Capítulo 5: Implementación

PREGUNTAS GUIA

1.¿Qué información se debe almacenar en cada nodo del árbol?
Cada nodo del árbol representará a una persona del árbol genealógico. Por lo tanto, es necesario almacenar información básica como:
Nombre completo: para identificar a la persona.
Generación: para su clasificación.
Género: para identificar relaciones de parentesco.
Relación familiar: identificar padre, madre, hijos, etc..

2.¿Cómo insertar y eliminar miembros del árbol sin romper su estructura?
Para insertar:
Si el nuevo miembro tiene menor valor que el nodo actual (por nombre o edad), va a la izquierda.
Si es de mayor valor irá a la derecha.
Este proceso lo haremos de forma recursiva o iterativa.
Para eliminar:
Si no tiene hijos: se elimina directamente.
Si tiene un solo hijo: el hijo sustituye al nodo eliminado.
Si tiene dos hijos: se reemplaza con el sucesor inorden (el menor de los mayores).

3.¿Qué métodos permiten recorrer el árbol para visualizar la geneología?
Para visualizar un árbol genealógico, los métodos más útiles son el recorrido por niveles (BFS) y el preorden. El recorrido por niveles muestra generación por generación (primero los abuelos, luego padres, después hijos, etc.), lo que es ideal para ver la estructura familiar completa de manera clara y ordenada. Por otro lado, el preorden (raíz → izquierda → derecha) es útil para seguir una línea específica, como la descendencia directa de un ancestro (ejemplo: abuelo → padre → hijo). Si buscas analizar relaciones heredadas, el postorden (hojas → raíz) puede ayudar, pero el más intuitivo para genealogía es el por niveles, ya que organiza la información generacionalmente.

4. ¿Cómo determinar si un miembro pertenece a una rama específica?
Podemos usar una búsqueda recursiva dentro del árbol, comparando el valor (nombre o edad) del nodo actual con el que se busca.
Si se encuentra, pertenece al árbol.
También podemos saber el camino desde la raíz hasta ese nodo para saber en qué rama está (izquierda o derecha).

5. ¿Cómo balancear el árbol si se vuelve demasiado profundo?
Cuando el árbol genealógico se vuelve muy profundo o desbalanceado, se puede balancear manualmente realizando un recorrido inorden (izquierda – raíz – derecha) para obtener todos los miembros en una lista ordenada. Luego, se guarda esta información en un arreglo auxiliar y se reconstruye el árbol eligiendo como nueva raíz el elemento del medio. Los elementos que están a la izquierda de esa raíz se colocan recursivamente en el subárbol izquierdo, y los que están a la derecha, en el subárbol derecho. Así se obtiene un árbol más equilibrado, lo que facilita la búsqueda y mantiene una estructura organizada.

SOLUCIÓN - PROTOTIPO

1. Descripción de estructuras de datos y operaciones

Estructura General del Sistema:

Para representar el árbol genealógico de esta antigua civilización, utilizaremos una estructura de nodos enlazados dinámicamente, ya que un árbol genealógico no tiene un número fijo de miembros ni una profundidad limitada. Cada persona será representada como un nodo con punteros a sus hijos/descendientes y un puntero a su padre.

Estructura de datos principal.

```
struct Persona {
    string nombre;
    Persona* padre;
    vector<Persona*> hijos;
    Persona(string_nombre) {
        nombre = _nombre;
        padre = nullptr;
    }
};
```

Descripción: nombre: almacenará el nombre que pongamos.

padre: nos servirá para rastrear a los ancestros.

hijos: esto es un vector que contiene punteros a los hijos.

-Agregar personas: Crearemos una nueva persona para añadirla al arbol.

```
Persona* agregarPersona(string nombre, Persona* padre = nullptr) {
    Persona* nueva = new Persona(nombre);
    nueva->padre = padre;
    if (padre != nullptr) {
        padre->hijos.push_back(nueva);
    }
    return nueva;
}
```

-Buscar persona por nombre: Utilizaremos este código para encontrar un nodo dado el nombre y también implementarlo de forma recursiva.

```
Persona* buscarPersona(Persona* raiz, string nombre) {
    if (raiz == nullptr) return nullptr;
    if (raiz->nombre == nombre) return raiz;
    for (Persona* hijo : raiz->hijos) {
        Persona* encontrado = buscarPersona(hijo, nombre);
        if (encontrado != nullptr) return encontrado;
    }
    return nullptr;
}
```

-Mostrar descendientes: Esto nos ayudará a mostrar a todos los descendientes de manera jerárquica osea en orden de antigüedad.

```
void mostrarDescendientes(Persona* persona, int nivel = 0) {
    for (int i = 0; i < nivel; ++i) cout << "  ";
    cout << persona->nombre << endl;
```

```
    for (Persona* hijo : persona->hijos) {
        mostrarDescendientes(hijo, nivel + 1);
    }
}
```

-Mostrar ancestros: Subirá por el árbol creado desde el nodo hasta la raíz y nos mostrará cada ancestro.

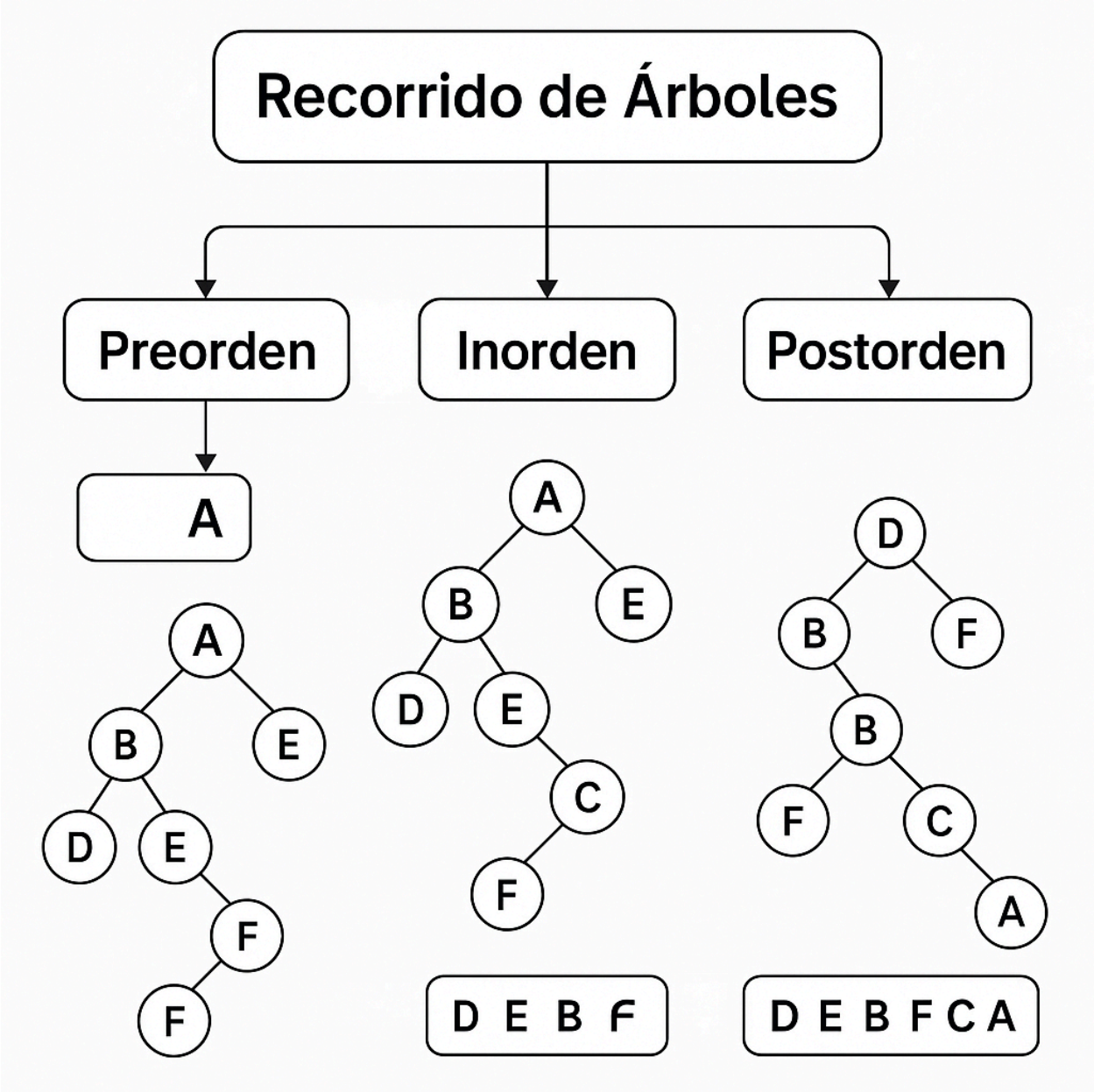
```
void mostrarAncestros(Persona* persona) {
    Persona* actual = persona->padre;
    while (actual != nullptr) {
        cout << actual->nombre << " -> ";
        actual = actual->padre;
    }
    cout << "Origen\n";
}
```

