|  |  |
| --- | --- |
| Texto  Descripción generada automáticamente | **Informática II Semestre 2024 – 1**  **Desafío 2. Abril 29 de 2024** |

NOMBRES:  
  
Alejandro Naranjo Naranjo  
Juan Pablo Rivero Garay.

PUNTOS DEL INFORME:

a. Análisis del problema y consideraciones para la alternativa de

solución propuesta.

b. Diagrama de clases de la solución planteada. Adicionalmente,

describa en alto nivel la lógica de las tareas que usted definió para

aquellos subprogramas cuya solución no sea trivial.

c. Algoritmos implementados debidamente intra-documentados.

d. Problemas de desarrollo que afrontó.

e. Evolución de la solución y consideraciones para tener en

cuenta en la implementación.

Esquema del desafio

1. Clase Estación:

- Cada estación puede ser representada como un objeto de la clase `Estación`.

- La clase `Estacion` puede tener atributos como un nombre único para identificar la estación

-eliminar estación

-agregar estación

-ver si hay conexiones con esa estacion

2. Clase Línea

- clase `Linea` para representar cada línea de metro en tu red.

- Cada objeto `Linea` puede contener información sobre las estaciones que pertenecen a esa línea.

-agregar línea

-eliminar línea (hipotéticamente hablando”

- También puedes incluir información adicional como el color de la línea (no se si sea necesario hay que preguntar), el tiempo de recorrido estimado entre estaciones.

3. clase redMetro:

- clase `redmaMetro` que actúe como el controlador principal del sistema de metro.

-básicamente que muestre todo en pantalla

- líneas

- estaciones que hacen parte de una línea en especifico

- Esta clase puede contener métodos para la planificación de rutas, el cálculo de tiempos de viaje(seria lo mismo que en la clase línea).

Definir como se guardaran los datos y ver si podemos crear una red metro aleatoria como de principio o una red básica y que el usuario la personalice

Bueno, una de las condiciones que se nos exigen en el desafio es que no podemos usar el STL (Standard Template Library) para este desafio, por lo cual, una de las soluciones para, aparte mejorar lo que sería, hablando en términos de eficiencia, se nos ocurrió como crear 2 clases, que “simulen” lo que hacen los vectores y las listas de la STL.

Las clases para el vector y la lista se crean para como hacer una versión mas personalizada y que hagan una especie de simulación de su funcionamiento, con esto, nos brindan un mayor control y flexibilidad.

Imagina que estás construyendo un juego de bloques Lego desde cero en   
lugar de usar uno prehecho. Al hacer esto, podemos diseñar los bloques Lego para que se ajusten exactamente a lo que necesitamos para nuestro proyecto, como agregar más características o cambiar cómo se conectan entre sí. En términos más simples, es como crear tus propias herramientas en lugar de usar las que ya están disponibles, lo que te permite adaptarlas a tus necesidades específicas.

Estas clases tiene las siguientes funciones:

Clase para el Vector (MiVector):

* Permite almacenar una secuencia de elementos del mismo tipo de manera dinámica.
* Proporciona funciones para agregar elementos al final de la secuencia
* Controla la capacidad y el tamaño del vector
* Puede incluir otras funciones típicas de un vector, como eliminar elementos, acceder a elementos específicos (operator[]), entre otras

Clase para la Lista (MiLista):

* Representa una estructura de datos enlazada donde cada elemento está conectado al siguiente.
* Permite agregar elementos al final de la lista o al principio
* Ofrece la capacidad de eliminar elementos
* Proporciona funciones para recorrer la lista y acceder a elementos
* específicos.

En resumen, estas clases funcionan de manera similar a los vectores y listas estándar de C++, pero están diseñadas para adaptarse mejor a las necesidades específicas de un proyecto y para evitar el uso directo de las estructuras estándar del lenguaje.