FACULTADES DE INGENIERÍA EN SISTEMAS E INFORMATICA

TEMA:

Árbol binario de búsqueda

ASIGNATURA:

ESTRUCTURA DE DATOS

DOCENTE:

Ninoska Nataly Cohaila Bravo

ESTUDIANTES:

Gamero de la cruz angeles
García Quijada Álvaro
Pariona Soriano Diego

Huancayo - Perú, 2025

Introducción

Un **árbol binario de búsqueda (ABB)** es una estructura de datos jerárquica que organiza la información en nodos, cada uno con un valor y como máximo dos hijos: izquierdo y derecho; su característica esencial es que los valores del subárbol izquierdo son menores que el del nodo y los del subárbol derecho son mayores, lo que permite realizar operaciones de búsqueda, inserción y eliminación de manera eficiente en tiempo promedio (O(\log n)). Esta propiedad lo convierte en una herramienta fundamental en ciencias de la computación, ya que facilita la organización ordenada de datos y sirve como base para estructuras más avanzadas como los árboles AVL o rojo-negro. Su importancia radica en aplicaciones prácticas como bases de datos, sistemas de archivos, inteligencia artificial y representación de relaciones jerárquicas, entre ellas los **árboles genealógicos**, donde cada individuo puede ser un nodo identificado por atributos únicos (nombre, fecha de nacimiento, ID), permitiendo consultas rápidas de ancestros y descendientes y garantizando la integridad de la información familiar.

Capítulo 1 – Fase de Ideación

Descripción del Problema:

¿Cómo se puede aplicar un árbol binario de búsqueda para representar y consultar un árbol genealógico?

En un gran número de familias, y en especial la que tiene una historia extensa o acoge una gran diversidad de estructura familiar, es sencillo perder la noción de cómo se relacionan unas personas con otras. Con el correr del tiempo, perdido el hilo de la discusión, conviene dejar testimonio de quién descendía en cada caso o intentar examinar cómo se relacionaban unas ramas familiares con otras, lo cual resulta complicado de seguir.

Por ello surge la necesidad de una aplicación informática que nos permita edificar un árbol genealógico y poder, por tanto, agregar, eliminar y consultar la información de los miembros de la familia. La herramienta tiene que permitir respuestas rápidas y mostrar relaciones familiares con claridad.

Requerimientos del Sistema:

Requerimientos Funcionales

ID	Requerimiento	Descripción
RF1	Inserción de personas	Permitir añadir individuos con atributos (nombre, fecha de nacimiento, ID único).
RF2	Definición de relaciones	Registrar vínculos padre-hijo siguiendo la lógica del árbol binario de búsqueda.
RF3	Búsqueda de individuos	Consultar rápidamente un individuo por clave (ej. nombre o ID).
RF4	Visualización del árbol	Mostrar la estructura genealógica en forma jerárquica o gráfica.
RF5	Actualización de datos	Modificar información de un individuo sin romper la estructura del ABB.
RF6	Eliminación de nodos	Posibilidad de borrar individuos manteniendo la consistencia del árbol.
RF7	Consulta de ancestros	Mostrar padres, abuelos y generaciones anteriores.
RF8	Consulta de descendientes	Mostrar hijos, nietos y generaciones posteriores.
RF9	Búsqueda personalizada	Filtrar por nombre, rango de fechas o ID.

Requerimientos No Funcionales

ID	Requerimiento	Descripción
RNF1	Rendimiento	Operaciones de búsqueda, inserción y eliminación en tiempo logarítmico O(log n).
RNF2	Escalabilidad	Soportar árboles genealógicos grandes (múltiples generaciones).
RNF3	Usabilidad	Interfaz clara y sencilla para consultas y visualización.
RNF4	Portabilidad	Compatible con distintos sistemas operativos y entornos de desarrollo.
RNF5	Seguridad	Control de acceso y permisos para modificar o consultar el árbol.
RNF6	Integridad de datos	Validar que no existan relaciones contradictorias (ej. padres duplicados).
RNF7	Mantenibilidad	Código modular y fácil de actualizar o extender.
RNF8	Disponibilidad	El sistema debe estar accesible de forma confiable para los usuarios.

