**DESAFIO II INFORMÁTICA II**

**JUAN DIEGO TORO FRANCO**

**LUIS ANGEL OSORIO ALANDETE**

**INGENIERÍA EN TELECOMUNICACIONES Y**

**ELECTRONICA**

**UNIVERSIDAD DE ANTIOQUIA**

**FACULTAD DE INGENIERÍA**

**PRIMER SEMESTRE 2024**

**2024**

**DESAFIO II INFORMÁTICA II**

**JUAN DIEGO TORO FRANCO**

**LUIS ANGEL OSORIO ALANDETE**

**INGENIERÍA EN TELECOMUNICACIONES Y**

**ELECTRONICA**

**INFORME DE SOLUCIÓN AL DESAFIO**

**PLANTEADO**

**ANIBAL GUERRA**

**AUGUSTO SALAZAR MARICON**

**DOCENTES**

**UNIVERSIDAD DE ANTIOQUIA**

**FACULTAD DE INGENIERÍA**

**PRIMER SEMESTRE 2024**

**2024**

**TABLA DE CONTENIDO**:

1. CONTEXTO DEL PROBLEMA…………………………………………………………………………………………… 1
2. OBJETIVOS DEL INFORME……………………………………………………………………………………………….2
3. METODOLOGÍA USADA…………..………………………………………………………………………………………2
   1. ALGORITMOS Y TÉCNICAS EMPLEADAS………..………………………………………………………..…3
   2. HERRAMIENTAS USADAS ………………………………………………………………………………………….4
4. ESQUEMA DE LA SOLUCIÓN ………………………………………………………………………………………….. 5
5. CÓDIGOS Y EJEMPLOS DE EJECUCIÓN …………………………………………………………………………….6
6. CONCLUSIONES Y POSIBLES ERRORES …………………………………………………………………………….9
7. REFERENCIAS Y BIBLIOGRAFÍA ……………………………………………………………………………………….10
8. **CONTEXTO DEL PROBLEMA**

Se plantea un problema de modelación de una situación en el mundo real, en este caso es una red de metros, se necesita la creación de una simulación de esta misma red, con diferentes funcionalidades para que el usuario pueda modificar, editar, mostrar y operar su red de metro

Es importante tener en cuenta que esta simulación debe ser fiel al funcionamiento de una red de metro, en donde se tienen diferentes condiciones como las siguientes:

* El sistema puede tener diferentes líneas, cada línea tiene una cantidad x de estaciones
* El sistema al tener diferentes líneas estas deben estar conectadas de forma que haya estaciones de transición las cuales los usuarios de esta red de metro usarán para cambiar de línea
* Se debe tener en cuenta los tiempos entre estaciones, para esto se definen en minutos a través de la consola
* El usuario que cree la red de metro puede eliminar líneas, eliminar estaciones y esto modificará el funcionamiento de la red y los tiempos establecidos

**Para este problema es necesario que cumpla los siguientes requisitos:**

1. Agregar una estación a una línea, en los extremos o en posiciones intermedias.

2. Eliminar una estación de una línea. No se pueden eliminar estaciones de

transferencia.

3. Saber cuántas líneas tiene una red Metro.

4. Saber cuántas estaciones tiene una línea dada.

5. Saber si una estación dada pertenece a una línea específica.

6. Agregar una línea a la red Metro.

7. Eliminar una línea de la red Metro (sólo puede eliminarse si no posee estaciones

de transferencia).

8. Saber cuántas estaciones tiene una red Metro (precaución con las estaciones de

transferencia).

**Se necesita que el desarrollador use el paradigma de programación orientada a objetos POO, uso de punteros y manejo eficiente de memoria y eficiencia de algoritmo.**

1. **OBJETIVOS DEL INFORME**

Hacer seguimiento de la solución que se implementará al problema utilizando estrategias para el correcto funcionamiento de el algoritmo realizado. Principalmente se toma como una plantilla para el tipo de solución que se implementará planeando, de forma específica, el cómo se llevará a cabo la solución.