-Verificar relación entre dos personas: Nos mostrará si una persona es ancestro o descendiente de otra. Por eso es necesario usar los valores booleanos (v)(f).

```
bool esAncestro(Persona* posibleAncestro, Persona* persona) {
    Persona* actual = persona->padre;
    while (actual != nullptr) {
        if (actual == posibleAncestro) return true;
        actual = actual->padre;
    }
    return false;
}
```

En resumen esta estructura del árbol nos permite representar relaciones jerárquicas , el uso de los punteros y vectores nos facilitará la expansión dinámica y las operaciones nos permiten consultar información ancestral y genealógica.

2. Diagrama de recorrido de un árbol



3. Un Árbol Binario.

pseudocodigo:

```
Algoritmo InsertarEnArbolBinario

    Definir Nodo como registro
        valor: Entero
        izquierdo: Nodo
        derecho: Nodo
    FinDefinir

    Funcion CrearNodo(valor: Entero) : Nodo
        nuevo ← nuevo Nodo
        nuevo.valor ← valor
        nuevo.izquierdo ← Nulo
        nuevo.derecho ← Nulo
        Retornar nuevo
    FinFuncion

    Funcion Insertar(nodo: Nodo, valor: Entero) : Nodo
        Si nodo = Nulo Entonces
            nodo ← CrearNodo(valor)
        Sino
            Si valor < nodo.valor Entonces
                nodo.izquierdo ← Insertar(nodo.izquierdo, valor)
            Sino
                nodo.derecho ← Insertar(nodo.derecho, valor)
            FinSi
        FinSi
        Retornar nodo
    FinFuncion

    // Programa principal
    Definir raiz Como Nodo
    raiz ← Nulo

    Mientras hayan datos por insertar hacer
        Escribir "Ingrese un número:"
        Leer dato
        raiz ← Insertar(raiz, dato)
    FinMientras

FinAlgoritmo
```

