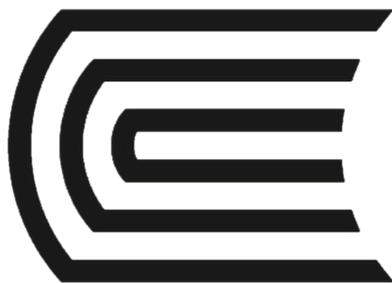


“Año de la recuperación y consolidación de la economía peruana”



**ESTRUCTURA DE DATOS
INFORME DE ABR**

Estudiantes:

LIMAYMANTA CASTRO, Dayana Evelin

NAPAN HERNÁNDEZ, Luisana Carolina

VALDIVIA BLAS, Yatzuri Xiomara

YAURI ANCCASI, Rosa

Docente:

OSORIO CONTRERAS, Rosario Delia

NRC: 29902

HUANCAYO - PERÚ

2025 - 10

ÍNDICE

Capítulo 1 – Fase de Ideación.....	3
1.1 Descripción del problema.....	3
1.2 Requerimientos del sistema.....	3
1.3 Respuesta a las preguntas guías.....	3
1.4 Herramienta Colaborativa.....	4
Capítulo 2 – Prototipo.....	7
2.1 Descripción de estructuras de datos y operaciones:.....	7
2.2. Algoritmos principales:.....	8
2.3 Diagramas de Flujo.....	12
2.4 Avance del código fuente.....	14
Capítulo 3 – Solución Final.....	18
3.1 Código limpio, bien comentado y estructurado.....	18
3.2 Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos.....	18
3.3 Repositorio con Control de Versiones (Capturas de Pantalla).....	18

CAPÍTULO 1 – FASE DE IDEACIÓN

1.1 Descripción del problema

Durante una expedición arqueológica, un grupo de investigadores descubre los restos de una antigua civilización previamente desconocida. Entre los hallazgos se encuentran inscripciones que documentan de forma detallada las relaciones familiares entre los miembros de esta civilización. El reto consiste en diseñar una solución computacional que permita organizar esta información como un árbol genealógico, facilitando consultas sobre ancestros, descendientes y relaciones familiares.

Para abordar esta problemática, se propone el uso de una estructura de datos Árbol Binario de Búsqueda (ABB), que permite insertar, eliminar y consultar miembros de manera eficiente, a la vez que mantiene la integridad de la genealogía descubierta.

1.2 Requerimientos del sistema

Requerimientos Funcionales

- El sistema debe permitir insertar nuevos miembros al árbol genealógico.
- Debe ser posible eliminar miembros sin comprometer la estructura del árbol.
- El sistema debe permitir consultar ancestros y descendientes de un miembro.
- Se debe poder visualizar el árbol genealógico de forma jerárquica.
- El sistema debe poder determinar si un miembro pertenece a una rama específica.
- Se debe garantizar que las operaciones se realicen de manera eficiente (tiempo logarítmico en promedio).

Requerimientos No Funcionales

- La aplicación debe ser intuitiva y fácil de usar.
- El sistema debe ser modular y escalable, permitiendo la adición de nuevas funcionalidades.
- El código debe estar documentado y seguir buenas prácticas de programación.
- Debe funcionar correctamente con árboles de gran profundidad y número de nodos.
- El rendimiento debe mantenerse óptimo incluso con grandes volúmenes de datos.

1.3 Respuesta a las preguntas guías

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

Cada nodo del árbol genealógico debe contener la información principal de una persona. Esto incluye: un identificador único (como el nombre completo o un ID numérico), la relación con otros miembros (por ejemplo, padre o madre), el año de nacimiento y, si es necesario, el año de fallecimiento.

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

→ Inserción:

Se busca el lugar correcto siguiendo las reglas del árbol (por ejemplo, en un árbol

binario de búsqueda, comparando valores) y se añade el nuevo nodo como hijo en ese lugar vacío, manteniendo el orden y la estructura.

→ **Eliminación:**

Depende del caso:

- ◆ Si el nodo es hoja (sin hijos), se elimina directamente.
- ◆ Si tiene un solo hijo, se conecta ese hijo con el padre del nodo eliminado.
- ◆ Si tiene dos hijos, se reemplaza el nodo con el sucesor inorden (el valor más pequeño del subárbol derecho) y luego se elimina ese sucesor, manteniendo el orden.

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

Los métodos principales son:

- Preorden: Visita primero el nodo actual (antepasado), luego sus hijos.
- Inorden: Visita primero el hijo izquierdo, luego el nodo, y después el hijo derecho (útil para obtener datos ordenados en árboles binarios).
- Postorden: Visita primero los hijos y al final el nodo actual.
- Por niveles (anchura): Recorre el árbol generación por generación, mostrando claramente la genealogía.

El recorrido por niveles es el más intuitivo para visualizar generaciones.