En este informe se llevará registro de:

* Planeación de solución.
* Metodología usada.
* Pruebas de escritorio.
* Explicación del algoritmo y ejemplos de funcionamiento
* Posibles fallos y mejoras a realizar
* Conclusiones del desarrollo.

1. **METODOLOGÍA USADA**

En este problema se usará el paradigma de programación orientada a objetos, esto debido a que el problema al ser una simulación de un problema en el mundo real requiere soluciones aplicables a este. El paradigma de la programación orientada a objetos es una herramienta que nos permite como desarrolladores abordar estos problemas como la red metro de forma mas poderosa, esto debido a su capacidad para modelar entidades y relaciones de manera natural, facilita la reutilización de código y la organización modular, nos permite como desarrolladores adaptarnos a diferentes contextos.

* 1. **Algoritmos y técnicas empleadas**

En los algoritmos que desarrollamos usamos técnicas como memoria dinámica, punteros y clases, encapsulación, instancias de objetos, métodos y polimorfismo de tal forma que:

La memoria dinámica se utilizó para el cambio de tamaños en arreglos en tiempo de ejecución y esto nos permitió flexibilidad esto también para hacer la búsqueda de datos guardados en memoria de forma más eficiente y los punteros nos ayudan precisamente al manejo de esta memoria dinámica por medio de asignaciones tipo punteros

Las clases es la parte más importante del código, este concepto hace referencia a la modelación de objetos en la vida real, este objeto tiene funcionalidades y características que hacen que ese objeto sea considerado como tal, en este problema se modelaron diferentes clases ya que una red de metro tiene en común que tanto líneas como estaciones tienen sus propias características que las diferencian, luego tenemos otras clases más específicas para el código como menús y red. El uso de estas clases también nos permitió aprovechar el polimorfismo siendo esto el como el comportamiento cambia según el contexto del problema.

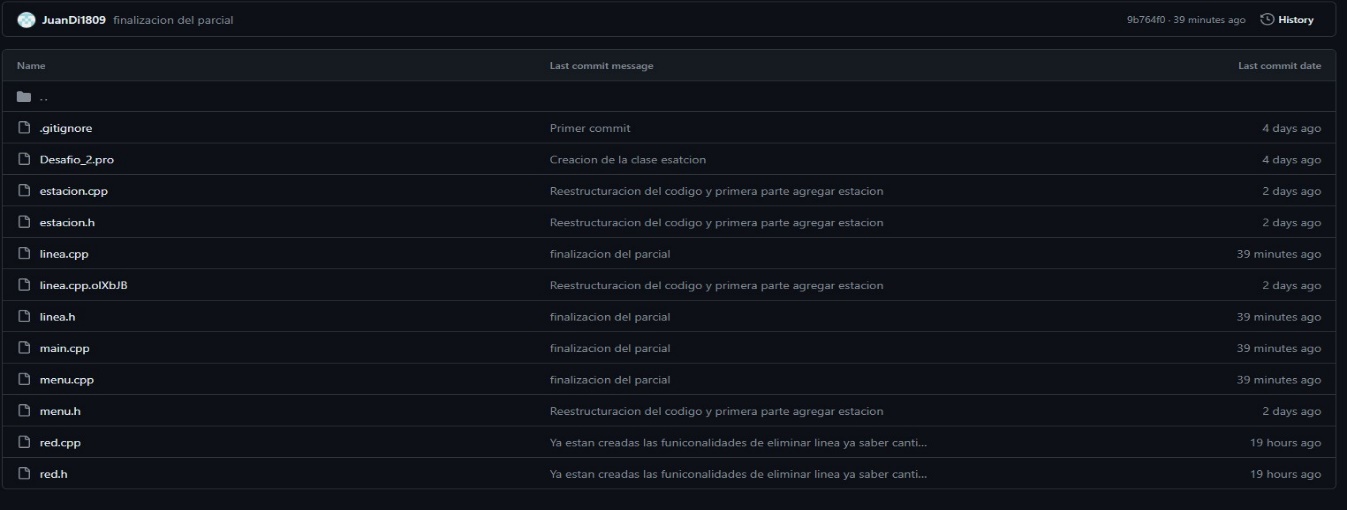
* 1. **Herramientas usadas**

Usamos el intérprete QT como entorno de desarrollo, Git como software de control de versiones y de forma paralela utilizamos GitHub como repositorio de la solución desarrollada, estas dos últimas con el propósito de facilitar el trabajo colaborativo y la actualización constante de versiones para el algoritmo.