4. Prototipo de código

```
#include <iostream>
#include <string>
#include <fstream>
#include <sstream>
using namespace std;

// Estructura para un nodo del arbol genealogico
struct Nodo {
    string nombreCompleto;      // Nombre completo del miembro
    string genero;              // Genero del miembro
    string relacionFamiliar;    // Relacion familiar
    Nodo* izquierda;            // Puntero al hijo izquierdo
    Nodo* derecha;              // Puntero al hijo derecho

    // Constructor para inicializar los datos del nodo
    Nodo(string nombre, string gen, string relacion)
        : nombreCompleto(nombre), genero(gen), relacionFamiliar(relacion), izquierda(NULL),
derecha(NULL) {}
};

// Funcion para agregar un nuevo miembro al arbol genealogico

void agregarMiembro(Nodo*& raiz) {
    string nombre, genero, relacion;
    cout << "Ingrese el nombre completo: ";
    cin >> nombre; // Usa cin para leer el nombre
    cout << "Ingrese el genero: ";
    cin >> genero; // Usa cin para leer el genero
    cout << "Ingrese la relacion familiar: ";
    cin >> relacion; // Usa cin para leer la relacion

    Nodo* nuevo = new Nodo(nombre, genero, relacion);

    // Si el arbol esta vacio, el nuevo nodo sera la raiz
    if (raiz == NULL) {
        raiz = nuevo;
        cout << "Miembro agregado como raiz del arbol." << endl;
        return;
    }

    // Busca la posicion correcta para insertar el nuevo nodo
    Nodo* actual = raiz;
    Nodo* padre = NULL;
```

```
    while (actual != NULL) {
        padre = actual;
        if (nombre < actual->nombreCompleto)
            actual = actual->izquierda;
        else
            actual = actual->derecha;
    }
    // Inserta el nuevo nodo como hijo izquierdo o derecho segun corresponda
    if (nombre < padre->nombreCompleto)
        padre->izquierda = nuevo;
    else
        padre->derecha = nuevo;

    cout << "Miembro agregado correctamente." << endl;
}

// Funcion recursiva para eliminar un miembro del arbol genealagico

bool eliminarMiembro(Nodo*& raiz, const string& nombre) {
    if (raiz == NULL) {
        return false; // No se encontro el miembro
    }

    if (nombre < raiz->nombreCompleto) {
        return eliminarMiembro(raiz->izquierda, nombre);
    } else if (nombre > raiz->nombreCompleto) {
        return eliminarMiembro(raiz->derecha, nombre);
    } else {
        // Nodo encontrado
        if (raiz->izquierda == NULL && raiz->derecha == NULL) {
            // Sin hijos
            cout << "El miembro \"" << raiz->nombreCompleto << "\" no tiene descendencia.
Eliminando..." << endl;
            delete raiz;
            raiz = NULL;
        } else if (raiz->izquierda == NULL || raiz->derecha == NULL) {
            // Un solo hijo
            cout << "ALERTA: El miembro \"" << raiz->nombreCompleto << "\" tiene descendencia.
Eliminando y reubicando descendientes..." << endl;
            Nodo* temp = (raiz->izquierda != NULL) ? raiz->izquierda : raiz->derecha;
            delete raiz;
            raiz = temp;
        } else {
            // Dos hijos
```

```
        cout << "ALERTA: El miembro \"" << raiz->nombreCompleto << "\" tiene descendencia.
Eliminando y reubicando descendientes..." << endl;
        // Buscar el sucesor inorden (el menor de la rama derecha)
        Nodo* sucesor = raiz->derecha;
        Nodo* padreSucesor = raiz;
        while (sucesor->izquierda != NULL) {
            padreSucesor = sucesor;
            sucesor = sucesor->izquierda;
        }
        // Copiar datos del sucesor
        raiz->nombreCompleto = sucesor->nombreCompleto;
        raiz->genero = sucesor->genero;
        raiz->relacionFamiliar = sucesor->relacionFamiliar;
        // Eliminar el sucesor
        if (padreSucesor->izquierda == sucesor)
            eliminarMiembro(padreSucesor->izquierda, sucesor->nombreCompleto);
        else
            eliminarMiembro(padreSucesor->derecha, sucesor->nombreCompleto);
      }
      return true;
   }
}

// Función para buscar un miembro en el árbol
Nodo* buscarMiembro(Nodo* raiz, const string& nombre) {
   if (raiz == NULL || raiz->nombreCompleto == nombre) {
      return raiz;
   }
   if (nombre < raiz->nombreCompleto) {
      return buscarMiembro(raiz->izquierda, nombre);
   } else {
      return buscarMiembro(raiz->derecha, nombre);
   }
}

// Función para mostrar los datos de un miembro encontrado
void buscarMiembroPrompt(Nodo* raiz) {
   string nombre;
   cout << "Ingrese el nombre completo del miembro a buscar: ";
   cin >> nombre; // Usa cin para leer el nombre directamente

   Nodo* encontrado = buscarMiembro(raiz, nombre);
   if (encontrado != NULL) {
      cout << "\n=== Miembro Encontrado ===" << endl;
```

```
        cout << "Nombre: " << encontrado->nombreCompleto << endl;
        cout << "Género: " << encontrado->genero << endl;
        cout << "Relación: " << encontrado->relacionFfamiliar << endl;
    } else {
        cout << "Miembro no encontrado en el árbol." << endl;
    }
}

// Función para mostrar los ancestros de un miembro (camino desde la raíz hasta el nodo)
void mostrarAncestros(Nodo* raiz, const string& nombre) {
    if (raiz == NULL) {
        cout << "Miembro no encontrado." << endl;
        return;
    }
    if (raiz->nombreCompleto == nombre) {
        cout << "=== Ancestros de " << nombre << " ===" << endl;
        return;
    }
    cout << raiz->nombreCompleto << " -> ";
    if (nombre < raiz->nombreCompleto) {
        mostrarAncestros(raiz->izquierda, nombre);
    } else {
        mostrarAncestros(raiz->derecha, nombre);
    }
}

// Función para solicitar el nombre del miembro y mostrar sus ancestros
void mostrarAncestrosPrompt(Nodo* raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro para ver sus ancestros: ";
    cin >> nombre; // Usa cin para leer el nombre directamente

    if (buscarMiembro(raiz, nombre) == NULL) {
        cout << "Miembro no encontrado." << endl;
        return;
    }
    mostrarAncestros(raiz, nombre);
}

// Función auxiliar para recorrer y mostrar todos los descendientes de un nodo (inorden)
void mostrarDescendientesRec(Nodo* nodo) {
    if (nodo == NULL) return;
    mostrarDescendientesRec(nodo->izquierda);
    cout << "- " << nodo->nombreCompleto << " (" << nodo->relacionFamiliar << ")" << endl;
```

```
    mostrarDescendientesRec(nodo->derecha);
}

// Función para mostrar los descendientes de un miembro
void mostrarDescendientes(Nodo* raiz, const string& nombre) {
    Nodo* miembro = buscarMiembro(raiz, nombre);
    if (miembro == NULL) {
        cout << "Miembro no encontrado." << endl;
        return;
    }
    cout << "\n=== Descendientes de " << nombre << " ===" << endl;
    if (miembro->izquierda == NULL && miembro->derecha == NULL) {
        cout << "No tiene descendientes." << endl;
    } else {
        mostrarDescendientesRec(miembro->izquierda);
        mostrarDescendientesRec(miembro->derecha);
    }
}

// Función para solicitar el nombre del miembro y mostrar sus descendientes
void mostrarDescendientesPrompt(Nodo* raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro para ver sus descendientes: ";
    cin >> nombre; // Usa cin para leer el nombre directamente

    mostrarDescendientes(raiz, nombre);
}


// Funcion para solicitar al usuario el nombre del miembro a eliminar y llamar a la funcion de
eliminacion
void eliminarMiembroPrompt(Nodo*& raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro a eliminar: ";
    cin >> nombre; // Usa cin para leer el nombre directamente

    if (!eliminarMiembro(raiz, nombre)) {
        cout << "Miembro no encontrado en el arbol." << endl;
    }
}

void visualizarArbol(Nodo*& raiz) {

}
```

```
// Recorrido Inorden: izquierda -> raíz -> derecha
void recorridoInorden(Nodo* raiz) {
    if (raiz == NULL) return;
    recorridoInorden(raiz->izquierda);
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " << raiz->genero << ")"
<< endl;
    recorridoInorden(raiz->derecha);
}

// Recorrido Preorden: raíz -> izquierda -> derecha
void recorridoPreorden(Nodo* raiz) {
    if (raiz == NULL) return;
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " << raiz->genero << ")"
<< endl;
    recorridoPreorden(raiz->izquierda);
    recorridoPreorden(raiz->derecha);
}

// Recorrido Postorden: izquierda -> derecha -> raíz
void recorridoPostorden(Nodo* raiz) {
    if (raiz == NULL) return;
    recorridoPostorden(raiz->izquierda);
    recorridoPostorden(raiz->derecha);
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " << raiz->genero << ")"
<< endl;
}
void mostrarMenu() {
    cout << "===== MENU ARBOL GENEALOGICO =====" << endl;
    cout << "1. Agregar miembro al arbol genealogico" << endl;
    cout << "2. Eliminar miembro del arbol genealogico" << endl;
    cout << "3. Buscar miembro" << endl;
    cout << "4. Visualizar el arbol (recorridos)" << endl;
    cout << "5. Mostrar ancestros" << endl;
    cout << "6. Mostrar descendientes" << endl;
    cout << "7. Verificar pertenencia a una rama" << endl;
    cout << "0. Salir" << endl;
    cout << "Seleccione una opcian: ";
}

// Funcion recursiva para guardar el arbol en un archivo

void guardarArbol(Nodo* raiz, ofstream& archivo) {
    if (raiz == NULL) return;
    guardarArbol(raiz->izquierda, archivo);
```

```
    archivo << raiz->nombreCompleto << "|" << raiz->genero << "|" << raiz->relacionFamiliar <<
endl;
    guardarArbol(raiz->derecha, archivo);
}

// Funcion para guardar el arbol genealogico en un archivo de texto

void guardarArbolEnArchivo(Nodo* raiz) {
    ofstream archivo("arbol.txt");
    if (!archivo.is_open()) {
        cout << "No se pudo abrir el archivo para guardar." << endl;
        return;
    }
    guardarArbol(raiz, archivo);
    archivo.close();
    cout << "arbol guardado correctamente en 'arbol.txt'." << endl;
}

// Funcion para insertar un nuevo nodo en el arbol genealogico
void insertarNodo(Nodo*& raiz, const string& nombre, const string& genero, const string&
relacion) {
    Nodo* nuevo = new Nodo(nombre, genero, relacion);
    if (raiz == NULL) {
        raiz = nuevo;
        return;
    }
    Nodo* actual = raiz;
    Nodo* padre = NULL;
    while (actual != NULL) {
        padre = actual;
        if (nombre < actual->nombreCompleto)
            actual = actual->izquierda;
        else
            actual = actual->derecha;
    }
    if (nombre < padre->nombreCompleto)
        padre->izquierda = nuevo;
    else
        padre->derecha = nuevo;
}

// Funcion para cargar el arbol genealogico desde un archivo de texto
void cargarArbolDesdeArchivo(Nodo*& raiz) {
    ifstream archivo("arbol.txt");
```

```
    if (!archivo.is_open()) {
        cout << "No se pudo abrir el archivo para cargar." << endl;
        return;
    }

    string nombre, genero, relacion;
    while (archivo >> nombre >> genero >> relacion) { // Leer directamente usando >>
        insertarNodo(raiz, nombre, genero, relacion);
    }

    archivo.close();
    cout << "Arbol cargado correctamente desde 'arbol.txt'." << endl;
}
// Función para mostrar opciones de recorrido del árbol
void visualizarArbol(Nodo* raiz) {
    int opcion;
    cout << "=== Visualización del Árbol Genealógico ===" << endl;
    cout << "1. Recorrido Inorden (ordenado alfabéticamente)" << endl;
    cout << "2. Recorrido Preorden (estructura de árbol)" << endl;
    cout << "3. Recorrido Postorden (descendientes primero)" << endl;
    cout << "Seleccione un tipo de recorrido: ";
    cin >> opcion;

    cout << "\nResultado del recorrido:" << endl;
    switch(opcion) {
        case 1:
            recorridoInorden(raiz);
            break;
        case 2:
            recorridoPreorden(raiz);
            break;
        case 3:
            recorridoPostorden(raiz);
            break;
        default:
            cout << "Opción inválida." << endl;
    }
}
int main() {
    Nodo* raiz = NULL;
    int opcion;

    // Cargar datos al iniciar
    cargarArbolDesdeArchivo(raiz);
```

```
    do {
        mostrarMenu();
        cin >> opcion;
        switch(opcion) {
            case 1:
                agregarMiembro(raiz);
                break;
            case 2:
                eliminarMiembroPrompt(raiz);
                break;
            case 3:
                buscarMiembroPrompt(raiz);
                break;
            case 4:
                visualizarArbol(raiz);
                break;
            case 5:
                mostrarAncestrosPrompt(raiz);
                break;
            case 6:
                mostrarDescendientesPrompt(raiz);
                break;
            case 7:

                break;
            case 0:
                // Guardar datos antes de salir
                guardarArbolEnArchivo(raiz);
                cout << "Saliendo..." << endl;
                break;
            default:
                cout << "Opcion invalida. Intente de nuevo." << endl;
        }
        cout << endl;
    } while(opcion != 0);

    return 0;
}
```

