Universidad San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Estructura de Datos

MANUAL TECNICO

Victor Eduardo Franco Suret

202001767

21 de Septiembre de 2024

OBJETIVO DE ESTE MANUAL

El presente manual técnico tiene como finalidad describir aspectos técnicos y la forma en la que está estructurado, así como la interacción del usuario con este. Se muestra una fase 2 del proyecto general que es una aplicación de escritorio con una interfaz gráfica de usuario (GUI), lo que proporcionará una experiencia más intuitiva y accesible para los usuarios.

DIRIGIDO

El uso de este manual es exclusivo para programadores o diseñadores que desean saber más sobre el funcionamiento técnico de la aplicación. Para la comprensión de este manual se debe tener conocimientos previos sobre el lenguaje de programación C++.

CONOCIMIENTOS PREVIOS

Los conocimientos mínimos que deben tener las que utilizaran este manual son:

- Conocimientos sobre el Lenguaje C++

-Uso de Apuntadores, Ciclos , Listas dinámicas,arboles.

-Conocimientos Generales en el uso de las librerías Graphviz y Nlohmann(para archivos json)

ESPECIFICACIONES TECNICAS

Sistema Operativo: Windows 11

Lenguaje de Programación: c++.

IDE: Se utilizo Qt Creators

Tener instalado Graphviz y Nlomann.

Siendo esta la Fase 2 del proyecto, se dará énfasis a las nuevas funcionalidades, así como las sustituciones de la fase anterior.

Para mayor comodidad los archivos correspondientes a los arboles llevan el nombre de este.

Vadmin referencia a la ventana del administrado el cual tiene acceso total para adición, edición y eliminación de datos.

Vuser referencia a la ventana de usuarios, en este es donde se concentra la mayoría de código siendo que es la que ejecute mas cantidad de operaciones.

Para ingresar al usuario administrador se toman las mismas credenciales:
Usuario: admin@gmail.com

Password: EDD2S2024

ARBOL AVL

Propósito y Funcionalidad:

El árbol AVL se utiliza para gestionar los usuarios registrados en la red social. Su principal ventaja es mantener el balance del árbol para asegurar que todas las operaciones comunes (búsqueda, inserción y eliminación) se realicen de manera eficiente en tiempo O(log n). Esto es especialmente útil en aplicaciones con una gran cantidad de usuarios, ya que mantiene las operaciones de búsqueda y modificación rápidas.

Características Clave:

Auto-Balanceo: Cada vez que se inserta o elimina un nodo, el árbol AVL verifica el balanceo de cada nodo y, si es necesario, realiza rotaciones (simples o dobles) para mantener la altura de los subárboles dentro de una diferencia de uno.

Rotaciones: Existen cuatro tipos de rotaciones que el árbol puede realizar:

- Rotación Simple a la Izquierda
- Rotación Simple a la Derecha
- Rotación Doble a la Izquierda-Derecha
- Rotación Doble a la Derecha-Izquierda

Estas rotaciones permiten que el árbol se reestructure después de una inserción o eliminación, manteniendo un equilibrio óptimo.

Criterio de Ordenación: Los nodos se ordenan usando los correos electrónicos de los usuarios en orden alfabético. Esto permite búsquedas eficientes por correo, lo que es esencial para las funciones como iniciar sesión, buscar amigos o gestionar solicitudes.

Gestión de Amigos y Solicitudes: Las funciones que manejan las solicitudes de amistad y la verificación de si dos usuarios son amigos se apoyan en la eficiencia del AVL.