Respuestas De Las Preguntas Guía:

¿Qué información se debe almacenar en cada nodo del árbol?

En este caso, cada nodo representaría a una persona, integrante de la familia. Por lo que los datos que tendría que concentrar serían:

- ID único
- Nombre del integrante
- Fecha de nacimiento.
- Padres
- Hijos

¿Cómo insertar y eliminar miembros del árbol sin romper su estructura?

Inserción:

- Si el nuevo ID es menor al nodo actual, va a la izquierda.
- Si es mayor, va a la derecha.
- Recursivamente, hasta encontrar posición nula.

Eliminación:

- Caso 1: nodo hoja → se eliminan directamente.
- Caso 2: nodo con un hijo → este lo sustituye
- Caso 3: nodo con dos hijos → se sustituye por el menor del subárbol derecho (sucesor inorden).

¿Qué métodos permiten recorrer el árbol para visualizar la genealogía?

- Inorden (izq - raíz - der): miembros mostrados en orden creciente (por ID o nombre).
- Preorden (raíz - izq - der): útil para la reconstrucción del árbol.
- Postorden (izq - der - raíz): útil para eliminar el árbol.
- Nivel por nivel (BFS): para visualizar por generaciones.

¿Cómo determinar si un miembro pertenece a una rama específica?

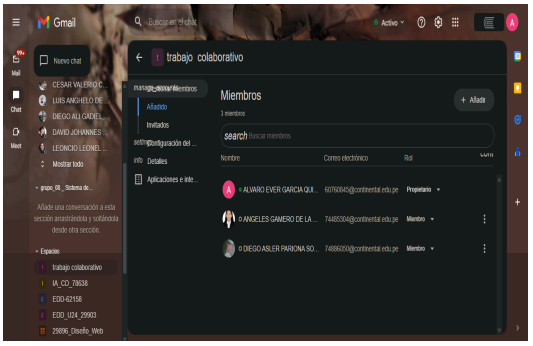
- Definir la rama como un subárbol.
- Buscar el nodo raíz de la rama, luego buscar recursivamente dentro de ese subárbol si el miembro existe.

¿Cómo balancear el árbol si se vuelve demasiado profundo?

El ABB puede volverse degenerado (como una lista). Para balancear:

- Convertirlo en un árbol que mantiene balance después de cada inserción/eliminación
- O recorrerlo inorden, almacenar en lista, y reconstruir un árbol balanceado desde la lista.

Herramientas colaborativas:

Gmail-chat https://chat.google.com/room/AAQAbht99yc?cls=7	

Github https://github.com/alvarocaldo/ABR_Arbol_Genealogico_grup08	

Capítulo 2 - Prototipo

Descripción de estructuras de datos y operación

Componentes del Árbol (Persona):

Datos identificativos:

- Campo numérico único (ID)
- Cadena de texto para el nombre completo
- Fecha de nacimiento en formato texto

Vínculos ascendentes:

- Referencia al nodo padre biológico
- Referencia al nodo madre biológica
  (Ambas referencias pueden ser nulas)

Estructura de ordenamiento:

- Nodo hijo menor (subárbol izquierdo)
- Nodo hijo mayor (subárbol derecho)

```
1  struct Persona {
2      int id;              // Identificador único
3      string nombre;       // Nombre del familiar
4      string fecha_nac;    // Fecha de nacimiento
5      Persona* padre;      // Puntero al padre (NULL si no existe)
6      Persona* madre;      // Puntero a la madre (NULL si no existe)
7      Persona* izq;        // Hijo izquierdo (para el BST)
8      Persona* der;        // Hijo derecho (para el BST)
9  };
```

Operación	Descripción
Agregar persona()	Crea un nuevo nodo con datos básicos.
Eliminar persona()	Inserta un nodo en el BST según su id y establece relaciones familiares.
Buscar()	Busca un nodo por id usando recursividad.

