaciones=new T2[cantidad]**: Crea otro arreglo dinámico de tipo **T2** con tamaño **cantidad** y asigna su dirección de memoria al puntero **estaciones**. 5. **tiempos=new T3[cantidad]**: Crea un tercer arreglo dinámico de tipo **T3** con tamaño **cantidad** y asigna su dirección de memoria al puntero **tiempos**. |
|  | La función **~red()** es un destructor que libera la memoria asignada dinámicamente para los arreglos **lineas**, **estaciones** y **tiempos**. Esto se hace utilizando el operador **delete[]** seguido del nombre del puntero que apunta al arreglo.  El uso de **delete[]** libera la memoria que fue asignada con **new[]**, asegurando que los recursos utilizados por los arreglos dinámicos se devuelvan al sistema operativo cuando la instancia de la clase se destruya o salga de ámbito. Es una buena práctica realizar esta liberación de memoria en el destructor para evitar fugas de memoria y asegurar la gestión adecuada de los recursos. |
|  | Esta función **agregar\_valor** tiene como objetivo añadir un conjunto de valores (**linea1**, **estacion** y **tiempo**) a los arreglos dinámicos **lineas**, **estaciones** y **tiempos**. Si el tamaño actual es igual o mayor que la capacidad (**cantidad**), se llama a la función **nueva\_ dimension()** para redimensionar los arreglos y hacer espacio para más elementos. |
|  | 1. Declara dos variables de tipo **string**, **primero** y **segundo**. 2. Busca la posición de la coma en la cadena **time** usando **time.find(',')**. 3. Asigna a **primero** la subcadena que va desde el inicio hasta justo antes de la coma (**comaPos**). 4. Asigna a **segundo** la subcadena que va después de la coma (**comaPos + 1**). 5. Comprueba el valor de **n**:    * Si **n** es igual a 1, devuelve la variable **primero**.    * Si **n** es igual a 2, devuelve la variable **segundo.** |
|  | Esta función **agregar\_estacion** parece ser un método para insertar una estación en una posición específica dentro de una estructura que almacena líneas, estaciones y tiempos. Aquí tienes un análisis paso a paso de la función:   1. Primero, verifica si la estación que se desea agregar ya existe en la línea especificada. Si es así, muestra un mensaje y devuelve -1 para indicar que no se pudo agregar la estación. 2. Luego, comprueba si la posición en la que se quiere agregar la estación está dentro del rango válido. Si no lo está, muestra un mensaje de error y devuelve -1. 3. Si la cantidad actual de estaciones es mayor o igual a la capacidad, llama a la función **nueva\_dimension** para redimensionar la estructura. 4. A continuación, mueve las estaciones existentes para abrir espacio en la posición deseada. 5. Agrega la nueva estación en la posición especificada. 6. Si la posición es diferente de la primera o la última, actualiza los tiempos de las estaciones vecinas para mantener la coherencia. 7. Si la posición es la última, actualiza el tiempo de la nueva estación y agrega la estación al final. 8. Incrementa el tamaño de la estructura y muestra un mensaje indicando que la estación se creó correctamente. |
|  | Esta función obtener\_estacion permite contar la cantidad de estaciones que tiene una línea específica y luego muestra ese número. Aquí tienes una descripción paso a paso de lo que hace la función:  Inicia una variable con en 0 para contar la cantidad de estaciones encontradas.  Recorre todas las estaciones en el arreglo.  Si encuentra una estación que coincide con la línea especificada (temp), incrementa el contador con.  Después de recorrer todas las estaciones, muestra la cantidad total de estaciones encontradas para la línea especificada. |
|  | Esta función **obtener\_estacion1** permite buscar una combinación específica de línea y estación dentro de la estructura de datos. Aquí está la descripción de lo que hace paso a paso:   1. Inicia una variable **con** en 0 para contar cuántas veces se encuentra la combinación de línea y estación. 2. Recorre todas las estaciones en el arreglo. 3. Verifica si la línea en la posición **i** coincide con la línea especificada (**temp**). 4. Si hay coincidencia en las líneas, verifica si la estación en la posición **i** coincide con la estación especificada (**estacion**). 5. Si ambas coincidencias ocurren, incrementa el contador **con**. 6. Después de recorrer todas las estaciones, si **con** es diferente de 0, devuelve **true**, lo que indica que se encontró al menos una coincidencia.   Si **con** es 0, devuelve **false**, lo que indica que no se encontró ninguna coincidencia. |
|  | 1. Recorre el arreglo de **lineas** hasta el índice **n**, que se pasa como argumento. 2. En cada iteración, compara la cadena especificada (**cadena**) con la cadena en la posición **i** del arreglo de **lineas**. 3. Si encuentra una coincidencia, devuelve **true**, lo que indica que la cadena está repetida en el arreglo. 4. Si no encuentra ninguna coincidencia después de recorrer todo el arreglo hasta el índice **n**, devuelve **false**, lo que indica que la cadena no está repetida en el arreglo. |
|  | Esta función **saber\_linea** muestra las líneas existentes en el sistema. Aquí está cómo funciona:   1. Inicia un contador **con** en 1 para enumerar las líneas. 2. Recorre el arreglo de **lineas** hasta el tamaño actual (**tamano**). 3. Para cada línea en el arreglo, verifica si esa línea ya ha sido mostrada anteriormente (es decir, si no está repetida hasta el índice actual). 4. Si la línea no está repetida, la muestra en la salida estándar junto con su número de enumeración (**con**).   Incrementa el contador **con** para la siguiente línea que se mostrará. |