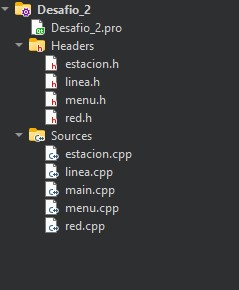
Para la documentación y un amplio entendimiento de como implementar una lógica más completa usamos la plataforma StackOverflow tomando como referencia códigos subidos por otros usuarios de funcionalidades especificas necesitadas para la complementación de las soluciones planteadas al problema.

1. **ESQUEMA DE SOLUCIÓN**

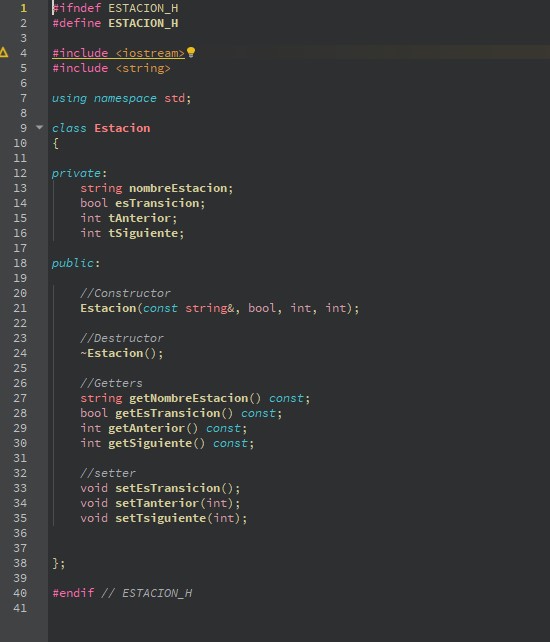
Se definieron diferentes archivos que complementan al archivo main, este archivo main es el encargado de mostrar la interfaz al usuario y que se establezca la lógica necesaria, de esta forma la modularizarían del código es efectiva y nos permite tener de la solución plantead de forma mas organizada y de forma mas especifica para que el funcionamiento sea como se espera.

Usamos git para hacer el control de versiones:

En el proyecto la vista de los headers y sus respectivos archivos es la siguiente:



El uso de clases se definió usando cabeceras, se relacionan dos archivos de la siguiente manera:



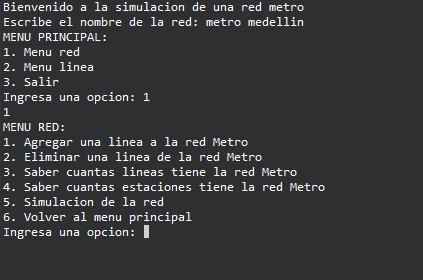
La vista del archivo que contiene los métodos y sus definiciones es la siguiente:



1. **CODIGOS Y EJEMPLOS DE EJECUCUCÍON**

En este informe solo se mostrará los menús que muestra el código y se explicarán sus funcionalidades en texto, debido a que la ejecución es extensa como para adjuntarla en este mismo informe.