Mostrar descendientes()	Elimina un nodo manteniendo la estructura BST (3 casos: hoja, 1 hijo, 2 hijos).
MostrarAncestros()	Muestra padres, abuelos, etc. (recursivo).
Ver tabla de personas ()	Muestra hijos, nietos, etc. (simplificado).
Ver recorridos del arbol ()	Reconstruye el árbol para optimizar búsquedas (usando recorrido inorden).
Recorridos	inorden (), preorden (), postorden (), porNiveles ().

Algoritmos Principales:

- *Pseudocódigo para crear una árbol binario*

```
1   FUNCIÓN crearArbolBinario()
2       raíz ? NULL
3       RETORNAR raíz
4   FIN FUNCIÓN
5
6   FUNCIÓN crearNodo(id, nombre, fecha)
7       nuevoNodo ? Reservar memoria para Persona
8       nuevoNodo.id ? id
9       nuevoNodo.nombre ? nombre
10      nuevoNodo.fecha_nac ? fecha
11      nuevoNodo.padre ? NULL
12      nuevoNodo.madre ? NULL
13      nuevoNodo.izq ? NULL
14      nuevoNodo.der ? NULL
15      RETORNAR nuevoNodo
16  FIN FUNCIÓN
```

- *Pseudocódigo para realizar el recorrido de un árbol binario*

  - *Recorrido Preorden*

```
// PREORDEN: Raíz - Izquierda - Derecha
void preorden(Persona* raiz) {
    if (raiz == NULL) return;
    cout << "[" << raiz->id << "] " << raiz->nombre << " (" << raiz->fecha_nac << ")\n";
    preorden(raiz->izq);
    preorden(raiz->der);
}
```

  - *Recorrido Inorden*

```
// INORDEN: Izquierda - Raíz - Derecha
void inorden(Persona* raiz) {
    if (raiz == NULL) return;
    inorden(raiz->izq);
    cout << "[" << raiz->id << "] " << raiz->nombre << " (" << raiz->fecha_nac << ")\n";
    inorden(raiz->der);
}
```

  - *Recorrido Postorden*

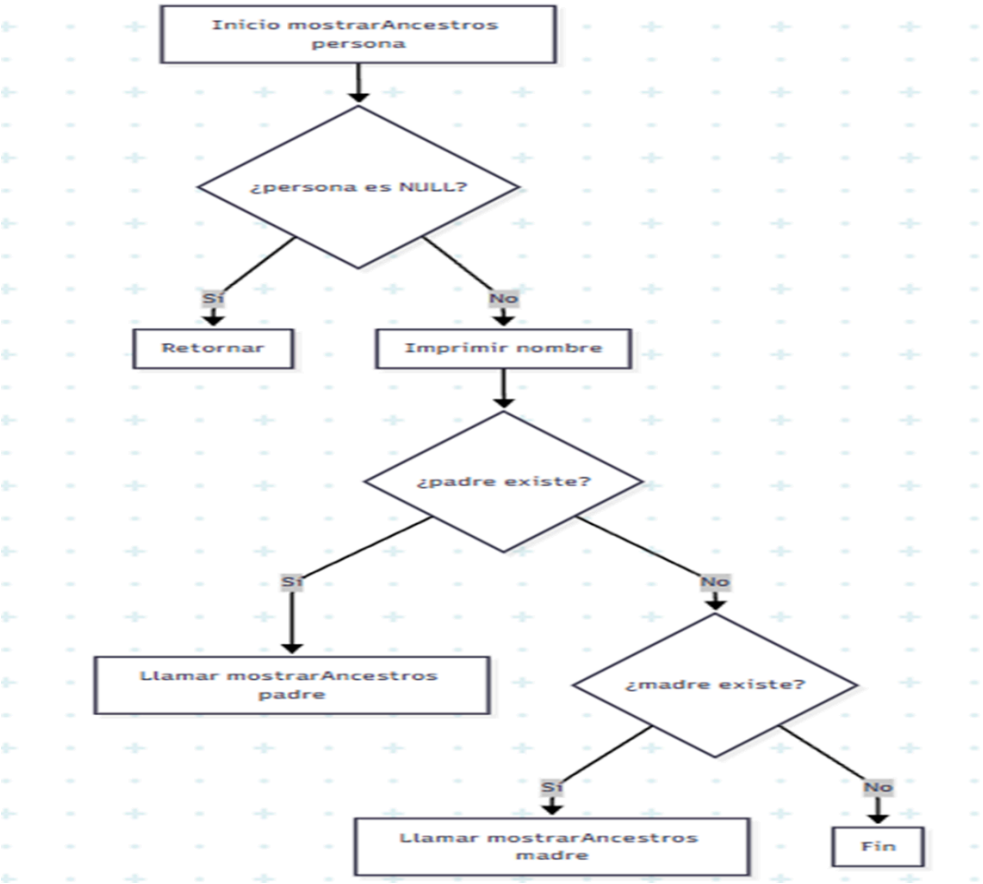
```
// POSTORDEN: Izquierda - Derecha - Raíz
void postorden(Persona* raiz) {
    if (raiz == NULL) return;
    postorden(raiz->izq);
    postorden(raiz->der);
    cout << "[" << raiz->id << "] " << raiz->nombre << " (" << raiz->fecha_nac << ")\n";
}
```