|  | Esta versión de la función **saber\_linea** cuenta la cantidad de líneas existentes en el sistema sin mostrar las líneas en sí. Funciona de la siguiente manera:   1. Inicia un contador **con** en 0 para contar las líneas. 2. Recorre el arreglo de **lineas** hasta el tamaño actual (**tamano**). 3. Para cada línea en el arreglo, verifica si esa línea ya ha sido contada anteriormente (es decir, si no está repetida hasta el índice actual). 4. Si la línea no está repetida, incrementa el contador **con**.   Al final del bucle, muestra la cantidad total de líneas contadas en la salida estándar. |
|  | Esta función **saber\_estacion\_de\_linea** muestra las estaciones asociadas a una línea específica. Aquí está cómo funciona:   1. Recibe como parámetro la línea de la cual se desea obtener las estaciones (**linea**). 2. Inicia un contador **con** en 1 para enumerar las estaciones de manera ordenada. 3. Recorre el arreglo de **lineas** y **estaciones** hasta el tamaño actual (**tamano**). 4. Para cada índice **i**, verifica si la línea en el índice **i** es igual a la línea proporcionada como parámetro (**lineas[i] == linea**). 5. Si la línea es igual, muestra el número de estación (**con**) seguido del nombre de la estación (**estaciones[i]**). 6. Incrementa el contador **con** después de mostrar cada estación asociada a la línea especificada. |
|  | Esta función **saber\_tiempo** busca el tiempo asociado a una estación de una línea específica. Aquí está su funcionamiento:   1. Recibe dos parámetros: **linea** que representa la línea de interés y **estacion** que representa la estación de la línea de la cual se desea conocer el tiempo. 2. Itera sobre los arreglos **lineas**, **estaciones** y **tiempos** hasta el tamaño actual (**tamano**). 3. En cada iteración, verifica si la línea en el índice **i** coincide con la línea proporcionada (**lineas[i] == linea**). 4. Si la línea coincide, verifica si la estación en el índice **i** coincide con la estación proporcionada (**estaciones[i] == estacion**). 5. Si ambas coinciden, muestra el tiempo asociado a esa estación (**tiempos[i]**). |
|  | En la función **saber\_indice**, buscas el índice correspondiente a una combinación de línea y estación. Aquí tienes el funcionamiento:   1. Recibe dos parámetros: **linea** que representa la línea de interés y **estacion** que representa la estación de la línea de la cual se desea obtener el índice. 2. Itera sobre los arreglos **lineas** y **estaciones** hasta el tamaño actual (**tamano**). 3. En cada iteración, verifica si la línea en el índice **i** coincide con la línea proporcionada (**lineas[i] == linea**). 4. Si la línea coincide, verifica si la estación en el índice **i** coincide con la estación proporcionada (**estaciones[i] == estacion**).   Si ambas coinciden, devuelve el índice **i**. |
|  | Esta función **imprimir\_red** se encarga de mostrar la información almacenada en los arreglos **lineas** y **estaciones** de tu clase. Aquí está el funcionamiento:   1. Comprueba si la red está vacía (si **tamano** es igual a cero). Si es así, imprime un mensaje indicando que la red está vacía y sale de la función. 2. Si la red no está vacía, comienza imprimiendo la información de la primera línea y estación (**lineas[0]** y **estaciones[0]**). 3. Luego, inicia un bucle **for** desde **i = 1** hasta **tamano - 1**. 4. En cada iteración, verifica si la línea en la posición **i** es diferente a la línea en la posición **i - 1**. Si es diferente, significa que estamos en una nueva línea y se imprime la información de la nueva línea y su estación correspondiente. 5. Si la línea es la misma que la anterior, solo se agrega la estación a la lista de estaciones para esa línea, separada por comas. 6. Al final del bucle, se imprime una nueva línea para separar la información. |
|  | 1. Verifica si la estación que se desea eliminar es una estación de transferencia utilizando la función **verificacion** . Si es una estación de transferencia, muestra un mensaje indicando que la línea no se puede eliminar porque pertenece a una estación de transferencia. 2. Si la estación no es una estación de transferencia y el índice está dentro del rango válido (mayor o igual a cero y menor que **tamano**), se procede a eliminar el elemento en esa posición. 3. Se utiliza un bucle **for** que comienza desde la posición **indice** hasta **tamano - 1**, moviendo cada elemento una posición hacia adelante para llenar el espacio dejado por el elemento eliminado. 4. Se reduce el tamaño de la red (**tamano**) en uno para reflejar la eliminación del elemento. 5. Si el índice está fuera de rango, muestra un mensaje de error indicando que el índice está fuera de rango. |
|  | 1. Recorre el arreglo **lineas** utilizando un bucle **for** desde **i = 0** hasta **tamano - 1**. 2. Compara cada elemento del arreglo **lineas** con el valor **buscado**. 3. Si encuentra una coincidencia, muestra un mensaje indicando que la línea no se puede eliminar porque pertenece a una estación de transferencia y devuelve **true**. 4. Si no encuentra ninguna coincidencia después de recorrer todo el arreglo, muestra un mensaje indicando que la línea que ingresaste no pertenece a la red metro y devuelve **false**. |
|  | La función **verificacion** que proporcionaste busca un elemento en el arreglo **lineas**. Si encuentra una coincidencia con **buscado**, devuelve **true**; de lo contrario, devuelve **false**. El parámetro **n** no se usa en esta función |
|  | La función **total\_estaciones** simplemente imprime el número total de estaciones en la red Metro, utilizando la variable **tamano** que almacena este valor. |
|  | Lo que hace esta función es crear el menú principal donde vamos a realizar toda la simulación de toda la red metro y usar todas las funciones mencionadas anteriormente. |