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

Se realiza una búsqueda dentro del subárbol que representa esa rama:

- Se comienza en el nodo raíz de la rama.
- Se recorren sus nodos hijos (usando búsqueda en profundidad o en anchura).
- Se compara el dato de cada nodo con el miembro buscado.
- Si se encuentra, el miembro pertenece a esa rama; si no, no pertenece.

Este proceso usa algoritmos de búsqueda en árboles para confirmar la pertenencia

• **¿Cómo balancear el árbol si se vuelve demasiado profundo?**

Para evitar que el árbol sea muy profundo (lo que afecta el rendimiento), se usan árboles平衡ados, que aplican rotaciones o reorganizaciones automáticas al insertar o eliminar nodos, por ejemplo:

- Árboles AVL: Mantienen el balance calculando la altura de subárboles y rotando nodos para equilibrar.
- Árboles rojo-negro: Usan reglas de color para mantener el balance.
- Árboles B y B+: Usados en bases de datos, mantienen múltiples hijos y balancean para evitar profundidad excesiva.

Estas técnicas garantizan que el árbol mantenga una altura mínima, mejorando la eficiencia en búsquedas, inserciones y eliminaciones.

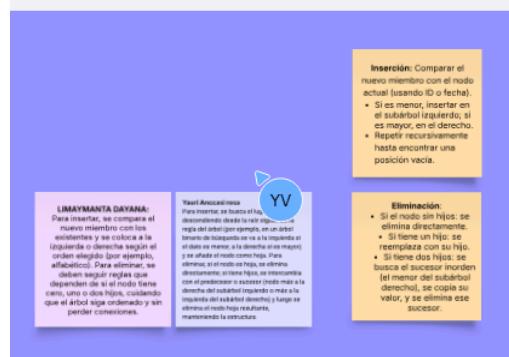
1.4 Herramienta Colaborativa

Enlace de la herramienta colaborativa utilizada:

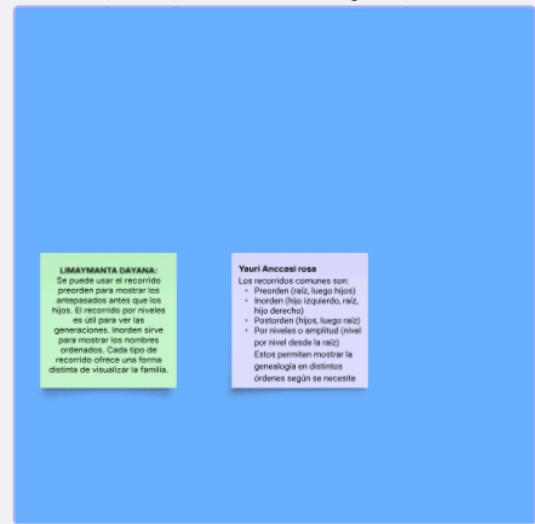
https://lucid.app/lucidspark/de6a8032-fd08-4d3b-9388-30613796143b/edit?view_items=ZYlo5D-j9d_c&invitationId=inv_3c0eed1e-a3b2-4bba-8d33-14b5b3cdf241

Capturas de pantalla:

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



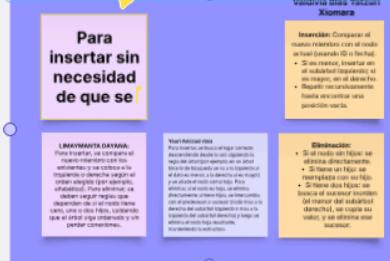
¿Qué métodos permiten recorrer el árbol para



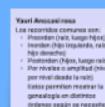
¿Qué información se debe acercar en cada nodo del árbol?