  - *Recorrido Niveles*

```cpp
// RECORRIDO POR NIVELES (BFS)
void porNiveles(Persona* raiz) {
    if (raiz == NULL) return;

    Cola c;
    inicializarCola(c);
    encolar(c, raiz);

    int nivelActual = 0;
    int nodosNivel = 1;
    int nodosContados = 0;

    while (!colaVacia(c)) {
        Persona* actual = frente(c);
        desencolar(c);

        cout << "[" << actual->id << "] " << actual->nombre << " (" << actual->fecha_nac << ")  ";
        nodosContados++;

        if (actual->izq != NULL) encolar(c, actual->izq);
        if (actual->der != NULL) encolar(c, actual->der);

        if (nodosContados == nodosNivel) {
            cout << "\n";
            nivelActual++;
            nodosNivel *= 2;
            nodosContados = 0;
        }
```

Diagramas de Flujo:

Diagrama Insertar Raíz y Búsqueda de ancestros:



Avance del código fuente

```
// MENÚ PRINCIPAL
// ================================================================================
void menu() {
    Persona* arbol = NULL;
    int opcion, id_padre, id_madre;
    string nombre, fecha;

    while (true) {
        system("clear || cls");

        cout << "\n+-----------------------------------------------------------------------------------------+\n";
        cout << "¦                          ARBOL GENEALOGICO                                              ¦\n";
        cout << "¦-----------------------------------------------------------------------------------------¦\n";
        cout << "¦  1. Agregar persona                                                                      ¦\n";
        cout << "¦  2. Eliminar persona                                                                     ¦\n";
        cout << "¦  3. Buscar persona                                                                       ¦\n";
        cout << "¦  4. Ver tabla de personas                                                               ¦\n";
        cout << "¦  5. Mostrar ancestros                                                                   ¦\n";
        cout << "¦  6. Mostrar descendientes                                                               ¦\n";
        cout << "¦  7. Ver recorridos del arbol                                                            ¦\n";
        cout << "¦  8. Salir                                                                               ¦\n";
        cout << "+-----------------------------------------------------------------------------------------+\n";
        cout << "Ingrese opcion: ";
        cin >> opcion;
        cin.ignore();
```

```
        switch(opcion) {
            case 1: {
                system("clear || cls");
                cout << "\n-------------------------------------------------------------------------------\n";
                cout << "                          AGREGAR NUEVA PERSONA \n";
                cout << "-------------------------------------------------------------------------------\n\n";

                int nuevoID = proximoID++;

                cout << " ID asignado automaticamente : " << nuevoID << "\n\n";

                cout << "Ingrese nombre: ";
                getline(cin, nombre);

                bool fechaValida = false;
                while (!fechaValida) {
                    cout << "Ingrese fecha de nacimiento (dd/mm/aaaa): ";
                    cin >> fecha;
                    if (esValida(fecha)) {
                        fechaValida = true;
                    } else {
                        cout << " Fecha invalida. Intente de nuevo (formato: dd/mm/aaaa, 1900-2025)\n";
                    }
                }
```

```
        cin.ignore();

        cout << "\n ¿Desea asignar padres? (s/n): ";
        char respuesta;
        cin >> respuesta;
        cin.ignore();

        Persona* padre = NULL;
        Persona* madre = NULL;

        if (respuesta == 's' || respuesta == 'S') {
            cout << "ID del padre (0 si no existe): ";
            cin >> id_padre;
            padre = (id_padre != 0) ? buscar(arbol, id_padre) : NULL;

            cout << "ID de la madre (0 si no existe): ";
            cin >> id_madre;
            madre = (id_madre != 0) ? buscar(arbol, id_madre) : NULL;
            cin.ignore();
        }

        insertar(arbol, nuevoID, nombre, fecha, padre, madre);
        agregarATabla(buscar(arbol, nuevoID));

        cout << "\n  Persona agregada correctamente\n";
        cout << "\n Presione ENTER para continuar...";
        cin.get();
        break;
    }
```

```
    case 2: {
        system("clear || cls");
        cout << "\n-------------------------------------------------------------------------------\n";
        cout << "                          ELIMINAR PERSONA \n";
        cout << "-------------------------------------------------------------------------------\n\n";

        int id;
        cout << "ID a eliminar: ";
        cin >> id;

        Persona* encontrado = buscar(arbol, id);
        if (encontrado != NULL) {
            arbol = eliminar(arbol, id);
            cout << "\n Persona eliminada correctamente \n";
        } else {
            cout << "\n Persona no encontrada\n";
        }

        cout << "\n Presione ENTER para continuar...";
        cin.ignore();
        cin.get();
        break;
    }
```

