Universidad de San Carlos de Guatemala

Laboratorio de Estructura de Datos

Sección A

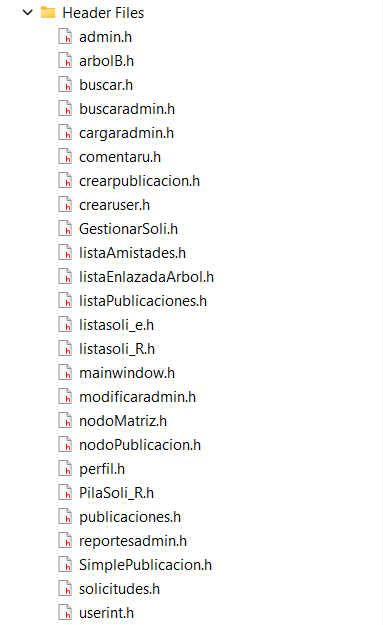
**MANUAL TECNICO**

Esdras Eliab Bautista Guerra

202100301

El programa se desarrollo con el lenguaje de programación C++ , con el GUI QT Creator, para poder hacerlo mas interactivo para el usuario.

* Clases utilizadas:
  + Clases .h:



* + Clases .cpp:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente con confianza media

* **Clase listaEnlazadaArbol.h:** Tenemos lo que es nuestro árbol binario de búsqueda (ABB o BST) en el cual vamos a guardar a todos los usuarios con el correo como llave, así mismo tenemos ciertos métodos los cuales ayudaran a que el árbol este equilibrado, en dado caso no lo este se realizan las rotaciones respectivas.



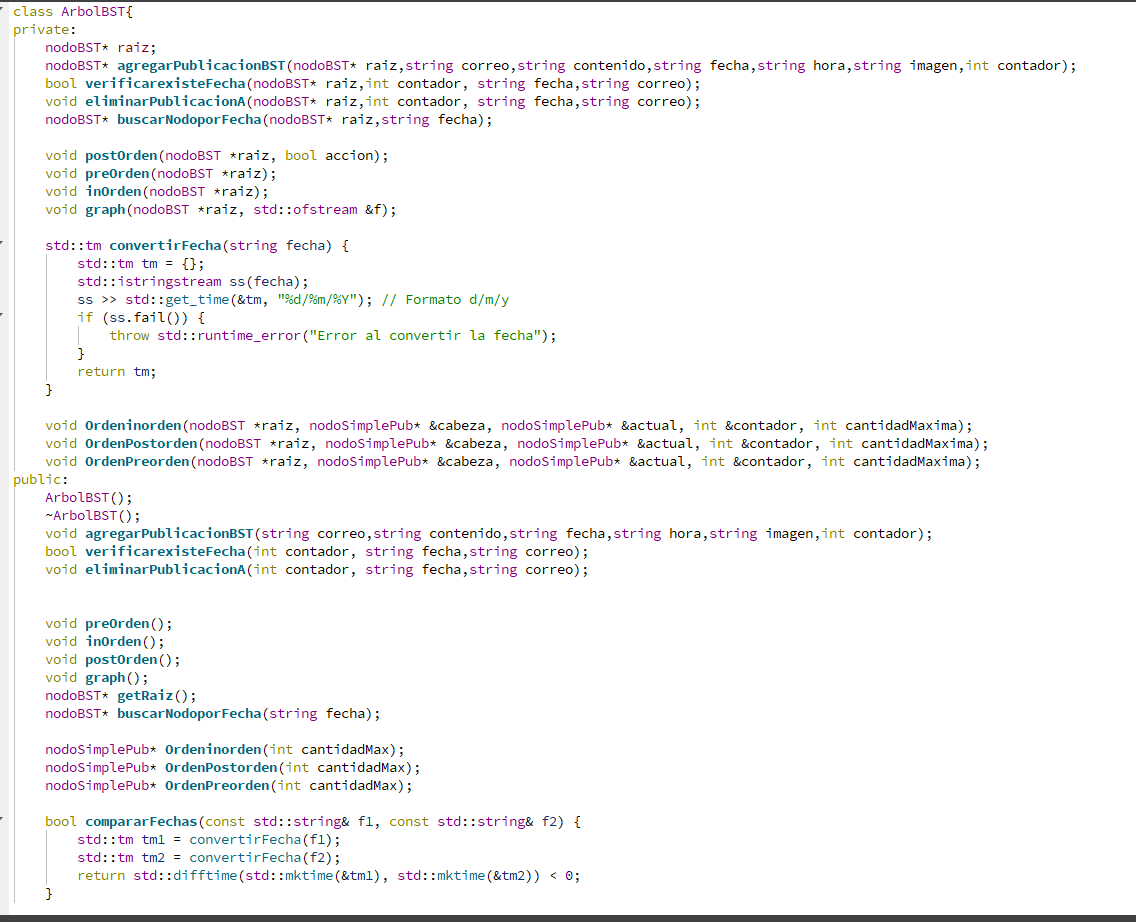
En nuestro nodoArbol, tenemos ciertos atributos importantes como lo son sus datos principales y así mismo tenemos una pila y listas para cada nodo, para así poder manejar de forma más eficiente la información para cada nodo.

Y así mismo en la clase .cpp procedemos a desarrollar los métodos, los getters, setter, constructores y destructores.

* **Clase listaPublicaciones.h:** Tenemos lo que es nuestro AVL, en el cual guardaremos todas las fechas de publicaciones, y cada nodo tendrá una lista enlazada que tendrá las publicaciones echas en esa fecha y cada nodo de las publicaciones tendrá un árbol B que es el árbol donde se almacenan los comentarios de cada publicación.