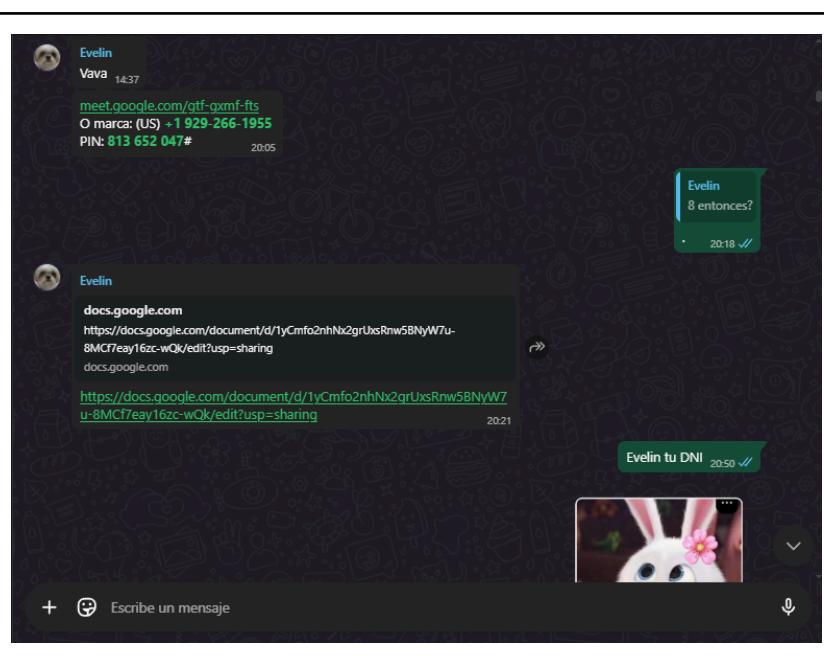
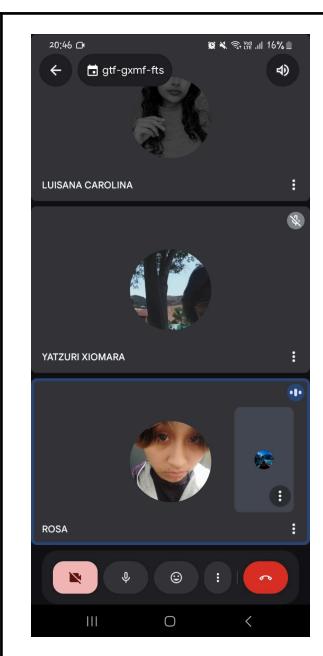
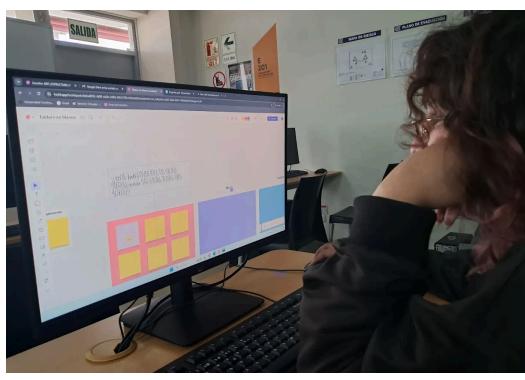
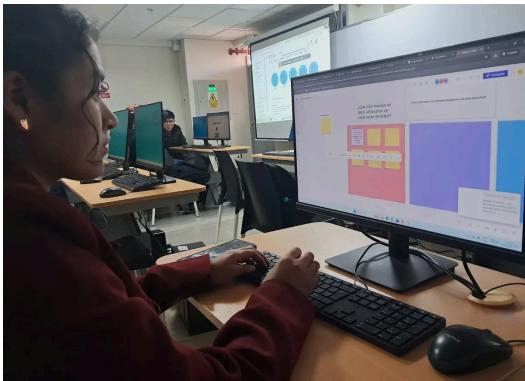
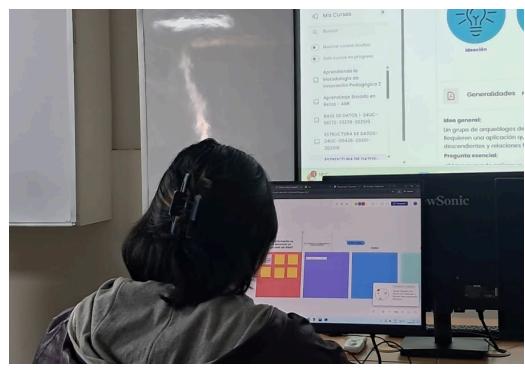


¿Cómo insertar y eliminar miembros del árbol sin romper su estructura?



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





CAPÍTULO 2 – PROTOTIPO

2.1 Descripción de estructuras de datos y operaciones:

Estructura de Datos: Árbol Binario de Búsqueda (ABB)

Para representar el árbol genealógico de la civilización descubierta, se utilizará un Árbol Binario de Búsqueda (ABB), ya que permite insertar, buscar y eliminar miembros de manera eficiente, manteniendo la estructura ordenada.

Información que almacena cada nodo:

Cada nodo del árbol representará a una persona y contendrá los siguientes datos:

- ID (número entero): identificador único.
- Nombre (texto): nombre completo de la persona.
- Año de nacimiento (entero).
- Estado: si sigue vivo o ya falleció.
- Puntero izquierdo: hijo con valor menor.
- Puntero derecho: hijo con valor mayor.

Operaciones principales:

- Insertar miembro: se compara con los nodos existentes hasta encontrar su posición correcta según el ABB.
- Eliminar miembro: se manejan tres casos: sin hijos, con un hijo, o con dos hijos (reemplazo por sucesor inorden).
- Buscar miembro: se recorre el árbol comparando el ID o nombre.
- Recorrer árbol:
 - *Inorden*: muestra los miembros en orden.
 - *Preorden y Postorden*: permiten ver genealogía de forma jerárquica.
 - *Por niveles* (opcional): muestra generaciones completas.
- Consultar descendientes: se recorre hacia abajo desde el nodo.
- Verificar si pertenece a una rama: se busca en el subárbol correspondiente.