Capítulo 3 – Solución Final

Código Fuente:

```cpp
#include <iostream>
#include <string>
#include <iomanip>
#include <sstream>

using namespace std;

// ================================================================================
// ESTRUCTURA: Persona
// ================================================================================
struct Persona {
    int id;
    string nombre;
    string fecha_nac;
    Persona* padre;
    Persona* madre;
    Persona* izq;
    Persona* der;
};


// ================================================================================
// VALIDACIÓN DE FECHA
// ================================================================================
bool esValida(string fecha) {
    if (fecha.length() != 10 || fecha[2] != '/' || fecha[5] != '/') {
        return false;
    }

    for (int i = 0; i < 10; i++) {
        if (i != 2 && i != 5 && !isdigit(fecha[i])) {
            return false;
        }
    }

    int dia = stoi(fecha.substr(0, 2));
    int mes = stoi(fecha.substr(3, 2));
    int anio = stoi(fecha.substr(6, 4));

    if (mes < 1 || mes > 12) return false;
    if (dia < 1 || dia > 31) return false;
    if (mes == 2 && dia > 29) return false;
    if ((mes == 4 || mes == 6 || mes == 9 || mes == 11) && dia > 30) return false;
    if (anio < 1900 || anio > 2025) return false;

    return true;
}

// ================================================================================
// GESTIÓN DE ID AUTOMÁTICO
// ================================================================================
int proximoID = 1;

void actualizarProximoID(Persona* raiz) {
    if (raiz == NULL) return;
    if (raiz->id >= proximoID) {
        proximoID = raiz->id + 1;
    }
    actualizarProximoID(raiz->izq);
    actualizarProximoID(raiz->der);
}

// ================================================================================
// FUNCIÓN: crearPersona
// ================================================================================
Persona* crearPersona(int id, string nombre, string fecha) {
    Persona* nueva = new Persona;
    nueva->id = id;
    nueva->nombre = nombre;
    nueva->fecha_nac = fecha;
    nueva->padre = NULL;
    nueva->madre = NULL;
    nueva->izq = NULL;
    nueva->der = NULL;
    return nueva;
}