Dentro del árbol se pueden encontrar Operaciones CRUD (Crear, Leer, Actualizar, Eliminar): Todas estas operaciones sobre los usuarios son eficientes gracias a la estructura equilibrada del árbol AVL.

```cpp
ArbolAVL::ArbolAVL() : raiz(nullptr) {}

ArbolAVL::~ArbolAVL() {
    liberarMemoria(raiz);
}

void ArbolAVL::insertar(const Usuario& usuario) {
    raiz = insertarNodo(raiz, usuario);
}

void ArbolAVL::mostrarUsuarios() {
    if (!raiz) {
        std::cout << "El árbol está vacío.\n";
        return;
    }
    mostrarUsuariosInOrder(raiz);
}

void ArbolAVL::mostrarUsuariosInOrder(NodoAVL* nodo) {
    if (!nodo) return;

    // Recorrer el sub izquierdo
    mostrarUsuariosInOrder(nodo->izquierdo);

    std::cout << "Correo: " << nodo->usuario.correo.toStdString() << "\n";
    std::cout << "Nombres: " << nodo->usuario.nombres.toStdString() << "\n";
    std::cout << "Apellidos: " << nodo->usuario.apellidos.toStdString() << "\n";
    std::cout << "Fecha de Nacimiento: " << nodo->usuario.fechaNacimiento.toStdString() << "\n";
    std::cout << "----------------------------------------\n";

    // Recorrer el sub derecho
    mostrarUsuariosInOrder(nodo->derecho);
}


NodoAVL* ArbolAVL::insertarNodo(NodoAVL* nodo, const Usuario& usuario) {
    if (!nodo) {
        return new NodoAVL(usuario);
    }

    if (usuario.correo < nodo->usuario.correo) {
        nodo->izquierdo = insertarNodo(nodo->izquierdo, usuario);
    } else if (usuario.correo > nodo->usuario.correo) {
        nodo->derecho = insertarNodo(nodo->derecho, usuario);
    } else {
        return nodo; // Correo duplicado, no se inserta
    }

    nodo->altura = 1 + std::max(altura(nodo->izquierdo), altura(nodo->derecho));
    return balancear(nodo);
}
```

Árbol ABB

Propósito y Funcionalidad:

El árbol ABB se encarga de organizar las publicaciones de los usuarios por fecha, permitiendo un acceso cronológico y ordenado a los contenidos generados por los usuarios. Este árbol no tiene un auto-balanceo intrínseco como el AVL, lo cual puede ocasionar que se degrade a una lista enlazada en el peor de los casos si no se inserta de manera equilibrada.

Características Clave:

Ordenación por Fecha: Cada nodo del arbol ABB representa un día y contiene una lista doblemente enlazada de publicaciones realizadas en esa fecha. Esto permite que múltiples publicaciones del mismo día se agrupen en un solo nodo.

Listas Simples para Publicaciones: En cada nodo (fecha), se mantiene una lista enlazada de publicaciones ordenadas según la hora en la que se realizaron. Esta estructura facilita la gestión de publicaciones múltiples en una misma fecha.

Recorridos del Árbol: Se pueden realizar recorridos en orden, preorden o postorden para listar publicaciones de manera cronológica o en cualquier otro orden .

Filtrado por Fecha: A través de la interfaz, los usuarios pueden filtrar y ver publicaciones de una fecha específica, apoyándose en la estructura del árbol ABB.

Gestión de Contenidos: Las publicaciones se insertan de manera ordenada por fecha, facilitando la adición y visualización de contenido en la plataforma.

```cpp
ArbolABB::ArbolABB() : raiz(nullptr) {}

ArbolABB::~ArbolABB() {
    limpiar(raiz);
}

void ArbolABB::insertarPublicacion(const QString& correo, const QString& contenido, const QString& rutaImagen, const QDate& fecha) {
    raiz = insertarNodo(raiz, fecha);
    NodoABB* nodo = raiz;

    // Buscar el nodo con la fecha
    while (nodo->fecha != fecha) {
        if (fecha < nodo->fecha) {
            nodo = nodo->izquierda;
        } else {
            nodo = nodo->derecha;
        }
    }

    // Crear una nueva publi y agregarla a la lista del nodo
    Publicacion* nuevaPublicacion = new Publicacion(correo, contenido, rutaImagen);
    insertarEnLista(nodo, nuevaPublicacion);
}

NodoABB* ArbolABB::insertarNodo(NodoABB* nodo, const QDate& fecha) {
    if (nodo == nullptr) {
        return new NodoABB(fecha);
    }

    if (fecha < nodo->fecha) {
        nodo->izquierda = insertarNodo(nodo->izquierda, fecha);
    } else if (fecha > nodo->fecha) {
        nodo->derecha = insertarNodo(nodo->derecha, fecha);
    }
    // Si la fecha ya existe, no inserta un nuevo nodo solo a la lista

    return nodo;
}
```

Árbol B

Propósito y Funcionalidad:

El árbol B se utiliza para almacenar los comentarios en las publicaciones. Cada publicación tiene su propio árbol B que maneja los comentarios ordenados por fecha y hora.