2.2. Algoritmos principales:

- Pseudocódigo para crear un árbol binario.

```

Proceso ArbolGenealogico
    Dimension nombres[100], ids[100], nacimientos[100], fallecimientos[100]
    Dimension padre[100], izqs[100], ders[100]
    Definir total, opcion, i, idRelacionado, posRelativo, relacion, idBuscar Como Entero
    total ← 0

    Repetir
        Escribir "--- MENÚ ÁRBOL GENEALÓGICO ---"
        Escribir "1. Insertar miembro"
        Escribir "2. Buscar miembro"
        Escribir "3. Salir"
        Escribir "Ingrese una opción:"
        Leer opcion

        Segun opcion Hacer
            1:
                Si total ≥ 100 Entonces
                    Escribir "Límite máximo alcanzado."
                Sino
                    Escribir "Ingrese ID:"
                    Leer ids[total]
                    Escribir "Ingrese nombre:"
                    Leer nombres[total]
                    Escribir "Año de nacimiento:"
                    Leer nacimientos[total]
                    Escribir "Fallecimiento (o escriba Vivo):"
                    Leer fallecimientos[total]

                Si total = 0 Entonces
                    padre[total] ← -1
                    izqs[total] ← -1
                    ders[total] ← -1
                    Escribir "Miembro insertado como raiz del árbol."
                Sino
                    Escribir "Ingrese el ID del familiar relacionado:"
                    Leer idRelacionado
                    posRelativo ← -1

                    Para i ← 0 Hasta total - 1 Con Paso 1
                        Si ids[i] = idRelacionado Entonces
                            posRelativo ← i
                        FinSi
                    FinPara

```

```

        Si posRelativo = -1 Entonces
            Escribir "Familiar no encontrado."
        Sino
            Escribir "¿Qué relación tiene con ", nombres[posRelativo], "?"
            Escribir "1. Padre/Madre"
            Escribir "2. Hijo izquierdo"
            Escribir "3. Hijo derecho"
            Escribir "4. Hermano izquierdo"
            Escribir "5. Hermano derecho"
            Leer relacion

        Segun relacion Hacer
            1:
                padre[total] ← padre[posRelativo]
                izqs[total] ← posRelativo
                padre[posRelativo] ← total
                Escribir "Insertado como padre/madre."
            2:
                Si izqs[posRelativo] = -1 Entonces
                    izqs[posRelativo] ← total
                    padre[total] ← posRelativo
                    Escribir "Insertado como hijo izquierdo."
                Sino
                    Escribir "Ya tiene hijo izquierdo."
                FinSi
            3:
                Si ders[posRelativo] = -1 Entonces
                    ders[posRelativo] ← total
                    padre[total] ← posRelativo
                    Escribir "Insertado como hijo derecho."
                Sino
                    Escribir "Ya tiene hijo derecho."
                FinSi
            4:
                Si padre[posRelativo] ≠ -1 Y izqs[padre[posRelativo]] = posRelativo Entonces
                    izqs[padre[posRelativo]] ← total
                    padre[total] ← padre[posRelativo]
                    Escribir "Insertado como hermano izquierdo."
                Sino
                    Escribir "No se puede insertar como hermano izquierdo."
                FinSi
            5:
                Si padre[posRelativo] ≠ -1 Y ders[padre[posRelativo]] = posRelativo Entonces
                    ders[padre[posRelativo]] ← total
                    padre[total] ← padre[posRelativo]
                    Escribir "Insertado como hermano derecho."
                Sino
                    Escribir "No se puede insertar como hermano derecho."
                -- --
            De Otro Modo:
                Escribir "Relación no válida."
            FinSegun
        FinSi
        total ← total + 1
    FinSi

2:
    Escribir "Ingrese ID a buscar:"
    Leer idBuscar
    posRelativo ← -1

    Para i ← 0 Hasta total - 1 Con Paso 1
        Si ids[i] = idBuscar Entonces
            posRelativo ← i
        FinSi
    FinPara

    Si posRelativo ≠ -1 Entonces
        Escribir "Miembro encontrado: ", nombres[posRelativo]
        Escribir "Nacimiento: ", nacimientos[posRelativo]
        Escribir "Fallecimiento: ", fallecimientos[posRelativo]
    Sino
        Escribir "No encontrado."
    FinSi

3:
    Escribir "Saliendo del programa..."

De Otro Modo:
    Escribir "Opción no válida."
FinSegun
Hasta Que opcion = 3
FinProceso