```

```cpp
// ================================================================================
// FUNCIÓN: insertar
// ================================================================================
void insertar(Persona* &raiz, int id, string nombre, string fecha, Persona* padre, Persona* madre) {
    if (raiz == NULL) {
        raiz = crearPersona(id, nombre, fecha);
        raiz->padre = padre;
        raiz->madre = madre;
        return;
    }

    if (id < raiz->id) {
        insertar(raiz->izq, id, nombre, fecha, padre, madre);
    } else if (id > raiz->id) {
        insertar(raiz->der, id, nombre, fecha, padre, madre);
    }
}
```

```cpp

// ================================================================================
// FUNCIÓN: buscar
// ================================================================================
Persona* buscar(Persona* raiz, int id) {
    if (raiz == NULL || raiz->id == id) {
        return raiz;
    }

    if (id < raiz->id) {
        return buscar(raiz->izq, id);
    } else {
        return buscar(raiz->der, id);
    }
}

// ================================================================================
// FUNCIÓN: encontrarMinimo
// ================================================================================
Persona* encontrarMinimo(Persona* raiz) {
    while (raiz->izq != NULL) {
        raiz = raiz->izq;
    }
    return raiz;
}

```

```cpp
// ================================================================================
// FUNCIÓN: eliminar
// ================================================================================
Persona* eliminar(Persona* raiz, int id) {
    if (raiz == NULL) return raiz;

    if (id < raiz->id) {
        raiz->izq = eliminar(raiz->izq, id);
    } else if (id > raiz->id) {
        raiz->der = eliminar(raiz->der, id);
    } else {
        if (raiz->izq == NULL) {
            Persona* temp = raiz->der;
            delete raiz;
            return temp;
        } else if (raiz->der == NULL) {
            Persona* temp = raiz->izq;
            delete raiz;
            return temp;
        }

        Persona* temp = encontrarMinimo(raiz->der);
        raiz->id = temp->id;
        raiz->nombre = temp->nombre;
        raiz->fecha_nac = temp->fecha_nac;
        raiz->der = eliminar(raiz->der, temp->id);
    }
    return raiz;
```

```
}

// ================================================================================
// TABLA DE DATOS
// ================================================================================
struct TablaPersonas {
    Persona* personas[100];
    int cantidad;
};

TablaPersonas tablaGlobal;

void inicializarTabla() {
    tablaGlobal.cantidad = 0;
}

void agregarATabla(Persona* p) {
    if (tablaGlobal.cantidad < 100) {
        tablaGlobal.personas[tablaGlobal.cantidad++] = p;
    }
}

void llenarTabla(Persona* raiz) {
    if (raiz == NULL) return;
    llenarTabla(raiz->izq);
    agregarATabla(raiz);
    llenarTabla(raiz->der);
}
```