## d. Problemas de desarrollo que afrontó.

El principal problema estuvo relacionado a como abordar el desafío para su solución. Realizamos varios cambios de planes a medida íbamos cayendo en cuenta de una manera distinta de hacer lo que queríamos de una manera más sencilla que nos permitiese una mayor eficiencia a la hora de implementar los algoritmos, la lectura del código y un mejor manejo de memoria dentro del programa.

También tuvimos inconvenientes con sintaxis que nos hicieron perder tiempo tratando de encontrar el error. El tiempo sin lugar a duda fue un recurso muy limitado al momento de desarrollar nuestra solución, pero finalmente logramos la implementación de nuestro programa a tiempo

## e. Evolución de la solución y consideraciones para tener en cuenta en la implementación.

Inicialmente consideramos crear 3 clases, estaciones, líneas y red, pero nos dimos cuenta de que no era muy eficiente, por lo que consideramos crear 2 plantillas para implementarlas en la creación de las líneas y estaciones. El objetivo era crear las estaciones como un par clave-valor donde la clave sería el nombre y el valor un puntero al vector con los tiempos, así mismo, las líneas funcionarían de la misma manera, de clave el nombre de la línea y valor un puntero a un vector con las estaciones. Así la red sería un vector con todas las líneas.

Descartamos esta opción dado que tuvimos inconvenientes a la hora de juntar las 2 plantillas e implementarlas en cada clase, así que decidimos crear una sola plantilla, que almacena 3 valores, la línea, estación y tiempo (anterior, siguiente). Esta alternativa nos permitió un manejo más sencillo de los distintos valores almacenados en la red.

Se han implementado las funciones para casi todas las opciones del menú, ya logramos imprimir la red, crear estaciones y redes, así como almacenar y relacionar los tiempos entre una estación y otra. Aún falta implementar la opción para escoger la posición en la que se desea agregar la nueva estación (inicio de la línea, en medio de 2 estaciones existentes o al final de la línea)

Finalmente tenemos las funciones completas, incluyendo el conteo total de estaciones en la red, se puede escoger la posición en la que se creará la estación en una línea, así como las verificaciones correspondientes para evitar problemas de ejecuciones en el programa. Tenemos la solución final al desafío.