```

- Pseudocódigo para realizar el recorrido de un árbol.

```

1  Proceso ArbolGenealogico
2      Dimension nombres[20]
3      Dimension relaciones[20]
4      Dimension nacimiento[20]
5      Dimension fallecimiento[20]
6      Dimension ids[20]
7
8      Definir opcion, opc, total, i, nivel Como Entero
9      Definir idBuscar Como Cadena
10     Definir encontrado Como Logico
11
12     total ← 0
13
14     Repetir
15         Limpiar Pantalla
16         Escribir "*** MENU PRINCIPAL ***"
17         Escribir "1. Agregar familiar"
18         Escribir "2. Ver recorridos"
19         Escribir "3. Buscar por ID"
20         Escribir "4. Salir"
21         Escribir "Seleccione una opcion: "
22         Leer opcion
23
24         Segun opcion Hacer
25             1: // Agregar familiar
26                 Si total < 20 Entonces
27                     total ← total + 1
28                     Escribir "Ingrese ID (DNI): "
29                     Leer ids[total]
30                     Escribir "Ingrese nombre: "
31                     Leer nombres[total]
32                     Escribir "Ingrese relacion (Abuelo, Padre, Madre, Hijo, Nieto, etc.): "
33                     Leer relaciones[total]
34                     Escribir "Año de nacimiento: "
35                     Leer nacimiento[total]
36                     Escribir "Año de fallecimiento o Vivo: "
37                     Leer fallecimiento[total]
38                 SiNo
39                     Escribir "Limite de familiares alcanzado."
40                     Esperar Tecla
41                 FinSi
42
43             2: // Ver recorridos
44                 Si total = 0 Entonces
45                     Escribir "No hay familiares registrados."
46                     Esperar Tecla
47                 SiNo
48                     Escribir "*** SUBMENÚ DE RECORRIDOS ***"
49                     Escribir "1. Preorden"
50                     Escribir "2. Inorden"
51                     Escribir "3. Postorden"
52                     Escribir "4. Recorrido por niveles (simulado)"
53                     Escribir "Seleccione una opción: "
54                     Leer opc