VALIDACIÓN:

Proyecto: Árbol genealógico

Curso: Estructuras de DaABRtos
Unidad: Unidad 3 –
Equipo: Grupo 7

Presentación: Canva

GitHub:
https://github.com/AntoniRaul/Arbol-Genealogico/blob/main/Arbol_Genelogico.cpp

1. Manual de Usuario

Opciones del menú

1. Agregar miembro al árbol genealógico

- Ingresa el nombre, género y relación familiar del nuevo miembro.
- Si es el primer miembro, será la raíz del árbol.
- Si no, deberás indicar el nombre del padre/madre (o "ninguno" si no tiene).

2. Eliminar miembro del árbol genealógico

- Ingresa el nombre del miembro que deseas eliminar.
- No puedes eliminar la raíz si tiene descendientes.

3. Buscar miembro

- Ingresa el nombre del miembro.
- El programa mostrará sus datos y, si tiene, el nombre de su padre/madre.

4. Visualizar el árbol (recorridos)

- Muestra el árbol familiar de forma jerárquica, para ver cómo están conectados los miembros.

5. Mostrar ancestros

- Ingresa el nombre de un miembro.
- El programa mostrará la cadena de ancestros (padres, abuelos, etc.) de ese miembro.

6. Mostrar descendientes

- Ingresa el nombre de un miembro.
- El programa mostrará todos los hijos, nietos, etc., de ese miembro.

7. Verificar pertenencia a una rama

- Ingresa el nombre de un miembro y el de un ancestro.
- El programa te dirá si ese miembro pertenece a la rama descendiente de ese ancestro.

0. Salir

- Guarda automáticamente el árbol y cierra el programa.

Consejos

- Escribe los nombres exactamente igual cada vez (sin espacios si así lo pide el programa).
- Si tienes dudas sobre qué ingresar, sigue las instrucciones que aparecen en pantalla.
- La información se guarda automáticamente al salir.

Problemas comunes

- No encuentro a un miembro: Asegúrate de escribir el nombre exactamente como lo ingresaste.
- No puedo eliminar la raíz: Solo puedes eliminar la raíz si no tiene descendientes.
- El árbol está vacío: Agrega un miembro para comenzar.

Soporte

- Si tienes problemas, revisa que el archivo arbol.txt esté en la misma carpeta que el programa. Si no existe, se creará automáticamente al guardar.

