Estructuras de Datos

Proyecto Final: Comunicaciones télefonicas

Juan Luis Peña Mata

1 Introducción

En este archivo se explicara el funcionamiento del programa, además de explicar a groso modo el propósito de las clases usadas. Para el funcionamiento deseado del programa se usaron tres clases auxiliares:

- Cliente, que contiene la información de un cliente en particular.
- Estación, que contiene la información de los clientes que pertenecen a una estacion en particular.
- Red, que contiene referencias a todas las estaciones de la red.

En estas se usaron diferentes estructuras de datos para el manejo de red, estas son:

- Gráficas no dirigidas.
- Listas basadas en arreglos.
- Tablas hash.
- Colas basadas en arreglos

2 Implementación de gráficas no ligadas

Esta estructura de datos tiene dos datos, *int size* que es la referencia al número de vértices que hay en la gráfica, *int edgeSize* que es la referencia al número de aristas que hay en la gráfica, además usamos una lista basada en arreglos para guardar las referencias a los vértices, este lista usa una clase interna llamada $Vertice$.

- $Vertice$
  Esta clase tiene los siguientes datos, *E contenido* que hace referencia al elemento que hay dentro del vértice, *int etiqueta* que hace referencia a una etiqueta que se le da al vértice cuando este se crea, esta es la posición del vértice en la lista de vértices de la gráfica, $ArrayList < Integer > adyacencias$ es una lista con las etiquetas de los vértices que son adyacentes al vértice, *int anterior* que es la etiqueta del vértice anterior a este en una trayectoria delimitada por el método BFS, *int distancia* que es la referencia a la distancia que hay entre el vértice que empieza una trayectoria y este vértice, las trayectorias de delimitan usando el método BFS.

Esta clase implementa los métodos de la interfaz $NonDirectedGraph$ de la siguiente forma.

- *vertex*(*int index*)
  Este método revisa que el índice *index* sea valido, después saca el vértice con ese índice de la lista de vértices y extrae su contenido con el método *getContenido*()
- *vertexSet*()
  Este método recorre la lista de vértices y los añade el contenido de cada vértice a una lista temporal para después devolverla.
- *getEdge*(*int i*, *int j*)
  Este método revisa que ambos índices sean validos, después le asigna la lista de adyacencias del vértice i a una lista auxiliar para recorrerla y comparar cada adyacencia con el índice j, si encuentra uno igual devuelve $true$, en caso contrario devuelve $false$.
- *setEdge*(*int i*, *int j*)
  Este método revisa que ambos índices sean validos, después agrega el índice j a la lista de adyacencias del vértice i y agrega el índice i a la lista de adyacencias del vértice j, además aumenta en una unidad el número de aristas que hay en la gráfica.
- *edgeSet*(*int i*)
  Este método revisa que el índice i sea valido, devuelve la lista de adyacencias del vértice con índice i.

Además se añaden los siguientes métodos:

- $BFS(int\ i)$
  Este método implementa el recorrido BFS para gráficas, este recorre la gráfica desde el vértice con índice i y actualiza las referencias *int anterior* y *int distancia*.

- $recoverPath(int\ i, int\ j)$
  Este método devuelve la trayectoria mas corta entre el vértice i y el vértice j, esto usando la referencia de cada vértice llamada *anterior* desde el vértice j hasta llegar a el vértice i.

- $havePath(int\ i, int\ j)$
  Este método devuelve *true* si hay una trayectoria entre el vértice i y el vértice j, en caso contrario devuelve $false$.

- $pathLength(int\ i, int\ j)$
  Este método devuelve la longitud de la trayectoria que hay entre los vértices i y j, si no hay una trayectoria devuelve -1.

3 Clases auxiliares

3.1 Cliente

La clase contiene contiene tres datos, *nombre* que es la referencia al nombre del cliente, *numero* que es la referencia al número telefónico del cliente y *estacion* que es la referencia a la estación del cliente, todos estos son cadenas. Esta clase tiene los siguientes métodos:

- $hashCode()$

  Este método invoca el método privado $hashCode(String\ cadena)$ con el numero de telefónico del cliente, y ya que el número telefónico es de la forma $55 - xxx - yyyyyyy$ tomamos la sub cadena $yyyyyyy$ y sumamos los dígitos, el resultado lo devolvemos como el hash code del elemento.

- $equals(Object\ obj)$ Este método compara dos objeto cliente, si son del mismo tipo, compara el numero telefónico, si estos son iguales devuelve *true*, en cualquier otro caso devuelve $false$.

