INFORME PARCIAL 2

Daniel Esteban Pinzón Martínez

Nicolás Velásquez

Universidad De Antioquia

Informática II

Aníbal José Guerra Soler

Augusto Enrique Salazar Jiménez

02/05/2024

**Análisis**

A fin de realizar una simulación de una red de metro con las herramientas vistas hasta el momento en curso y con un enfoque de orientación a objetos se empezó con una breve visualización y comprensión de las partes esenciales que componen una red de metro y cómo interactúan entre ellas. Para ello se realizaron algunas preguntas de sonda cómo:

* ¿De qué elementos se compone o contiene esencialmente una red?
* Si existe, ¿Cuál es la relación de cardinalidad que existe entre estas partes?
* ¿Interactúan las partes de otras maneras entre sí?
* Para efectos de este desafío, ¿Qué atributos componen cada parte?
* De los subprogramas mencionados en el documento presentación, ¿A qué parte o clase pertenece cada uno de ellos?
* ¿Es necesario el reúso de alguna clase externa?

Adicionalmente se tienen en cuenta las consideraciones iniciales mencionadas en el documento presentación y en la clase conversatorio sobre el mismo, como el no uso de STL, que una línea que tenga un estacón de transferencia no puede ser eliminada, no hay bifurcaciones ni bucles, etc...

Por otro lado se considera el uso de apuntadores como argumentos de los métodos y memoria dinámica para el almacenamiento de los arreglos de las líneas y las estaciones, con el fin de cumplir con el criterio de eficiencia.

**Clases**

* **Diagrama de clases:**

Para la solución se prevé la codificación de tres clases: Red, Linea y Estación, además del reúso de la clase string para almacenar los nombres de cada uno de los objetos de cada clase:

Diagrama

Descripción generada automáticamente

* **Descripción de los subprogramas:**

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

* **Entradas:**

1. Nueva estación (\*string).
2. Estación inmediatamente anterior en la línea o extremos (-1 o 1), (\*string).
3. Valor del tiempo para llegar a la estación anterior(\*int).
4. Valor del tiempo para llegar a la estación siguiente(\*int).

* **Procedimiento:**

1. Contar el número de espacios del array que se encuentran ocupados.
2. Si no hay espacios disponibles en el arreglo, se crea un nuevo arreglo dinámico con n+2 espacios.
3. Asignar las estaciones en el mismo orden.
4. Asignar la nueva estación en su posición.
5. Modificar los tiempos de las estaciones adyacentes.
6. Completar el arreglo.
7. Liberar memoria del arreglo anterior si se creó uno nuevo y almacenar el nuevo apuntador al arreglo y a la última posición del mismo.

* **Salida:** Ninguna (Se modifica el apuntador o el arreglo del atributo directamente)

1. **Eliminar una estación de una línea. No se pueden eliminar estaciones de transferencia.**

* **Entradas:**

1. Estación a eliminar (\*string).
2. Tiempo entre las dos estaciones conexas a la que se va a eliminar. (\*int)

* **Procedimiento:**

1. Recorrer la línea en busca de la estación.
2. En la posición de la estación a eliminar se asigna la siguiente y se modifica el tiempo de las respectivas estaciones conexas.
3. Completar el arreglo asignando a la estación anterior la siguiente.
4. Se ingresa NULL en la última posición.

* **Salida:** Ninguna (Se modifica el apuntador o el arreglo del atributo directamente)

1. **Saber cuántas líneas tiene una red Metro**

* **Entradas:** Ninguna
* **Procedimiento:**

1. Recorrer el arreglo dinámico de líneas.
2. Contar las líneas hasta que se encuentre NULL o se llegue a la posición final del arreglo.
3. Devolver el contador.

* **Salida:** Contador con la cantidad de líneas (int).

1. **Saber cuántas estaciones tiene una línea dada**

* **Entradas:**

1. Línea a la cual se le quiere contar las estaciones (\*string)

* **Procedimiento:**

1. Recorrer el arreglo dinámico de líneas en busca de la línea.
2. En la línea, recorrer el respectivo arreglo dinámico de estaciones.
3. Contar las estaciones hasta que se encuentre NULL o se llegue a la posición final del arreglo.
4. Devolver el contador.

* **Salida:** Contador con la cantidad de estaciones (int).

1. **Saber si una estación dada pertenece a una línea específica.**

* **Entradas:**

1. Estación que se quiere buscar (\*string)
2. Línea en la que se pretende hallar la estación (\*string).

* **Procedimiento:**

1. Recorrer el arreglo dinámico de líneas en busca de la línea.
2. En la línea, recorrer el respectivo arreglo dinámico de estaciones en busca de la estación.

* **Salida:** Se devuelve la pertenencia o no de la estación a la línea (bool).

1. **Agregar una línea a la red Metro**.

* **Entradas:**

1. Nombre de la nueva línea (\*string).
2. Estación de transferencia (\*string).

* **Procedimiento:**

1. Verificar si la estación existe en la red y se le asigna como transferencia.
2. Contar el número de espacios del array de líneas que se encuentran ocupados.
3. Si no hay espacios disponibles en el arreglo, se crea un nuevo arreglo dinámico con n+1 espacios.
4. En la última posición del arreglo se crea la nueva línea con dos espacios, el primero contendrá la estación de trasferencia.
5. Liberar memoria del arreglo anterior si se creó uno nuevo y almacenar el nuevo apuntador al arreglo y a la última posición del mismo.

* **Salida:** Ninguna (Se modifica el apuntador o el arreglo del atributo directamente)

1. **Eliminar una línea de la red Metro**.

* **Entradas:**

1. Nombre de la línea a eliminar (\*string).

* **Procedimiento:**

1. Verificar si la línea existe en la red.
2. Recorrer la línea en busca de estaciones de transferencia.
3. Si no existen estaciones de transferencia en la línea, se libera la memoria del respectivo arreglo de estaciones.
4. Se asignan las demás líneas una posición atrás.
5. Se asigna NULL en la última posición.

* **Salida:** Ninguna (Se modifica el apuntador o el arreglo del atributo directamente)

1. **Saber cuántas estaciones tiene una red Metro**

* **Entradas:** Ninguna
* **Procedimiento:**

1. Recorrer cada línea de red contando las estaciones.
2. Si se encuentra una de transferencia se almacena su nombre original y se busca en las demás líneas, por cada vez que se encuentre se resta 1.

* **Salida:** Cantidad de estaciones (int)
* **Subprograma “Cálculo del tiempo de llegada”**

Se considera colocar este subprograma como método de la clase Red. Su descripción a continuación:

* **Entradas:**

1. Estación de salida (\*string)
2. Estación de destino (\*string)

* **Procedimiento:**

1. Recorrer cada línea de red en busca de la estación de salida.
2. Si se encuentra, en la misma línea se busca la estación de destino.
3. Si se encuentra, se toma como posición inicial la estación más a la izquierda.
4. Se suman todos los valores del tiempo de llegada de todas las estaciones hasta llegar a la estación final.

* **Salida:** Tiempo de llegada (int)

Es importante no olvidar que además de estos subprogramas (métodos), existen los .get, los .set, los constructores, destructores, y el menú de interacción.

**Detección de Problemas en el desarrollo**

Al iterar sobre el arreglo con los punteros no podíamos identificar si el espacio de la estación estaba vacío o si había una estación, ya que cuando apuntábamos a nullptr no funcionaba la comparación con nuestra clase creada, es por esta razón que se opto por crear un atributo llamado Vacio, el cual nos ayuda a identificar cuando el espacio esta vacio o cuando esta ocupado.