2. Código

```
#include <iostream> // Incluye la biblioteca estándar para entrada y salida.
#include <string>   // Incluye la biblioteca para manejar cadenas de texto.
#include <fstream>  // Incluye la biblioteca para manejo de archivos.
#include <sstream>  // Incluye la biblioteca para manipulación de strings como streams.
using namespace std; // Usa el espacio de nombres estándar para evitar escribir std::

// Estructura para un nodo del arbol genealogico
struct Nodo {
   string nombreCompleto;        // Nombre completo del miembro.
   string genero;                // Género del miembro.
   string relacionFamiliar;      // Relación familiar (hijo, padre, etc).
   Nodo* padre;                  // Puntero al padre.
   Nodo* hijo;                   // Puntero al primer hijo.
   Nodo* hermano;                // Puntero al siguiente hermano.

   // Constructor del nodo, inicializa los campos y punteros.
   Nodo(string nombre, string gen, string relacion, Nodo* p = NULL)
       : nombreCompleto(nombre), genero(gen), relacionFamiliar(relacion), padre(p),
hijo(NULL), hermano(NULL) {}
};

// Implementación de funciones (ahora definidas antes de main())

// Busca un miembro por nombre en el árbol, recursivamente.
Nodo* buscarMiembro(Nodo* raiz, const string& nombre) {
   if (raiz == NULL) return NULL; // Si el nodo es nulo, retorna NULL.
   if (raiz->nombreCompleto == nombre) // Si el nombre coincide, retorna el nodo.
```

```
        return raiz;

    Nodo* encontrado = buscarMiembro(raiz->hijo, nombre); // Busca en los hijos.
    if (encontrado != NULL) return encontrado; // Si lo encuentra en los hijos, retorna el
nodo.

    return buscarMiembro(raiz->hermano, nombre); // Si no, busca en los hermanos.
}

// Muestra los ancestros de un nodo, desde la raíz hasta el padre inmediato.
void mostrarAncestros(Nodo* nodo) {
    if (nodo == NULL || nodo->padre == NULL) return; // Si el nodo o su padre es nulo,
termina.

    mostrarAncestros(nodo->padre); // Llama recursivamente para mostrar ancestros
superiores.
    cout << nodo->padre->nombreCompleto; // Muestra el nombre del padre.
    if (nodo->padre->padre != NULL) { // Si hay más ancestros, imprime flecha.
        cout << " -> ";
    }
}

// Muestra recursivamente los descendientes de un nodo.
void mostrarDescendientesRec(Nodo* nodo) {
    if (nodo == NULL) return; // Si el nodo es nulo, termina.

    cout << "- " << nodo->nombreCompleto << " (" << nodo->relacionFamiliar << ")" <<
endl; // Muestra el nombre y relación.
    mostrarDescendientesRec(nodo->hijo); // Llama recursivamente para mostrar los
hijos.
    mostrarDescendientesRec(nodo->hermano); // Llama recursivamente para mostrar
los hermanos.
}

// Busca un miembro y muestra sus descendientes.
void mostrarDescendientes(Nodo* raiz, const string& nombre) {
    Nodo* miembro = buscarMiembro(raiz, nombre); // Busca el miembro por nombre.
    if (miembro == NULL) { // Si no lo encuentra, muestra mensaje.
        cout << "Miembro no encontrado." << endl;
        return;
    }

    cout << "\n=== Descendientes de " << nombre << " ===" << endl; // Encabezado.
    if (miembro->hijo == NULL) { // Si no tiene hijos, lo indica.
```

```
        cout << "No tiene descendientes." << endl;
    } else {
        mostrarDescendientesRec(miembro->hijo); // Muestra los descendientes.
    }
}

// Visualiza el árbol en forma jerárquica.
void visualizarArbol(Nodo* nodo, string prefijo = "", bool esUltimo = true) {
    if (nodo == NULL) return; // Si el nodo es nulo, termina.

    cout << prefijo; // Imprime el prefijo para la jerarquía.

    // Imprime el símbolo de rama.
    if (esUltimo) {
        cout << "+-- ";
    } else {
        cout << "+-- ";
    }

    cout << nodo->nombreCompleto << " (" << nodo->genero << ", " <<
nodo->relacionFamiliar << ")" << endl; // Muestra datos del nodo.

    string nuevoPrefijo;
    if (esUltimo) {
        nuevoPrefijo = prefijo + "    "; // Espaciado para el último hijo.
    } else {
        nuevoPrefijo = prefijo + "¦   "; // Espaciado para otros hijos.
    }

    Nodo* hijo = nodo->hijo; // Empieza con el primer hijo.
    while (hijo != NULL) {
        bool ultimoHijo = (hijo->hermano == NULL); // Verifica si es el último hijo.
        visualizarArbol(hijo, nuevoPrefijo, ultimoHijo); // Llama recursivamente.
        hijo = hijo->hermano; // Pasa al siguiente hermano.
    }
}

// Recorrido inorden del árbol (hijos, nodo, hermanos).
void recorridoInorden(Nodo* raiz) {
    if (raiz == NULL) return; // Si el nodo es nulo, termina.
    recorridoInorden(raiz->hijo); // Llama recursivamente para los hijos.
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " <<
raiz->genero << ")" << endl;
    Nodo* hermano = raiz->hermano;
```

```
    while (hermano != NULL) { // Recorre los hermanos.
        recorridoInorden(hermano);
        hermano = hermano->hermano; // Pasa al siguiente hermano.
    }
}

// Recorrido preorden del árbol (nodo, hijos, hermanos).
void recorridoPreorden(Nodo* raiz) {
    if (raiz == NULL) return;
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " <<
raiz->genero << ")" << endl;
    recorridoPreorden(raiz->hijo);
    recorridoPreorden(raiz->hermano);
}

// Recorrido postorden del árbol (hijos, nodo, hermanos).
void recorridoPostorden(Nodo* raiz) {
    if (raiz == NULL) return;
    recorridoPostorden(raiz->hijo);
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " <<
raiz->genero << ")" << endl;
    recorridoPostorden(raiz->hermano);
}

// Guarda un nodo y sus descendientes en un archivo.
void guardarNodo(Nodo* nodo, ofstream& archivo) {
    if (nodo == NULL) return;

    archivo << nodo->nombreCompleto << "|" << nodo->genero << "|"
            << nodo->relacionFamiliar << "|";
    if (nodo->padre != NULL) {
        archivo << nodo->padre->nombreCompleto;
    }
    archivo << endl;

    guardarNodo(nodo->hijo, archivo);
    guardarNodo(nodo->hermano, archivo);
}

// Guarda todo el árbol en un archivo de texto.
void guardarArbolEnArchivo(Nodo* raiz) {
    ofstream archivo("arbol.txt");
    if (!archivo.is_open()) {
        cout << "No se pudo abrir el archivo para guardar." << endl;
```

```
        return;
    }
    guardarNodo(raiz, archivo);
    archivo.close();
    cout << "Árbol guardado correctamente en 'arbol.txt'." << endl;
}

// Carga el árbol desde un archivo de texto.
void cargarArbolDesdeArchivo(Nodo*& raiz) {
    ifstream archivo("arbol.txt");
    if (!archivo.is_open()) {
        cout << "No se encontró archivo para cargar. Se creará uno nuevo al guardar." <<
endl;
        return;
    }

    string linea;
    while (getline(archivo, linea)) {
        stringstream ss(linea);
        string nombre, genero, relacion, nombrePadre;

        getline(ss, nombre, '|');
        getline(ss, genero, '|');
        getline(ss, relacion, '|');
        getline(ss, nombrePadre);

        if (raiz == NULL) {
            raiz = new Nodo(nombre, genero, relacion);
        } else {
            Nodo* padre = nombrePadre.empty() ? NULL : buscarMiembro(raiz,
nombrePadre);
            Nodo* nuevo = new Nodo(nombre, genero, relacion, padre);

            if (padre != NULL) {
                if (padre->hijo == NULL) {
                    padre->hijo = nuevo;
                } else {
                    Nodo* temp = padre->hijo;
                    while (temp->hermano != NULL) {
                        temp = temp->hermano;
                    }
                    temp->hermano = nuevo;
                }
            } else {
```

```
                Nodo* temp = raiz;
                while (temp->hermano != NULL) {
                    temp = temp->hermano;
                }
                temp->hermano = nuevo;
            }
        }
    }

    archivo.close();
    cout << "Árbol cargado correctamente desde 'arbol.txt'." << endl;
}

// Visualiza el árbol completo con leyenda.
void visualizarArbolCompleto(Nodo* raiz) {
    if (raiz == NULL) {
        cout << "El árbol genealógico está vacío." << endl;
        return;
    }

    cout << "\n=== ÁRBOL GENEALÓGICO ===\n";
    cout << "Leyenda:\n";
    cout << "+-- Miembro principal\n";
    cout << "    +-- Hijo/hermano\n";
    cout << "    +-- Último hijo/hermano\n\n";

    visualizarArbol(raiz);

    cout << "\n";
}

// Verifica si un miembro pertenece a una rama específica.
bool verificarPertenenciaARama(Nodo* raizRama, const string& nombreMiembro) {
    if (raizRama == NULL) return false;
    if (raizRama->nombreCompleto == nombreMiembro) return true;

    Nodo* hijo = raizRama->hijo;
    while (hijo != NULL) {
        if (verificarPertenenciaARama(hijo, nombreMiembro)) {
            return true;
        }
        hijo = hijo->hermano;
    }
    return false;
```

```
}