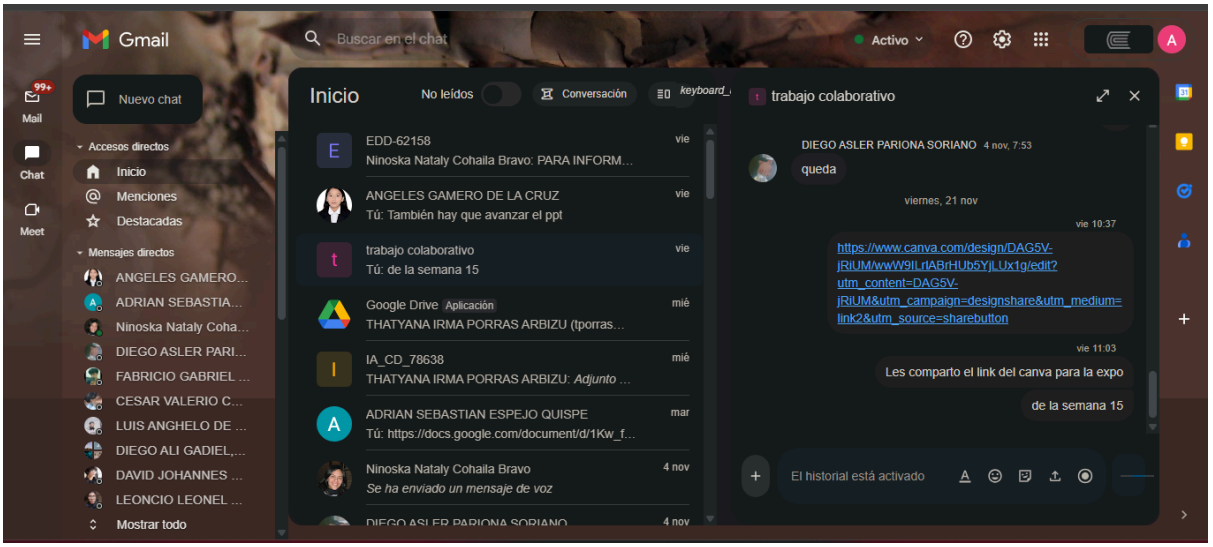
```
void mostrarTabla() {
    cout << "\n+-----------------------------------------------------------------------------------------+\n";
    cout << "¦                          TABLA DE PERSONAS                                              ¦\n";
    cout << "¦-----------------------------------------------------------------------------------------¦\n";
    cout << "¦ ID  ¦ NOMBRE              ¦ FECHA NACIMIENTO ¦ PADRE ¦ MADRE            ¦\n";
    cout << "¦-----------------------------------------------------------------------------------------¦\n";

    inicializarTabla();
    llenarTabla(tablaGlobal.personas[0] != NULL ? tablaGlobal.personas[0] : NULL);

    for (int i = 0; i < tablaGlobal.cantidad; i++) {
        Persona* p = tablaGlobal.personas[i];
        string padre = (p->padre != NULL) ? p->padre->nombre : "N/A";
        string madre = (p->madre != NULL) ? p->madre->nombre : "N/A";

        cout << "¦ " << setw(3) << p->id << " ¦ " << setw(20) << left << p->nombre
             << "¦ " << setw(16) << p->fecha_nac << "¦ " << setw(5) << (p->padre != NULL ? to_string(p->padre->id) : "N/A")
             << "¦ " << setw(5) << (p->madre != NULL ? to_string(p->madre->id) : "N/A") << "¦\n";
    }

    cout << "+-----------------------------------------------------------------------------------------+\n";
}
```

```
// ================================================================================
// FUNCIÓN: mostrarAncestros
// ================================================================================
void mostrarAncestros(Persona* persona, int nivel = 0) {
    if (persona == NULL) return;

    for (int i = 0; i < nivel; i++) cout << "  ";
    cout << persona->nombre << endl;

    if (persona->padre != NULL) {
        for (int i = 0; i < nivel; i++) cout << "  ";
        cout << "+- Padre: ";
        mostrarAncestros(persona->padre, nivel + 1);
    }

    if (persona->madre != NULL) {
        for (int i = 0; i < nivel; i++) cout << "  ";
        cout << "+- Madre: ";
        mostrarAncestros(persona->madre, nivel + 1);
    }
}
```

