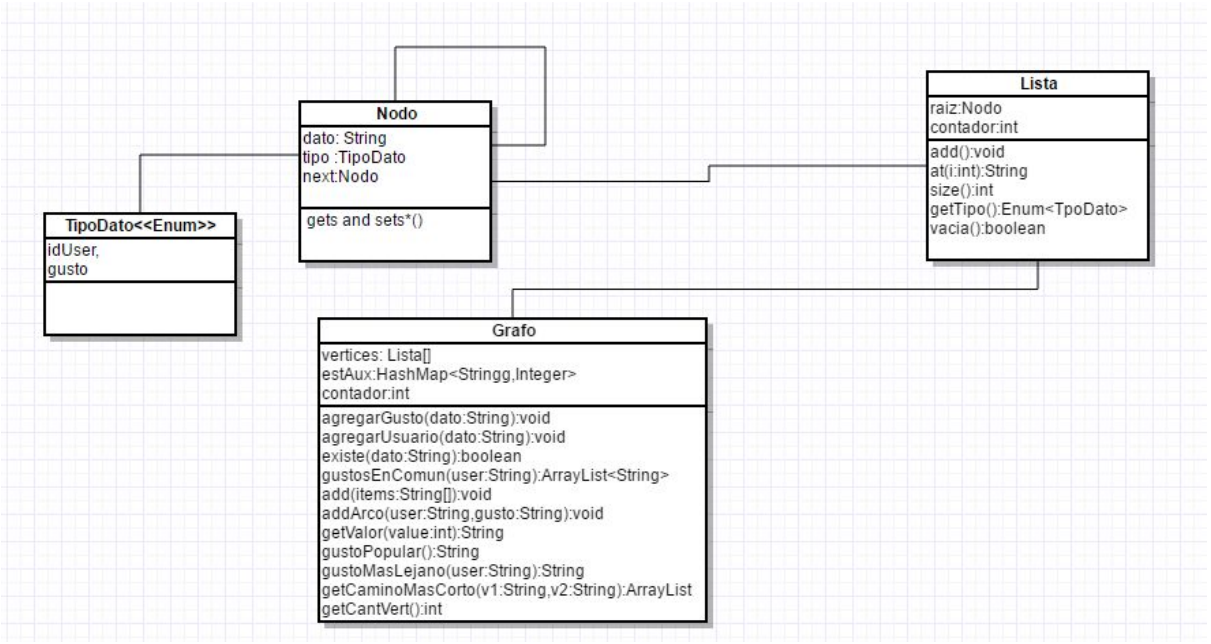
Informe Trabajo Práctico Especial

Programación 3 - TUDAI 2017

Alumnos: Pedro di Fonzo.
Lucas Pagadizabal.

Introducción

En el siguiente informe desarrollaremos las soluciones planteadas que creímos en el momento las mejores para abarcar el enunciado dado desde la cátedra.
Utilizamos estructuras y algoritmos vistos en la teoría, y tomamos diferentes ejercicios ya establecidos que desarrollamos en la práctica y los adaptamos para cubrir los requerimientos necesarios que plantea el enunciado



(breve descripción del diseño en diagrama)

Como decía el enunciado, utilizamos la estructura de un grafo. Para guardar los vértices del mismo utilizamos un arreglo de integer, donde cada posición de este arreglo tiene un puntero a una Lista Vinculada.
Decidimos guardar en el mismo arreglo las listas vinculadas ya sea para los gustos y/o para los usuarios. Para diferenciar lo que guarda cada lista utilizamos un enum donde detallamos qué tipo de información guarda la Lista, si la lista guardas gustos significa que el vértice es un usuario, en caso contrario si la lista guarda usuarios significa que el vértice es un gusto.
En el enunciado explicaba que el ID de un usuario era un String al igual que un gusto, esto nos generaba problema porque teníamos desarrollados los nodos con los datos de tipo int, pero lo pudimos resolver pasando el tipo de dato del nodo a String, entonces guarda ya sea el ID o el gusto. Al llevar a cabo este cambio nos surge otro problema, que anteriormente usábamos el dato del nodo como la posición del arreglo de vértices en la clase Grafo y como ahora son String no nos servía. A esta problemática la resolvimos utilizando una estructura auxiliar, un HashMap, donde la clave es el String (dato del Nodo) y el valor es la posición del arreglo de vértices de la clase Grafo.