// Agrega un nuevo miembro al árbol genealógico.
void agregarMiembro(Nodo*& raiz) {
    string nombre, genero, relacion, nombrePadre;

    cout << "Ingrese el nombre completo (sin espacios): "; cin >> nombre;
    cout << "Ingrese el género: "; cin >> genero;
    cout << "Ingrese la relación familiar: "; cin >> relacion;

    if (raiz == NULL) {
        raiz = new Nodo(nombre, genero, relacion);
        cout << "Miembro agregado como raíz del árbol." << endl;
        return;
    }

    cout << "Ingrese el nombre del padre/madre (escriba 'ninguno' si no tiene): "; cin >>
nombrePadre;

    if (nombrePadre == "ninguno") {
        Nodo* nuevo = new Nodo(nombre, genero, relacion);
        Nodo* temp = raiz;
        while (temp->hermano != NULL) {
            temp = temp->hermano;
        }
        temp->hermano = nuevo;
        cout << "Miembro agregado como familiar extendido." << endl;
    } else {
        Nodo* padre = buscarMiembro(raiz, nombrePadre);
        if (padre == NULL) {
            cout << "Padre/Madre no encontrado. Miembro no agregado." << endl;
            return;
        }

        Nodo* nuevo = new Nodo(nombre, genero, relacion, padre);

        if (padre->hijo == NULL) {
            padre->hijo = nuevo;
        } else {
            Nodo* temp = padre->hijo;
            while (temp->hermano != NULL) {
                temp = temp->hermano;
            }
            temp->hermano = nuevo;
```

```
        }
        cout << "Miembro agregado correctamente." << endl;
    }
}

// Elimina un miembro del árbol genealógico.
bool eliminarMiembro(Nodo*& raiz, const string& nombre) {
    if (raiz == NULL) return false;

    Nodo* nodo = buscarMiembro(raiz, nombre);
    if (nodo == NULL) return false;

    if (nodo == raiz) {
        if (nodo->hijo == NULL && nodo->hermano == NULL) {
            delete nodo;
            raiz = NULL;
        } else {
            cout << "No se puede eliminar la raíz con descendientes." << endl;
            return false;
        }
        return true;
    }

    Nodo* padre = nodo->padre;

    if (padre != NULL && padre->hijo == nodo) {
        if (nodo->hijo != NULL) {
            Nodo* temp = nodo->hijo;
            while (temp != NULL) {
                temp->padre = padre;
                temp = temp->hermano;
            }
            temp = nodo->hijo;
            while (temp->hermano != NULL) {
                temp = temp->hermano;
            }
            temp->hermano = padre->hijo->hermano;
            padre->hijo = nodo->hijo;
        } else {
            padre->hijo = nodo->hermano;
        }
        delete nodo;
        return true;
    }
```

```
    Nodo* anterior = padre != NULL ? padre->hijo : raiz;
    while (anterior != NULL && anterior->hermano != nodo) {
        anterior = anterior->hermano;
    }

    if (anterior != NULL) {
        if (nodo->hijo != NULL) {
            Nodo* temp = nodo->hijo;
            while (temp != NULL) {
                temp->padre = padre;
                temp = temp->hermano;
            }
            temp = nodo->hijo;
            while (temp->hermano != NULL) {
                temp = temp->hermano;
            }
            temp->hermano = nodo->hermano;
            anterior->hermano = nodo->hijo;
        } else {
            anterior->hermano = nodo->hermano;
        }
        delete nodo;
        return true;
    }

    return false;
}

// Solicita el nombre y busca un miembro, mostrando sus datos.
void buscarMiembroPrompt(Nodo* raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro a buscar: "; cin >> nombre;

    Nodo* encontrado = buscarMiembro(raiz, nombre);
    if (encontrado != NULL) {
        cout << "\n=== Miembro Encontrado ===" << endl;
        cout << "Nombre: " << encontrado->nombreCompleto << endl;
        cout << "Género: " << encontrado->genero << endl;
        cout << "Relación: " << encontrado->relacionFamiliar << endl;
        if (encontrado->padre != NULL) {
            cout << "Padre/Madre: " << encontrado->padre->nombreCompleto << endl;
        }
    } else {
```

```
        cout << "Miembro no encontrado en el árbol." << endl;
    }
}

// Solicita el nombre y muestra los ancestros del miembro.
void mostrarAncestrosPrompt(Nodo* raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro para ver sus ancestros: "; cin >>
nombre;

    Nodo* miembro = buscarMiembro(raiz, nombre);
    if (miembro == NULL) {
        cout << "Miembro no encontrado." << endl;
        return;
    }

    cout << "\n=== Ancestros de " << nombre << " ===" << endl;
    if (miembro->padre == NULL) {
        cout << "No tiene ancestros registrados (es la raíz del árbol)." << endl;
    } else {
        mostrarAncestros(miembro);
        cout << endl;
    }
}

// Solicita el nombre y muestra los descendientes del miembro.
void mostrarDescendientesPrompt(Nodo* raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro para ver sus descendientes: "; cin
>> nombre;

    mostrarDescendientes(raiz, nombre);
}

// Solicita el nombre y elimina un miembro del árbol.
void eliminarMiembroPrompt(Nodo*& raiz) {
    string nombre;
    cout << "Ingrese el nombre completo del miembro a eliminar: "; cin >> nombre;

    if (!eliminarMiembro(raiz, nombre)) {
        cout << "Miembro no encontrado en el árbol o no se pudo eliminar." << endl;
    } else {
        cout << "Miembro eliminado correctamente." << endl;
    }
```

```
}

// Solicita nombres y verifica si un miembro pertenece a la rama de un ancestro.
void verificarPertenenciaPrompt(Nodo* raiz) {
    if (raiz == NULL) {
        cout << "El árbol genealógico está vacío." << endl;
        return;
    }

    string nombreMiembro, nombreAncestro;

    cout << "Ingrese el nombre del miembro a verificar: "; cin >> nombreMiembro;
    cout << "Ingrese el nombre del ancestro/raíz de la rama: "; cin >> nombreAncestro;
    Nodo* ancestro = buscarMiembro(raiz, nombreAncestro);

    if (ancestro == NULL) {
        cout << "El ancestro '" << nombreAncestro << "' no existe en el árbol." << endl;
        return;
    }

    bool pertenece = verificarPertenenciaARama(ancestro, nombreMiembro);

    cout << "\n=== RESULTADO ===" << endl;
    if (pertenece) {
        cout << "El miembro '" << nombreMiembro << "' SI pertenece";
        cout << " a la rama de '" << nombreAncestro << "'." << endl;
    } else {
        cout << "El miembro '" << nombreMiembro << "' NO pertenece";
        cout << " a la rama de '" << nombreAncestro << "'." << endl;
    }
}

// Muestra el menú principal del programa.
void mostrarMenu() {
    cout << "===== MENU ARBOL GENEALOGICO =====" << endl;
    cout << "1. Agregar miembro al arbol genealogico" << endl;
    cout << "2. Eliminar miembro del arbol genealogico" << endl;
    cout << "3. Buscar miembro" << endl;
    cout << "4. Visualizar el arbol (recorridos)" << endl;
    cout << "5. Mostrar ancestros" << endl;
    cout << "6. Mostrar descendientes" << endl;
    cout << "7. Verificar pertenencia a una rama" << endl;
    cout << "0. Salir" << endl;
    cout << "Seleccione una opción: ";
```

```
}

// Función principal (ahora al final)
int main() {
    setlocale(LC_CTYPE,"Spanish"); // Configura la consola para caracteres en español.
    Nodo* raiz = NULL;             // Puntero a la raíz del árbol genealógico.
    int opcion;                    // Variable para la opción del menú.

    cargarArbolDesdeArchivo(raiz); // Carga el árbol desde archivo si existe.

    do {
        mostrarMenu();             // Muestra el menú principal.
        cin >> opcion;             // Lee la opción del usuario.

        switch(opcion) {
            case 1:
                agregarMiembro(raiz); // Agrega un miembro.
                break;
            case 2:
                eliminarMiembroPrompt(raiz); // Elimina un miembro.
                break;
            case 3:
                buscarMiembroPrompt(raiz); // Busca un miembro.
                break;
            case 4:
                visualizarArbolCompleto(raiz); // Visualiza el árbol.
                break;
            case 5:
                mostrarAncestrosPrompt(raiz); // Muestra ancestros.
                break;
            case 6:
                mostrarDescendientesPrompt(raiz); // Muestra descendientes.
                break;
            case 7:
                verificarPertenenciaPrompt(raiz); // Verifica pertenencia a rama.
                break;
            case 0:
                guardarArbolEnArchivo(raiz); // Guarda el árbol antes de salir.
                cout << "Saliendo..." << endl;
                break;
            default:
                cout << "Opción inválida. Intente de nuevo." << endl;
        }
        cout << endl;
```

```
    } while(opcion != 0);

    return 0;
}
```