Texto

Descripción generada automáticamente con confianza media



Aquí en la parte del árbol implementamos un método para poder pasar las fechas a un formato con el cual podamos compararla con otras fechas, y así poder ir insertando como se deben los nodos en nuestro árbol.

* **Clase SimplePublicacion.h:** Se implementó una lista doblemente enlazada, para poder ver los comentarios siguientes y anteriores al momento que el usuario vea su feed, este nodo tiene sus atributos necesarios, pero hay uno que resalta y es el árbol B, cada publicación tiene su propio árbol B, el cual es de comentarios.



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

En nuestra lista implementamos los métodos necesarios para la inserción correcta de los nodos, el método verificarExistencia verifica que la publicación a agregar no este agregada ya anteriormente, esto solo para prevenir posibles problemas, el método agregarPublicacionL la cual agrega una publicación, un nodo, a la lista doblemente enlazada.

* Las clases pila, listasoli\_e , nodoMatriz y nodoPublicacion siguen siendo las mismas con los mismos métodos que la fase 1.
* **Clase arbolB:** Se implemento un árbol B, en el cual se almacenan los comentarios de cada publicación

****

**Imagen que contiene Texto

Descripción generada automáticamente**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente**

* Clase gestionarSoli: En esta clase se gestiona todos los procesos del usuario y unos del administrador:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Tenemos los métodos de enviar solicitud, aceptar solicitud, rechazar solicitud, cancelar solicitud, eliminar cuenta, ver amigos, agregar publicaciones de amigos, agregar publicaciones árbol, mostrar publicaciones árbol, ordenar publicaciones por, estas son los métodos que tiene esta clase para gestionar cada usuario y sus acciones o solicitudes, mientras que los métodos para gestión de administrador son aceptar solicitud D y agregar publicación admin.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Métodos para enviar solicitud:

Este método evalúa primero si ya son amigos, si ya son amigos retorna un mensaje, de lo contrario si no son amigos, procede a evaluar si ese usuario al que se le quiere mandar solicitud no le ha enviado ya solicitud o si el no ha mandado ya la solicitud, ya que no se puede enviar dos veces una solicitud a una persona y no se le pude enviar solicitud a un usuario que ya te ha mandado solicitud.

Y se cumple eso retorna un mensaje de advertencia, y si no se cumple, procede a agregar a la pila de la persona a quien se le manda solicitud el correo y a nuestra lista de solicitudes el correo de ese usuario.

* Interfaces utilizadas:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

Se utilizaron estas clases .ui las cuales son las ventanas de interfaz que el usuario visualizara, cada una con su respectivo nombre de lo que hace, la ventana principal es mainwindow.ui donde empezara todo el programa.

* Cada reporte fue realizado con graphviz y mostrado en un label tanto para el usuario como para el administrador, ejemplo de graphviz para mostrar el árbol de usuarios (Árbol ABB):

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Ya que es un árbol lo recorremos recursivamente, así en ese orden vamos a ir nombrando nodos y haciendo las conexiones, hacemos un recorrido PreOrden.

Y al finalizar generamos un documento .dot y una imagen .png.

* Relación Amistad

En la clase de relacionAmistad.h tenemos lo siguiente:

Texto

Descripción generada automáticamente

En esta parte tenemos la clase para el vértice del nodo, en donde contendrá de forma privada un string correo y dos apuntadores, uno de tipo vértice nodo(vnodo) el cual será “siguiente” y uno de destino de vértice destino (enodo).

De forma pública tendemos el constructor, y unos métodos los cuales nos ayudarán a graficar y a insertar vértice destino, y sus getter y setters.

Texto

Descripción generada automáticamente

También tenemos la clase enodo, el cual es el destino del nodo, como el vértice destino, en donde de forma privada tenemos dos cosas, el correo del destino y un apuntador de siguiente de tipo enodo.

Así mismo de forma pública tenemos el constructor y los getter y setters correspondientes.

Texto

Descripción generada automáticamente

También tenemos la clase listaAdyacencia, la cual como el mismo nombre lo dice es para crear la lista de adyacencia con respecto a las amistades que tenga cada usuario, en donde de forma privada tendremos únicamente la cabeza de tipo vnodo (vértice nodo) para poder empezar a recorrer la lista. Así mismo tenemos de forma publica lo que es el insert, la verificación de amistad, la eliminación del grafo, la creación de conexión y la graficación.

En la parte del .cpp tenemos:

Para la clase enodo: Inicializamos el constructor, así creamos los métodos de getter y setter

Texto

Descripción generada automáticamente con confianza media

Para la clase vnodo: tenemos la inicialización del constructor y de los getter y setter y así mismo del destructor.

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

También tenemos la implementación de los método, los cuales serian la inserción de un nodo destino en donde lo insertamos como si fuera una lista simplemente enlazada.

La graficacion de las aristas, en donde mediante un ciclo while recorremos toda nuestra lista de nodos y los vamos graficando, mientras lo recorremos, usando Graphviz.

Texto

Descripción generada automáticamente

Hacemos lo mismo para graficar los nodos destino.

Para la clase listaAdyacencia: inicializamos el constructor, implementamos los métodos del destructor, de la inserción (insert)

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

El método para crear conexión, nos sirve para claramente crear conexión entre un nodo y otro, hacer esa conexión mediante una arista(viéndolo desde un grafo).

El método crearGrafo prácticamente nos grafica mediante Graphviz la estructura de amistad de forma de un grafo, con sus vértices y nodos.

Texto

Descripción generada automáticamente

El método de crearGrafoLista es para crear el mismo grafo pero de una forma diferente, de una forma de una lista adyacente.

Texto

Descripción generada automáticamente

El método verificarAmistadGrafo es utilizado para ver si ya son amigos o no, en donde se recorre cada nodo vértice y si coinciden los correos retorna un true, de lo contrario retorna un false.