Interfaces del Grafo

A continuación explicaremos algunos métodos que tenia que cumplir el grafo para utilizarlos como interfaz.

- agregarUsuario(String user)
    - En este método llevamos a cabo la inserción de un nuevo usuario en la estructura del grafo. Para esto primero chequeamos que la estructura de vértices no está completa, si esto sucede llamamos a otro metodo (agrandarEstructura()) que agranda la estructura dejando lugares libre para nuevas inserciones. En caso de que todavía tenga espacio chequeamos que el usuario a ingresar

no esté ya registrado en la estructura. Luego de estos dos chequeos previos insertamos en el arreglo de vértices una nueva Lista donde se van a guardar datos de los gustos del nuevo usuarios, y en la estructura auxiliar(HashMap) el ID como clave, y como valor la posición del arreglo donde fue insertada la Lista.
    - En el método agregarGusto(String user) se llevan a cabo los mismos procedimientos, nada más que la Lista va a guardar datos de Usuarios

- existe(String dato)
    - En este método nos devuelve si existe un usuario o un gusto en el grafo. Al tener un HashMap con todos los valores que se han insertado en el Grafo nos facilitó la búsqueda, ya que solo hacemos un contains sobre las key del HashMap

Servicios

- Dado un usuario, mostrarle las personas que tienen más de un gusto en común con él.

En este método buscamos los usuarios que poseen más de 1 gusto en común con el usuario dado.
Primero obtenemos la posición del arreglo en el cual está la lista de gustos del usuario dado. A estos los agregamos en un ArrayList, y recorremos todas las Listas de gustos de cada usuario, y vamos verificando si el usuario cumple con la condición de que posee más de un gusto que el usuario dado

- Mostrar qué gusto le gusta a más cantidad de gente.

En este método recorremos las Listas de los vértices de gustos, y vamos guardando el size() de cada una, ya que cuanto más nodos posea la Lista significa que más usuarios están conectados con ese

gusto. Al finalizar de recorrer devolvemos el gusto que posee el mayor size()

- Dada una persona, mostrar aquella que tenga gustos más lejanos a él.

En este método queremos demostrar que dado un usuario devolver el usuario que posea menos gustos en común con el dado.
Para llevar a cabo este servicio implementamos otro método, getCaminoCorto(String v1,String v2), que nos devuelve el camino más corto entre 2 vértices. Lo ejecutamos desde el vértice dado para con todos los vértices de tipo usuario que posee el grafo. Luego de obtener cada distancia hacia cada vértice de usuarios, lo único que hacemos es tomar la más lejana.

¿Se podrá realizar el mapeo de alguna manera más eficiente utilizando técnicas de hashing?

Se podría implementar una técnica de hashing abierta, para la cual se tendría que implementar una función que pase de string a números los id de los usuarios y los gustos en la cual se guardaría la posición del mismo dentro del arreglo de vértices y el valor.
Al querer buscar un usuario o un gusto habria que hacer la traduccion al número y realizando la función de hashing encontraríamos rápidamente la posición del mismo y ya al obtener el valor de la posición del elemento buscado en el arreglo de vértices tendremos un acceso directo a el y su lista de adyacentes.

Conclusión

En este trabajo pudimos llevar a cabo una implementación de Grafo complejo, respectivamente para nosotros, con datos del mismo tipo pero diferente significado, relacionados y de gran tamaño, donde creemos que sucede todo el tiempo en situaciones reales en lo que es desarrollo de sistemas.Siendo así podemos decir que nos llevamos un buen conocimiento sobre cómo manejar estos tipos de datos, ya sea como guardarlos en una estructura y/o como brindar los servicios necesarios para que el sistema sea mayormente eficiente y cumpla con los objetivos.