3. Evidencia de Funcionalidad

Agregar Raíz

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 1
Ingrese el nombre completo (sin espacios): JairLimasChagua
Ingrese el género: Masculino
Ingrese la relación familiar: Padre
Ingrese el nombre del padre/madre (escriba 'ninguno' si no tiene): ninguno
Miembro agregado como familiar extendido.

===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 4

=== ÁRBOL GENEALÓGICO ===
Leyenda:
+-- Miembro principal
    +-- Hijo/hermano
    +-- Último hijo/hermano

+-- JairLimasChagua (Masculino, Padre)
```

Agregar Hijos/Hermanos

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 1
Ingrese el nombre completo (sin espacios): Luis_Lara
Ingrese el género: Masculino
Ingrese la relación familiar: Padre/Hijo
Ingrese el nombre del padre/madre (escriba 'ninguno' si no tiene): JairLimasChagua
Miembro agregado correctamente.

===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 4

=== ÁRBOL GENEALÓGICO ===
Leyenda:
+-- Miembro principal
    +-- Hijo/hermano
    +-- Último hijo/hermano

+-- JairLimasChagua (Masculino, Padre)
    +-- Luis_Lara (Masculino, Padre/Hijo)
```

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 1
Ingrese el nombre completo (sin espacios): Valeri_Flores
Ingrese el género: Femenino
Ingrese la relación familiar: Madre/Hija
Ingrese el nombre del padre/madre (escriba 'ninguno' si no tiene): JairLimasChagua
Miembro agregado correctamente.

===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 4

=== ÁRBOL GENEALÓGICO ===
Leyenda:
+-- Miembro principal
    +-- Hijo/hermano
    +-- Último hijo/hermano
```

Eliminar Miembro

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 2
Ingrese el nombre completo del miembro a eliminar: ValeiFlores
Miembro no encontrado en el árbol o no se pudo eliminar.
```

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 2
Ingrese el nombre completo del miembro a eliminar: Valeri_Flores
Miembro eliminado correctamente.
```

```
=== ÁRBOL GENEALÓGICO ===
Leyenda:
+-- Miembro principal
    +-- Hijo/hermano
    +-- Último hijo/hermano

+-- JairLimasChagua (Masculino, Padre)
    +-- Luis_Lara (Masculino, Padre/Hijo)
```

Buscar Miembro

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 3
Ingrese el nombre completo del miembro a buscar: Valeri_Flores
Miembro no encontrado en el árbol.
```

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 3
Ingrese el nombre completo del miembro a buscar: Luis_Lara

=== Miembro Encontrado ===
Nombre: Luis_Lara
Género: Masculino
Relación: Padre/Hijo
Padre/Madre: JairLimasChagua
```

Visualizar Árbol

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 4

=== ÁRBOL GENEALÓGICO ===
Leyenda:
+-- Miembro principal
    +-- Hijo/hermano
    +-- Último hijo/hermano

+-- JairLimasChagua (Masculino, Padre)
    +-- Luis_Lara (Masculino, Padre/Hijo)
```

Mostrar Ancestros

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 5
Ingrese el nombre completo del miembro para ver sus ancestros: Luis_Lara

=== Ancestros de Luis_Lara ===
JairLimasChagua
```

Mostrar Descendientes

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 6
Ingrese el nombre completo del miembro para ver sus descendientes: Luis_Lara

=== Descendientes de Luis_Lara ===
No tiene descendientes.
```

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 6
Ingrese el nombre completo del miembro para ver sus descendientes: JairLimasChagua

=== Descendientes de JairLimasChagua ===
- Luis_Lara (Padre/Hijo)
```

Verificar pertenencia a una rama

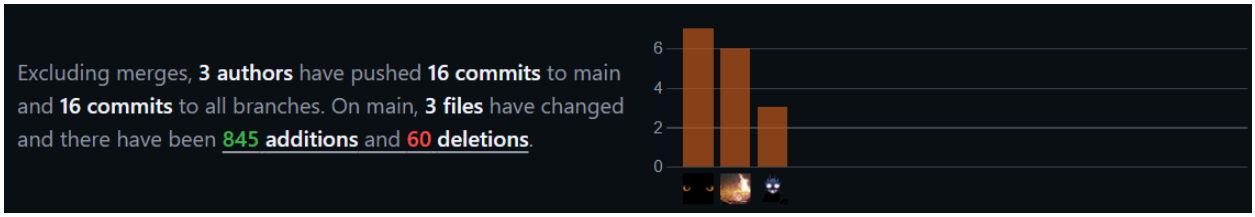
```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 7
Ingrese el nombre del miembro a verificar: Luis_Lara
Ingrese el nombre del ancestro/raíz de la rama: JairLimasChagua

=== RESULTADO ===
El miembro 'Luis_Lara' SI pertenece a la rama de 'JairLimasChagua'.
```

```
===== MENU ARBOL GENEALOGICO =====
1. Agregar miembro al arbol genealogico
2. Eliminar miembro del arbol genealogico
3. Buscar miembro
4. Visualizar el arbol (recorridos)
5. Mostrar ancestros
6. Mostrar descendientes
7. Verificar pertenencia a una rama
0. Salir
Seleccione una opción: 7
Ingrese el nombre del miembro a verificar: Valeri_Flores
Ingrese el nombre del ancestro/raíz de la rama: JairLimasChagua

=== RESULTADO ===
El miembro 'Valeri_Flores' NO pertenece a la rama de 'JairLimasChagua'.
```

4. Evidencia por Integrante



Quispe Linares Antoni

```cpp
#include <iostream> // Incluye la biblioteca estándar para entrada y salida.
#include <string>   // Incluye la biblioteca para manejar cadenas de texto.
#include <fstream>  // Incluye la biblioteca para manejo de archivos.
#include <sstream>  // Incluye la biblioteca para manipulación de strings como streams.
using namespace std; // Usa el espacio de nombres estándar para evitar escribir std::

// Estructura para un nodo del arbol genealogico
struct Nodo {
    string nombreCompleto;        // Nombre completo del miembro.
    string genero;                // Género del miembro.
    string relacionFamiliar;      // Relación familiar (hijo, padre, etc).
    Nodo* padre;                  // Puntero al padre.
    Nodo* hijo;                   // Puntero al primer hijo.
    Nodo* hermano;                // Puntero al siguiente hermano.