Menú de red:



**Explicación y funcionalidades:**

**Opción 1:**

* Cuando se agrega una línea en la primera ejecución el algoritmo pide al usuario nombre para la línea, en las siguientes iteraciones se leerán las líneas que ya se hayan creado anteriormente y se enlistarán

**Opción 2:**

* Al eliminar una línea se hacen dos verificaciones, que haya por lo menos una línea en la red y que esta no tenga estaciones de transición, de manera paralela también se establece que las líneas tienen al menos una estación agregada

**Opción 3:**

* Esta opción cuenta cuantas líneas tiene la red, también se tiene en cuenta las que se eliminan y las que se agregan

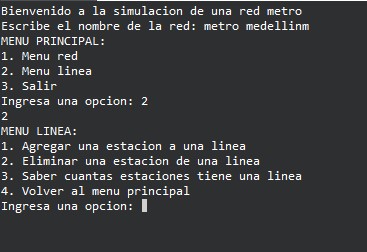
**Opción 4:**

* Este enlistará las líneas y luego el usuario debe ingresar de cual línea quiere saber la cantidad de estaciones, luego muestra la cantidad

**Opción 5:**

* Simula la red con las líneas creadas, estaciones y teniendo en cuenta sus respectivos tiempos

**Menú línea:**



**Explicación y funcionalidades:**

**Opción 1:**

* Cuando se agrega una línea se pregunta al usuario el nombre, luego se hacen las siguientes comprobaciones
  + Cuando no haya estaciones creadas anteriormente se establece el tiempo de la estación anterior a la siguiente en 0 ya que no hay estación anterior
  + Cuando hay al menos una estación se pregunta de forma lineal los tiempos de la estación anterior y de la estación siguiente
  + Si se va a agregar una estación en medio de dos estaciones, se hace un cambio de tiempos y luego se cambia el orden en el array línea el cual contiene las estaciones

**Opción 2:**

* Cuando el usuario quiere eliminar una estación se le pregunta al usuario de cual línea quiere liminar la estación, luego se le enlista las estaciones de esa línea y después se le pide un valor numérico el cual determina la posición de la cual quiere eliminar la estación, la estación se elimina si existe y si no es de transición (en este caso no se elimina) luego se cambian los tiempos de las estaciones contiguas de forma automática.

**Opción 3:**

* Se muestran las estaciones que tiene una línea

1. **CONCLUSIONES Y POSIBLES ERRORES**

Como equipo nos enfrentamos a un gran reto, este reto puso a prueba nuestras habilidades a la hora de brindar una solución a un problema que muy fácilmente puede ser necesario resolver en la vida real. El uso de POO nos brindó una forma más sencilla y ordenada de proponer la solución por su similitud con objetos de la vida real. Luego el uso de punteros y manejo de memoria nos ayudó a optimizar el funcionamiento del proyecto. En el camino se habían propuesto diferentes soluciones para este problema, cambiamos estructura de main varias veces, también se modificaron, eliminaron y agregaron métodos de las diferentes clases que creamos, esto nos dio una visión más allá de las soluciones que habíamos planteado ya que la complejidad fue establecida por que tan extenso y que tanto queríamos abarcar con la solución propuesta al final. Desde luego el código no es perfecto y se presentan diferentes problemas, incluso por la falta de tiempo y por la envergadura que tendría nuestra solución no fue posible complementar algunas funcionalidades, por ejemplo, las siguientes:

* Calcular las estaciones de la red
* Eliminar estación se presentaron fallas en casos específicos
* Agregar estación entre medias no fue posible terminar al completo su correcto funcionamiento

También identificamos fallas en nuestro código a final de este, por ejemplo, encontramos que hubo uso excesivo de la memoria por la instancia miento de objetos muchas veces en el main y el no eliminarlos correctamente, luego también descubrimos que había formas más sencillas de hacer llamados de métodos de unas clases desde otras clases por el uso de herencia o la instancia miento de objetos de manera única y no repetida como el caso del main.

**REFERENCIAS Y BIBLIOGRAFÍA**

<https://www.freecodecamp.org/espanol/news/guia-para-principiantes-de-git-y-github/>

<https://docs.github.com/es/get-started/getting-started-with-git/set-up-git>

<https://docs.github.com/es/get-started/start-your-journey/git-and-github-learning-resources>

para el uso del control de versiones