```

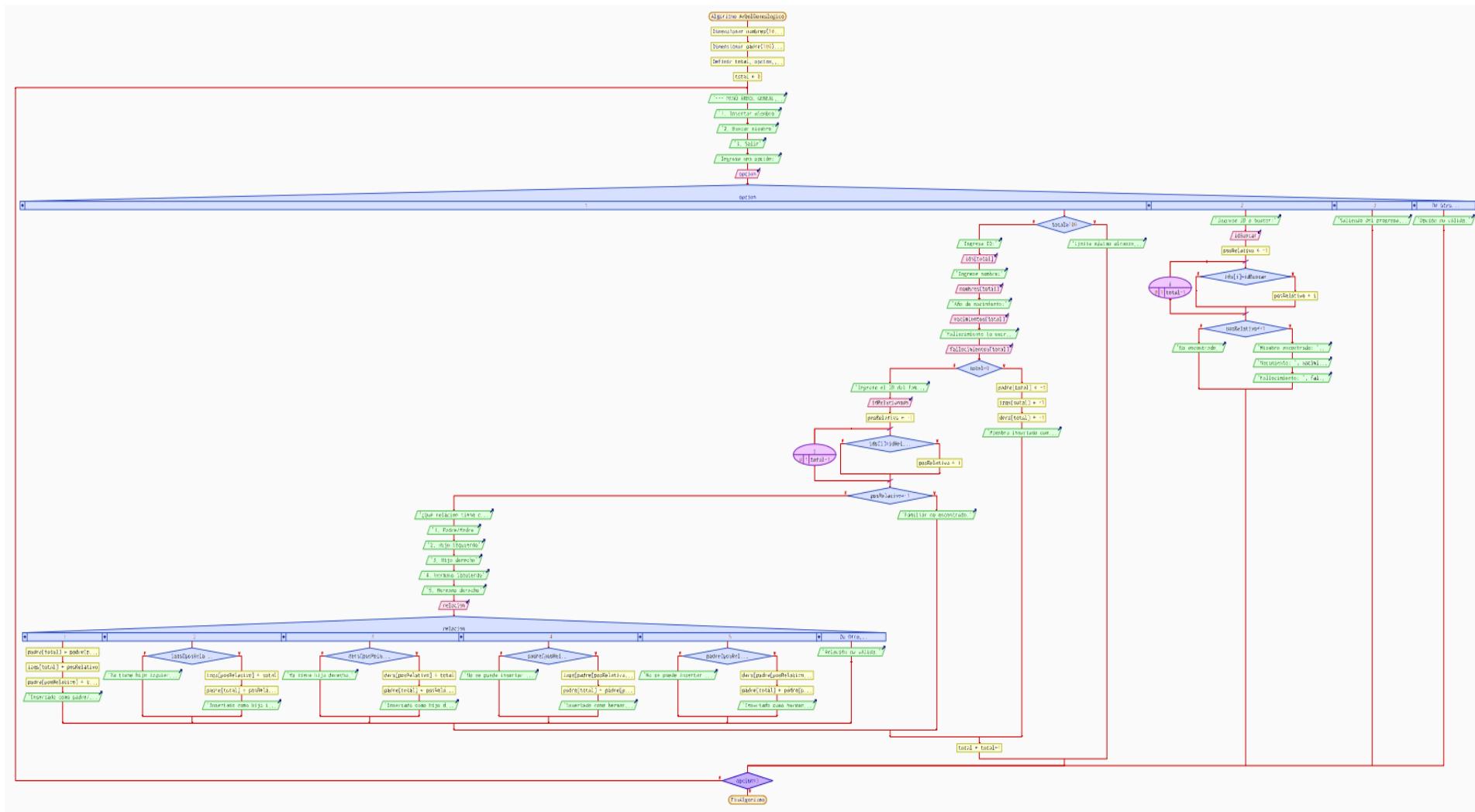
```

56
57
58
59
60 +   Segun opc Hacer
61
62     1:    Escribir "*** Recorrido Preorden (simulado) ***"
63     Para i ← 1 Hasta total
64         Escribir nombres[i], " (", relaciones[i], ") - Nac.: ", nacimiento[i]
65     FinPara
66
67     2:    Escribir "*** Recorrido Inorden (simulado) ***"
68     Para i ← 1 Hasta total
69         Escribir relaciones[i], " -> ", nombres[i]
70     FinPara
71
72     3:    Escribir "*** Recorrido Postorden (simulado) ***"
73     Para i ← total Hasta 1 Con Paso -1
74         Escribir nombres[i], " (", relaciones[i], ")"
75     FinPara
76
77     4:    Escribir "*** Recorrido por niveles (simulado según relación) ***"
78     Para i ← 1 Hasta total
79         nivel ← ObtenerNivel(relaciones[i])
80         Escribir "Nivel ", nivel, ": ", nombres[i], " (", relaciones[i], ")"
81     FinPara
82     De Otro Modo:
83         Escribir "Opción no válida."
84     FinSegun
85     Esperar Tecla
86     FinSi
87
88     3: // Buscar por ID
89         Escribir "Función de búsqueda aún no implementada."
90         Esperar Tecla
91
92     4:    Escribir "Saliendo del programa..."
93     De Otro Modo:
94         Escribir "Opción no válida."
95         Esperar Tecla
96     FinSegun
97     Hasta Que opcion = 4
98 FinProceso
99
100 Funcion nivelRelacion ← ObtenerNivel(relacion)
101     Segun Mayusculas(relacion) Hacer
102         "HIJO":
103         "HIJA":
104             nivelRelacion ← 1 // Hijos del abuelo
105         "NIETO":
106         "NIETA":
107             nivelRelacion ← 2 // Nietos del abuelo
108         "BISNIETO":
109         "BISNIETA":
110             nivelRelacion ← 3
111         "PADRE":
112         "MADRE":
113             nivelRelacion ← -1 // Padres del abuelo
114         "ABUELO":
115         "ABUELA":
116             nivelRelacion ← 0 // El mismo abuelo (raíz)
117         De Otro Modo:
118             nivelRelacion ← 99 // Relación desconocida
119     FinSegun
120 FinFuncion

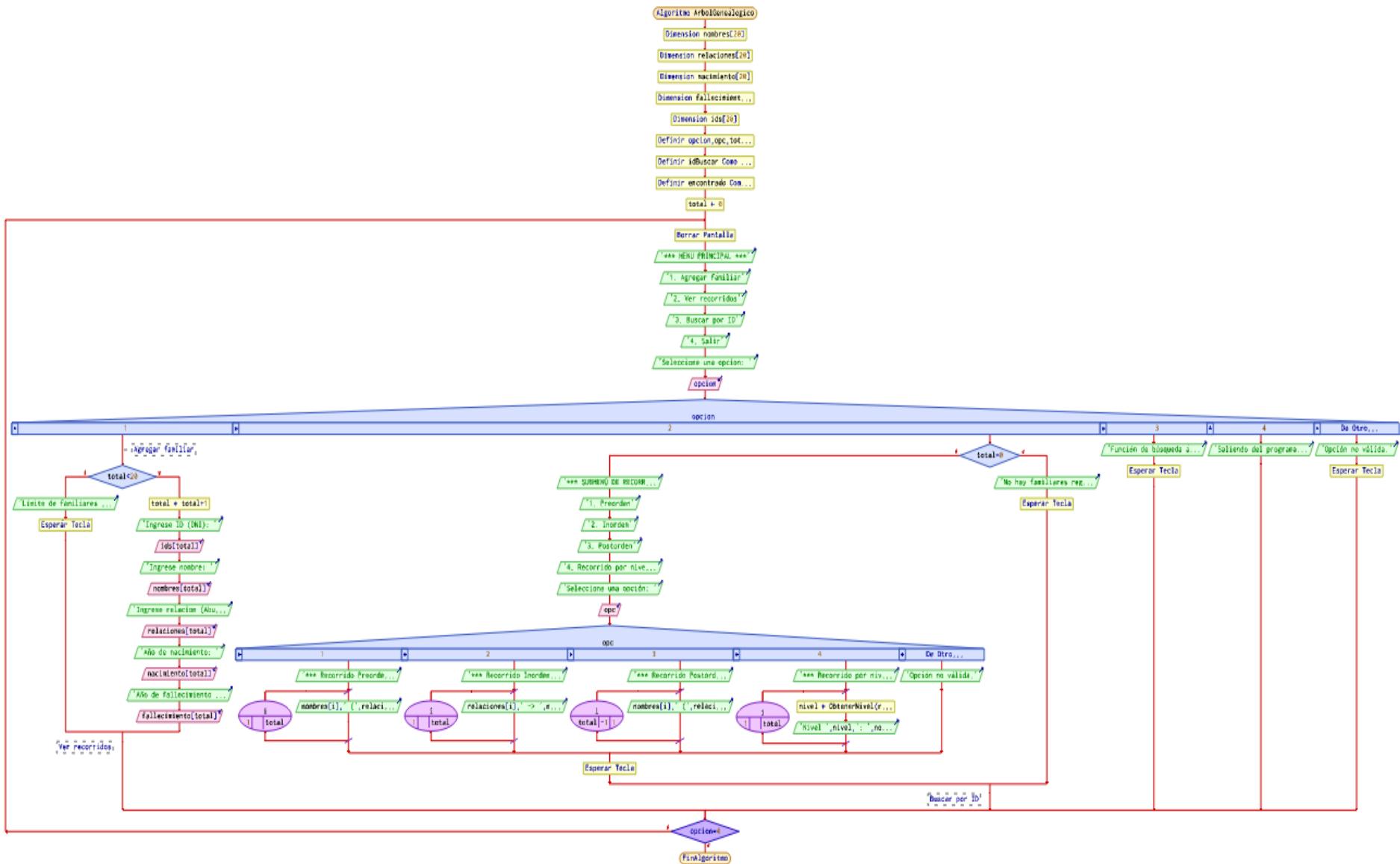
```