    // Constructor del nodo, inicializa los campos y punteros.
    Nodo(string nombre, string gen, string relacion, Nodo* p = NULL)
        : nombreCompleto(nombre), genero(gen), relacionFamiliar(relacion), padre(p), hijo(NULL), hermano(NULL) {}
};
```

```
void agregarMiembro(Nodo*& raiz) {
    string nombre, genero, relacion, nombrePadre;

    cout << "Ingrese el nombre completo (sin espacios): "; cin >> nombre;
    cout << "Ingrese el género: "; cin >> genero;
    cout << "Ingrese la relación familiar: "; cin >> relacion;

    if (raiz == NULL) {
        raiz = new Nodo(nombre, genero, relacion);
        cout << "Miembro agregado como raíz del árbol." << endl;
        return;
    }

    cout << "Ingrese el nombre del padre/madre (escriba 'ninguno' si no tiene): "; cin >> nombrePadre;

    if (nombrePadre == "ninguno") {
        Nodo* nuevo = new Nodo(nombre, genero, relacion);
        Nodo* temp = raiz;
        while (temp->hermano != NULL) {
            temp = temp->hermano;
        }
        temp->hermano = nuevo;
        cout << "Miembro agregado como familiar extendido." << endl;
    } else {
        Nodo* padre = buscarMiembro(raiz, nombrePadre);
        if (padre == NULL) {
            cout << "Padre/Madre no encontrado. Miembro no agregado." << endl;
            return;
        }

        Nodo* nuevo = new Nodo(nombre, genero, relacion, padre);
```

```
        Nodo* nuevo = new Nodo(nombre, genero, relacion, padre);

        if (padre->hijo == NULL) {
            padre->hijo = nuevo;
        } else {
            Nodo* temp = padre->hijo;
            while (temp->hermano != NULL) {
                temp = temp->hermano;
            }
            temp->hermano = nuevo;
        }
        cout << "Miembro agregado correctamente." << endl;
    }
}
```

```cpp
// Busca un miembro por nombre en el árbol, recursivamente.
Nodo* buscarMiembro(Nodo* raiz, const string& nombre) {
    if (raiz == NULL) return NULL; // Si el nodo es nulo, retorna NULL.
    if (raiz->nombreCompleto == nombre) // Si el nombre coincide, retorna el nodo.
        return raiz;

    Nodo* encontrado = buscarMiembro(raiz->hijo, nombre); // Busca en los hijos.
    if (encontrado != NULL) return encontrado; // Si lo encuentra en los hijos, retorna el nodo.

    return buscarMiembro(raiz->hermano, nombre); // Si no, busca en los hermanos.
}
```

Limas Chagua Jair

```cpp
    // Recorrido postorden del árbol (hijos, nodo, hermanos).
    void recorridoPostorden(Nodo* raiz) {
        if (raiz == NULL) return;
        recorridoPostorden(raiz->hijo);
        cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " << raiz->genero << ")" << endl;
        recorridoPostorden(raiz->hermano);
    }

```

```cpp
    // Visualiza el árbol completo con leyenda.
    void visualizarArbolCompleto(Nodo* raiz) {
        if (raiz == NULL) {
            cout << "El árbol genealógico está vacío." << endl;
            return;
        }

        cout << "\n=== ÁRBOL GENEALÓGICO ===\n";
        cout << "Leyenda:\n";
        cout << "+-- Miembro principal\n";
        cout << "    +-- Hijo/hermano\n";
        cout << "    +-- Último hijo/hermano\n\n";

        visualizarArbol(raiz);

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

Solis Sedano Anderson

```cpp
void visualizarArbol(Nodo* nodo, string prefijo = "", bool esUltimo = true) {
    if (nodo == NULL) return; // Si el nodo es nulo, termina.

    cout << prefijo; // Imprime el prefijo para la jerarquía.

    // Imprime el símbolo de rama.
    if (esUltimo) {
        cout << "+-- ";
    } else {
        cout << "+-- ";
    }

    cout << nodo->nombreCompleto << " (" << nodo->genero << ", " << nodo->relacionFamiliar << ")" << endl; // Muestra datos del nodo.

    string nuevoPrefijo;
    if (esUltimo) {
        nuevoPrefijo = prefijo + "    "; // Espaciado para el último hijo.
    } else {
        nuevoPrefijo = prefijo + "¦   "; // Espaciado para otros hijos.
    }

    Nodo* hijo = nodo->hijo; // Empieza con el primer hijo.
    while (hijo != NULL) {
        bool ultimoHijo = (hijo->hermano == NULL); // Verifica si es el último hijo.
        visualizarArbol(hijo, nuevoPrefijo, ultimoHijo); // Llama recursivamente.
        hijo = hijo->hermano; // Pasa al siguiente hermano.
    }
}

// Recorrido inorden del árbol (hijos, nodo, hermanos).
```

```cpp
// Recorrido preorden del árbol (nodo, hijos, hermanos).
void recorridoPreorden(Nodo* raiz) {
    if (raiz == NULL) return;
    cout << raiz->nombreCompleto << " (" << raiz->relacionFamiliar << ", " << raiz->genero << ")" << endl;
    recorridoPreorden(raiz->hijo);
    recorridoPreorden(raiz->hermano);
}
```

Castro Solano Anthony

```
351     // Solicita el nombre y busca un miembro, mostrando sus datos.
352     void buscarMiembroPrompt(Nodo* raiz) {
353         string nombre;
354         cout << "Ingrese el nombre completo del miembro a buscar: "; cin >> nombre;
355 
356         Nodo* encontrado = buscarMiembro(raiz, nombre);
357         if (encontrado != NULL) {
358             cout << "\n=== Miembro Encontrado ===" << endl;
359             cout << "Nombre: " << encontrado->nombreCompleto << endl;
360             cout << "Género: " << encontrado->genero << endl;
361             cout << "Relación: " << encontrado->relacionFamiliar << endl;
362             if (encontrado->padre != NULL) {
363                 cout << "Padre/Madre: " << encontrado->padre->nombreCompleto << endl;
364             }
365         } else {
366             cout << "Miembro no encontrado en el árbol." << endl;
367         }
368     }
369 
```

```
370     // Solicita el nombre y muestra los ancestros del miembro.
371     void mostrarAncestrosPrompt(Nodo* raiz) {
372         string nombre;
373         cout << "Ingrese el nombre completo del miembro para ver sus ancestros: "; cin >> nombre;
374 
375         Nodo* miembro = buscarMiembro(raiz, nombre);
376         if (miembro == NULL) {
377             cout << "Miembro no encontrado." << endl;
378             return;
379         }
380 
381         cout << "\n=== Ancestros de " << nombre << " ===" << endl;
382         if (miembro->padre == NULL) {
383             cout << "No tiene ancestros registrados (es la raíz del árbol)." << endl;
384         } else {
385             mostrarAncestros(miembro);
386             cout << endl;
387         }
388     }
```