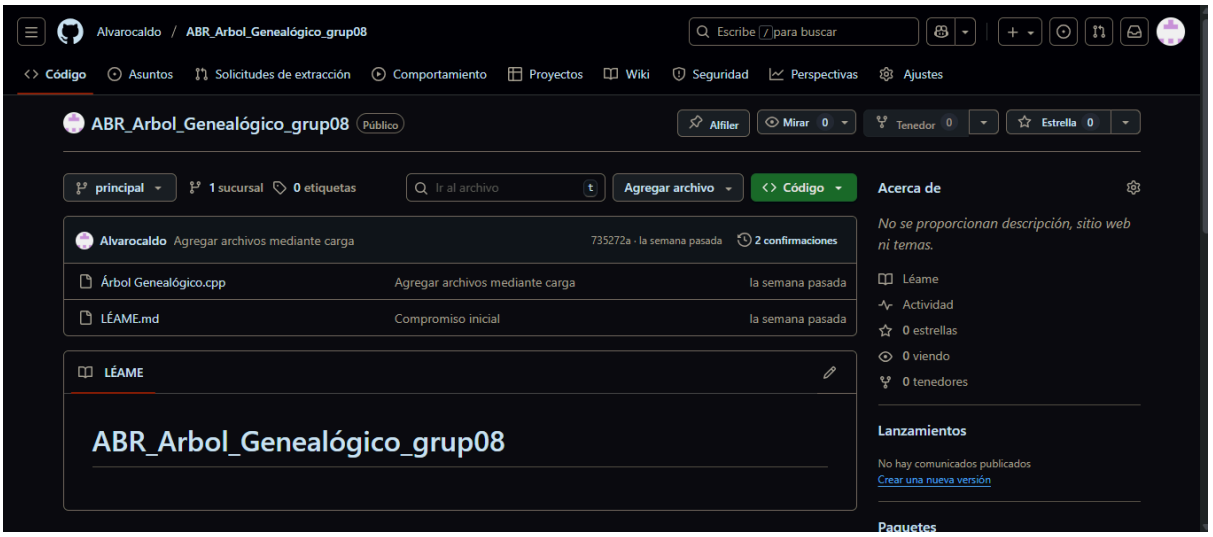
```cpp
// ============================================================================
// FUNCIÓN: mostrarDescendientes
// ============================================================================
void mostrarDescendientes(Persona* persona) {
    if (persona == NULL) return;

    cout << "Descendientes de " << persona->nombre << ":\n";
    if (persona->izq != NULL) {
        cout << "- " << persona->izq->nombre << endl;
        mostrarDescendientes(persona->izq);
    }
    if (persona->der != NULL) {
        cout << "- " << persona->der->nombre << endl;
        mostrarDescendientes(persona->der);
    }
}

// ============================================================================
// RECORRIDOS DEL ÁRBOL
// ============================================================================

// PREORDEN: Raíz - Izquierda - Derecha
void preorden(Persona* raiz) {
    if (raiz == NULL) return;
    cout << "[" << raiz->id << "] " << raiz->nombre << " (" << raiz->fecha_nac << ")\n";
    preorden(raiz->izq);
    preorden(raiz->der);
}

// INORDEN: Izquierda - Raíz - Derecha
void inorden(Persona* raiz) {
    if (raiz == NULL) return;
    inorden(raiz->izq);
    cout << "[" << raiz->id << "] " << raiz->nombre << " (" << raiz->fecha_nac << ")\n";
    inorden(raiz->der);
}

// POSTORDEN: Izquierda - Derecha - Raíz
void postorden(Persona* raiz) {
    if (raiz == NULL) return;
    postorden(raiz->izq);
    postorden(raiz->der);
    cout << "[" << raiz->id << "] " << raiz->nombre << " (" << raiz->fecha_nac << ")\n";
}

// ============================================================================
// ESTRUCTURA: Cola para recorrido por niveles
// ============================================================================
struct Cola {
    Persona* elementos[100];
    int frente;
    int final;
};

void inicializarCola(Cola &c) {
    c.frente = 0;
    c.final = -1;
}

// ============================================================================
// MENÚ PRINCIPAL
// ============================================================================
void menu() {
    Persona* arbol = NULL;
    int opcion, id_padre, id_madre;
    string nombre, fecha;

    while (true) {
        system("clear || cls");

        cout << "\n+-------------------------------------------------------------------------------+\n";
        cout << "|                          ARBOL GENEALOGICO                                    |\n";
        cout << "|-------------------------------------------------------------------------------|\n";
        cout << "|  1. Agregar persona                                                           |\n";
        cout << "|  2. Eliminar persona                                                          |\n";
        cout << "|  3. Buscar persona                                                            |\n";
        cout << "|  4. Ver tabla de personas                                                     |\n";
        cout << "|  5. Mostrar ancestros                                                         |\n";
        cout << "|  6. Mostrar descendientes                                                     |\n";
        cout << "|  7. Ver recorridos del arbol                                                  |\n";
        cout << "|  8. Salir                                                                     |\n";
        cout << "+-------------------------------------------------------------------------------+\n";
        cout << "Ingrese opcion: ";
        cin >> opcion;
        cin.ignore();
```

Capítulo 4 – Evidencias de Trabajo Colaborativo

Grupo:Gmail



git hub:



GITHUB:https://github.com/alvarocaldo/ABR_Arbol_Genealogico_grup08