3.2 Estación

Esta clase contiene dos datos y una estructura de datos, *String nombreEstacion* que es la referencia al nombre de la estación, *String codigoArea* que es la referencia al código de área de la estación, y el conjunto de $HashMap < Cliente > clientes$ y $Cliente[\ ]\ hash$ que representan un hash de clientes. Esta clase tiene los siguientes métodos:

- $agregarCliente(Cliente\ cliente)$
  Este método agrega en *Cliente cliente* al hash de clientes.

- $Iterable < Cliente > getClientes()$
  Este método devuelve el hash de clientes.

- $find(Cliente\ cliente)$
  Este método devuelve el índice donde se encuentra el *cliente* dado, si este no se encuentra en la estación devuelve -1.

3.3 Red

Esta clase contiene dos datos y una estructura de datos, *String numEstaciones* que es la referencia al numero de estaciones que hay en la red, *String numEnlaces* que es la referencia al numero de enlaces que hay entre las estaciones, y $LinkedListNonDirectedGraph < Estacion > grafica$ que es la referencia a una gráfica de *estaciones*. El contructor de la red inicializa la gráfica y usa el método $load(String\ fileName)$.

Esta clase tiene los siguientes metodos:

- $puedeLlamar(String\ saliente, String\ entrante)$
  Este método revisa que exista una conexión entre las estaciones de los números representados por las cadenas *saliente* y *entrante*, esto extrayendo el código de área que hay en cada número telefónico, usando el método *havePath* devuelve *true* si los números pueden llamarse, y $false$ en caso contrario.

- *hacerLlamada*(*String saliente*, *String entrante*)
  Este método realiza la llamada entre los números *saliente* y *entrante* y devuelve una cadena con la distancia entre las estaciones y las estaciones que se usaron para realizar la llamada.

- *valid*(*String aux*)
  Este método revisa que la cadena *aux* sea un número telefónico valido, después extrae la estación a la que el número pertenece y busca si el numero pertenece a la estaciona, si este número esta en la red devuelve *true*, en caso contrario devuelve $false$.

- *distancia*(*String saliente*, *String entrante*)
  Este método extrae las estaciones a las que los números pertenecen y verifica que estas estaciones estén conectadas entre si, si estas no lo están devuelve -1, en caso de que si lo estén devuelve la distancia entre las dos estaciones.

- *hacerVideoLlamada*(*String saliente*, *String entrante*)
  Este método realiza la vídeo llamada entre los números *saliente* y *entrante* y devuelve una cadena con la distancia entre las estaciones y las estaciones que se usaron para realizar la vídeo llamada.

- *getAll*()
  Este método crea una lista donde se estarán todos los clientes de la red.

- *load*(*String fileName*)
  Este método carga los datos que hay en el archivo $fileName$ a la red telefónica.

4 Prueba

En la clase prueba tenemos los siguientes métodos auxiliares:

- *getInt*

- *getLine*

- *print*

- *llamar*(*Red red*, *String saliente*, *String entrante*)
  Este método verifica con el método *puedeLlamar* que los números *saliente* y *entrante* puedan llamarse dentro de la red telefónica *red*, si se puede realizar la llamada devuelve la cadena resultante del método *hacerLlamada*, en caso contrario devuelve "La llamada no pudo realizarse pues no existe conexión directa entre ambos números."

En Esta clase se encuentra nuestro método main, cuando lo ejecutamos tenemos tres funciones en nuestro menú:

- Realizar llamada telefónica.
  En esta opción se usa el método auxiliar *getLine* para obtener dos números telefónicos, después con el método *valid* se verifica que ambos números son validos, si estos no son validos se le hará saber al usuario y no se realizara la llamada, en caso de que si sean validos se usara el método auxiliar *llamar* con los números obtenidos y con la red telefónica.

- Realizar vídeo llamada. Como en la opción anterior, se usa el método auxiliar *getLine* para obtener dos números telefónicos, después con el método *valid* se verifica que ambos números son validos, después se le asignara valor a *videoLlamada* con el método *distancia* entre los dos números obtenidos, ahora, si *videoLlamada* es igual a -1 imprimiremos "La vídeo llamada no pudo realizarse, se intentara una llamada simple.", y se intentara hacer una llamada con el método auxiliar *llamar*, si *videoLlamada* es mayor a 6 se imprimirá "La vídeo llamada no pudo realizarse, la distancia " + *videollamada* +" esta fuera del rango de vídeo llamadas.", y se intentara hacer una llamada con el método auxiliar *llamada*, en cualquier otro caso se usara el método *hacerVideoLlamada* obtener una cadena que imprimirá el resultado de la vídeo llamada.

- Imprimir directorio de números.
  Esta opción usa el método *getAll*() para imprimir todos los clientes dentro de la red telefónica.