Implementacion Proyecto

Macias Hernandez Gerardo Antonio

11 de junio de 2023

1. Descripcion de la aplicacion

Este proyecto tiene como proposito el desarrollo de un programa interactivo en el que se toman decisiones sobre la forma de establecer comunicaciones telefonicas, de acuerdo a la informacion de una empresa telefonica que posee una red de n estaciones de conmutacion, conectadas por m enlaces de comunicacion de alta velocidad. El telefono de cada cliente esta conectado directamente con una estacion correspondiente a su area de conexion.
Tambien necesitamos decidir que grafica utlizaremos para nuestra red, en este caso optaremos por una grafica no dirigida. En cuanto a la estructura de datos que utilizaremos sera con una lista dinamica basada en arreglos, en la que implementamos la interfaz List y utilizamos un arreglo para almacenar los elementos de la lista.

2. Metodos y clases que implementamos

Para completar este proyecto, se tuvieron que crear varias clases conforme las intrucciones, las clases que gregamos son **Estacion**, **Cliente**, **RedTelefonica**, **ImplementationNonDirectedGraph** y **Prueba**.

2.1. Class Cliente

Esta clase la creamos porque requerimos guardar la informacion de cada uno de los clientes que hay por seccion y para ello ocupamos el nombre del cliente y telefono del cliente, tambien necesitaremos un tercer parametro para almacenar el codigo de la estacion en la que se encuentra el cliente.

2.2. Class Estacion

La clase estacion la ocupamos para guardar el nombre de la estacion y su codigo(id)

2.3. Class ImplementationNonDirectedGraph

Aqui es donde viene lo interesante, hacemos uso de una grafica no dirigida para modelar nuestra red. Haciendo uso de de la clase NonDirectedGraph en la carpeta de de estructuras/graficas, la implemetamos en nuestra clase ImplementattionNonDirectedGraph, declaramos nuestras varibales Lis<E>estaciones, boolean matrizDeAdyacencia e int size() y teniendo asi los metodos:

- **int size()** el cual nos devuelve el tamano de la grafica.
- **E vertex(int i)** el cual recibe un indice y devuelve el vertice que esta en dicha posicion.
- **Iterable<E>vertexSet()** el cual itera sobre un Array que almacenara los vertices y por medio de un bucle for-each sobre una la lista de clientes se va agregando cada vertice al Array.
- **int edgeNum()** nos devuelve el numero de aristas de la grafica.
- **boolean getEdge(int i, int j** devuelve true si existe la arista dada por los vertices i y j.
- **void setEdge(int i, int j)** agrega la arista dada por los indices i y j.

- **Iterable <E>edgeSet(int i)** devuelve un iterable de aristas que se encuentran almacenados en la grafica
- **Iterable <E>iterator()** para nuestro iterator necesitamos implementar una clase **class vertexIterator implements Iterator <E>** donde incluiremos nuestros metodos hasNext() y next().

Claro tambien tenemos nuetro metodo constructor ImplementationNonDirectedGraph(E[] estaciones) que recibe como parametro un arreglo e inicializa el tamano de las de las estaciones, el tamano de nuestra matriz, se intancia un ArrayList y se se realiza un bucle for-each sobre el arreglo estaciones, en acada iteracion se agrega un lemento a la lista estaciones.

2.4. Class RedTelefonica

En este metodo relizaremos todo respecto a las llamadas y videollamadas, tenemos un metodo constructor RedTelefonica() que tiene el metodo cargaDatos(FILENAME), donde FILENAME = red.xml que es desde donde se cargara toda la informacion.