Características y Funcionamiento:

Balanceo Automático: El árbol B es un árbol de búsqueda balanceado diseñado para manejar grandes volúmenes de datos con múltiples niveles, adecuado para almacenamiento secundario.

Múltiples Llaves por Nodo: Cada nodo del árbol B puede contener múltiples llaves y punteros a otros nodos, permitiendo que las operaciones de inserción, eliminación y búsqueda se realicen de manera eficiente incluso cuando los datos no caben en la memoria principal.

Ordenar Comentarios por Fecha y Hora: Los comentarios se ordenan cronológicamente dentro del árbol B de cada publicación, lo que facilita un acceso rápido y ordenado a los comentarios.

Almacenar Comentarios: Cada publicación puede tener múltiples comentarios, y estos se organizan y almacenan en su propio árbol B. Esto ayuda a mantener los comentarios ordenados y facilita la visualización y gestión de los mismos.

```cpp
ArbolB::~ArbolB() {
    limpiar(raiz);
}

void ArbolB::insertarComentario(const QString& usuario, const QString& texto, const QDateTime& fechaHora) {
    Comentario nuevoComentario(usuario, texto, fechaHora);

    // Si el arbol esta vacio crea la raiz
    if (!raiz) {
        raiz = new NodoB(true);
        raiz->comentarios.push_back(nuevoComentario);
    } else {
        // Si la raiz esta llena, se necesita dividir
        if (raiz->comentarios.size() == 4) {  //ajustar orden del arbol
            NodoB* nuevaRaiz = new NodoB(false);
            nuevaRaiz->hijos.push_back(raiz);
            dividirNodo(nuevaRaiz, 0, raiz);

            int i = (nuevaRaiz->comentarios[0].fechaHora < nuevoComentario.fechaHora) ? 1 : 0;
            insertarEnNodo(nuevaRaiz->hijos[i], nuevoComentario);

            raiz = nuevaRaiz;
        } else {
            insertarEnNodo(raiz, nuevoComentario);
        }
    }
}

void ArbolB::insertarEnNodo(NodoB* nodo, const Comentario& comentario) {
    int i = nodo->comentarios.size() - 1;

    // Si el nodo es hoja, inserta directamente en el nodo
    if (nodo->esHoja) {
        nodo->comentarios.push_back(comentario);
        std::sort(nodo->comentarios.begin(), nodo->comentarios.end(), [](const Comentario& a, const Comentario& b) {
            return a.fechaHora < b.fechaHora;
        });
    } else {
        // Encuentra la posición correcta para el hijo
        while (i >= 0 && comentario.fechaHora < nodo->comentarios[i].fechaHora) {
```

Otras funcionalidades que se agregaron fuero para la visualización y gráficos de los arboles.

Integración con Graphviz: Se añadieron funciones para generar gráficos visuales de los árboles y listas utilizando Graphviz, incluyendo la representación del Árbol AVL de usuarios y el ABB de publicaciones, también para visualizar el árbol b de comentarios para cada publicación.

Interfaz Gráfica Mejorada: Se mejoró la interfaz gráfica con Qt para presentar las publicaciones, comentarios y usuarios de manera más visual e interactiva, incluyendo el uso de QScrollArea, QGroupBox, QTableWidget, y QComboBox para una mejor experiencia de usuario.

Validaciones Adicionales: Se añadieron validaciones para verificar que los campos requeridos están llenos durante el registro y carga masiva.

Botones Interactivos en Publicaciones: Se añadieron botones interactivos para comentar, ver comentarios y ver el árbol de comentarios directamente en las publicaciones.

Visualización de Publicaciones y Comentarios en Scroll Areas: Las publicaciones ahora se muestran en un QScrollArea, permitiendo a los usuarios desplazarse y visualizar fácilmente el contenido.