2.3 Diagramas de Flujo

- Diagrama para crear un árbol binario



- Diagrama para realizar el recorrido de un árbol.



2.4 Avance del código fuente

<https://github.com/LuisanaNapan/ArbolGenealogico.git>

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  struct Nodo {
6      int id;
7      string nombre;
8      int nacimiento;
9      string fallecimiento;
10     Nodo* izq;
11     Nodo* der;
12     Nodo* padre;
13 };
14
15 Nodo* raiz = NULL;
16
17 // Crear nuevo nodo
18 Nodo* crearNodo(int id, string nombre, int nacimiento, string fallecimiento) {
19     Nodo* nuevo = new Nodo;
20
21     nuevo->id = id;
22     nuevo->nombre = nombre;
23     nuevo->nacimiento = nacimiento;
24     nuevo->fallecimiento = fallecimiento;
25     nuevo->izq = nuevo->der = nuevo->padre = NULL;
26     return nuevo;
27 }
28
29 // Insertar miembro
30 void insertar(Nodo*& actual, Nodo* nuevo, Nodo* padre = NULL) {
31     if (actual == NULL) {
32         nuevo->padre = padre;
33         actual = nuevo;
34         cout << "Miembro insertado correctamente.\n";
35     } else if (nuevo->id < actual->id) {
36         insertar(actual->izq, nuevo, actual);
37     } else {
38         insertar(actual->der, nuevo, actual);
39     }
40 }
41 // Buscar miembro por ID
42 Nodo* buscar(Nodo* actual, int id) {
```

```

43     if (actual == NULL || actual->id == id) return actual;
44     return (id < actual->id) ? buscar(actual->izq, id) : buscar(actual->der, id);
45 }
46
47 // Mínimo del subárbol
48 Nodo* minimo(Nodo* actual) {
49     while (actual && actual->izq != NULL)
50         actual = actual->izq;
51     return actual;
52 }
53
54 // Eliminar nodo
55 Nodo* eliminar(Nodo* raiz, int id) {
56     if (!raiz) return NULL;
57     if (id < raiz->id) {
58         raiz->izq = eliminar(raiz->izq, id);
59     } else if (id > raiz->id) {
60         raiz->der = eliminar(raiz->der, id);
61     } else {
62         // Nodo encontrado
63         if (!raiz->izq) {
64             Nodo* temp = raiz->der;
65             delete raiz;
66             return temp;
67         } else if (!raiz->der) {
68             Nodo* temp = raiz->izq;
69             delete raiz;
70             return temp;
71         } else {
72             Nodo* sucesor = minimo(raiz->der);
73             raiz->id = sucesor->id;
74             raiz->nombre = sucesor->nombre;
75             raiz->nacimiento = sucesor->nacimiento;
76             raiz->fallecimiento = sucesor->fallecimiento;
77             raiz->der = eliminar(raiz->der, sucesor->id);
78         }
79     }
80     return raiz;
81 }
82
83 // Recorridos
84 void inorder(Nodo* nodo) {
85     if (!nodo) return;
86     inorder(nodo->izq);
87     cout << nodo->nombre << "(" << nodo->id << ")\n";
88     inorder(nodo->der);
89 }
90
91 void preorden(Nodo* nodo) {
92     if (!nodo) return;
93     cout << nodo->nombre << "(" << nodo->id << ")\n";
94     preorden(nodo->izq);
95     preorden(nodo->der);
96 }
97
98 void postorden(Nodo* nodo) {
99     if (!nodo) return;
100    postorden(nodo->izq);
101    postorden(nodo->der);
102    cout << nodo->nombre << "(" << nodo->id << ")\n";
103 }
104
105 void porNiveles(Nodo* raiz) {
106     if (raiz == NULL) return;

```