- **void cargaDatos(String datos)**.
  - En primer lugar inicializamos dos listas estaciones y clientes como ArrayList, aunque solo terminaremos usando clientes para despues.
  - Inicializamos otra variables como nombreCliente, telefono e id, e inicializamos nuestra variable grafo tambien.
  - Luego, se realiza la lectura del archivo de texto utilizando la clase ReaderWriter y se almacena cada línea en la lista redtext.
  - Luego se itera sobre la lista redtext utilizando un iterador. En cada iteracion se procesa la operacion 'step' para extraer informacion importante.
  - Si la linea contiene la etiqueta **"<Estacion** se extrae el nombre de la estacion y su codigo(id) y se crea un objeto **Estacion** con estos datos.
  - Si la linea contiene la etiqueta **<Cliente** se extrae el nombre del cliente, su numero de telefono y con el id ya extraido creamos un Objeto del tipo **Cliente** y lo agregamos a la lista clientes.
  - Si la linea contiene la etiqueta **<Enlace**, se extraen los identidicadores de las estaciones involucradas en el enlace y utilizamos el metodo **'setEdge(x,y)'** de la clase ImplementationNonDirectedGraph para establecer una conexion entre las estaciones.
- **int[] BFSwithMiDistance(int start)**. Este metodo implemente lo que es el algoritmo BFS para calcular las distancias minimas desde un nodo de inicio ahsta todos los demas nodos del grafo utilizando un arreglo **distances** y una lista **'ls'** para el recorrido.
  - El metodo recibe un parametro **start** que es nuestro nodo de inicio.
  - creamos el arreglo **'distances'** qu tendra el tamanos de las estaciones para almacenar las distancias minimas desde el nodo de inicio hasta el resto. De inicio establecemos las distancias en general con MAX_VALUE para indicar que son desconocidas.
  - La distancia desde el nodo de inicio a sí mismo se establece en 0.
  - Se crea una lista **'ls'** para ir almacenando todos los nodos del recorrido.
  - Iniciamos un bucle while que tiene como condicion que ls no sea vacia y mientras esto ocurre inicializamos una variable nodo que guarda el primer elemento que removemos de ls.
  - Luego procedemos a recorrer todos los nodos adyacentes. Si un nodo adyacente aún no ha sido visitado (su distancia es Integer.MAX_VALUE), se actualiza su distancia sumando 1 a la distancia del nodo actual y se agrega a la lista ls.
  - finalmente se devuelde el arreglo **'distances'** que contiene las distancias minimas.
- **int caminoMasCorto(int i, int j)**. Calcula la longitud del camino mas corto entre los nodos i y j.

- Primero checamos si i = j, en ese caso la distancia seria 0, lo cual indica que son el mismo nodo.
- En el caso de que i y j no sean iguales llamamos al metodo BFSWithMiDistance() para calcular las distancias minimas desde el nodo i.
- Luego verificamos si la distancia desde i a j es diferente de MAX_Value y si es distinto significa que hay un camino entre i y j y nos devuelve la distancia que hay entre ellos.

- **Estacion find(String numero)**. este método busca una estación en la lista estaciones utilizando un número de teléfono asociado a un cliente. Si se encuentra una coincidencia válida, se devuelve la estación correspondiente. Si no se encuentra ninguna coincidencia válida, se devuelve null.
- **String existeCamino(String num1, String num2)**. Este metodo lo que hace es que dados dos numeros checa si son validos utilizando find(), si cualquiera de los dos no es valido regresa que el numero no es valido, de lo contrario si existen ambos numeros lo que hace es calcular la distancia entre estos.
- **String llamada(String num1, String num2)**. Lo que hace este metodo es que dados numeros checa con existeCamino() si son validos los numeros, luego si lo son procedemos a comprobar si la distancia que nos devuelve exiteCamino es mayor igual que cero o menor igual que 6 que es la distancia maxima para tener una llamada decente.
- **String videoLlamada(String num1, String num2)**. Lo que hace este metodo es que dados numeros checa con existeCamino() si son validos los numeros, luego si lo son procedemos a comprobar si la distancia que nos devuelve exiteCamino es mayor igual que cero o menor igual que 6 que es la distancia maxima para tener una videollamda decente.
- **List¡Cliente¿getClientes()**. Regresa una lista de todos los clientes.
- **List¡Estacion¿getEstaciones()**. Devuelve una lista de todas las estaciones.

2.5. Class Prueba

En clase prueba implementamos metodos que ya hemos usado en las practicas anteriores, el unico metodo que agremos es print(String msg, Iterable it) que lo usamos para la opcion 3 de nuestro menu y poder imprimir nuestro directorio.
Implementamos un swicth-case para nuestra interfaz de usuario, 1 para realizar una llamada normal, 2 para realizar una videollamda y 3 para poder ver el directorio. en 1 y 2 mandamos a llamar a los metodos llamada y videLlamada de las clase RedTelefonica.