```

108     const int MAX = 100;
109     Nodo* cola[MAX];
110     int inicio = 0, fin = 0;
111
112     // Encolar raíz
113     cola[fin++] = raiz;
114
115     while (inicio < fin) {
116         Nodo* actual = cola[inicio++]; // Desencolar
117
118         cout << actual->nombre << " (" << actual->id << ")\n";
119
120         // Encolar hijos
121         if (actual->izq != NULL) {
122             cola[fin++] = actual->izq;
123         }
124         if (actual->der != NULL) {
125             cola[fin++] = actual->der;
126         }
127     }
128 }
129
130 // Ancestros
131 void mostrarAncestros(Nodo* nodo) {
132     while (nodo) {
133         cout << nodo->nombre << " (" << nodo->id << ")\n";
134         nodo = nodo->padre;
135     }
136 }
137
138 // Descendientes (en preorden)
139 void mostrarDescendientes(Nodo* nodo) {
140     if (!nodo) return;
141     cout << nodo->nombre << " (" << nodo->id << ")\n";
142     mostrarDescendientes(nodo->izq);
143     mostrarDescendientes(nodo->der);
144 }
145
146 // Menú principal
147 int main() {
148     int opcion;
149     do {
150         cout << "\n---- MENU ÁRBOL GENEALÓGICO ----\n";
151         cout << "1. Insertar miembro\n";
152         cout << "2. Buscar miembro\n";
153         cout << "3. Eliminar miembro\n";
154         cout << "4. Consultar ancestros\n";
155         cout << "5. Consultar descendientes\n";
156         cout << "6. Recorridos\n";
157         cout << "7. Salir\n";
158         cout << "Seleccione una opción: ";
159         cin >> opcion;

```

```

161         if (opcion == 1) {
162             int id, nac;
163             string nombre, fallecido;
164             cout << "ID (DNI): "; cin >> id;
165             cin.ignore();
166             cout << "Nombre: "; getline(cin, nombre);
167             cout << "Año de nacimiento: "; cin >> nac;
168             cin.ignore();
169             cout << "Fallecimiento (o 'Vivo'): "; getline(cin, fallecido);
170             Nodo* nuevo = crearNodo(id, nombre, nac, fallecido);
171             insertar(raiz, nuevo);
172         } else if (opcion == 2) {
173             int id;
174             cout << "Ingrese ID a buscar: ";
175             cin >> id;
176             Nodo* resultado = buscar(raiz, id);
177             if (resultado)
178                 cout << "Encontrado: " << resultado->nombre << " (" << resultado->id << ")\n";
179             else
180                 cout << "No encontrado.\n";
181         } else if (opcion == 3) {
182             int id;
183             cout << "ID a eliminar: ";
184             cin >> id;
185             raiz = eliminar(raiz, id);
186             cout << "Eliminado si existía.\n";
187         } else if (opcion == 4) {
188             int id;
189             cout << "ID para ver ancestros: ";
190             cin >> id;
191             Nodo* n = buscar(raiz, id);
192             if (n) mostrarAncestros(n->padre);
193             else cout << "No encontrado.\n";
194         } else if (opcion == 5) {
195             int id;
196             cout << "ID para ver descendientes: ";
197             cin >> id;
198             Nodo* n = buscar(raiz, id);
199             if (n) mostrarDescendientes(n);
200             else cout << "No encontrado.\n";
201         } else if (opcion == 6) {
202             int tipo;
203             cout << "1. Inorden\n2. Preorden\n3. Postorden\n4. Por niveles\nOpción: ";
204             cin >> tipo;
205             if (tipo == 1) inorden(raiz);
206             else if (tipo == 2) preorden(raiz);
207             else if (tipo == 3) postorden(raiz);
208             else if (tipo == 4) porNiveles(raiz);
209         }
210     } while (opcion != 7);
211
212     cout << "Programa finalizado.\n";
213     return 0;
214 }
```

CAPÍTULO 3 – SOLUCIÓN FINAL

3.1 Código limpio, bien comentado y estructurado.

```
#include <iostream>
#include <string>
#include <queue> //se usa para implementar el algoritmo de Búsqueda en Anchura (BFS)
using namespace std;

struct Nodo {
    int id;
    string nombre;
    int nacimiento;
    string estado;
    Nodo* izq; //hijo menor o hermano menor
    Nodo* der; //hijo mayor o hermano mayor
    Nodo* padre;
};

Nodo* raiz = NULL;
int generaciones=0;

// Crear nuevo nodo
Nodo* crearNodo(int id, string nombre, int nacimiento, string estado) {
    Nodo* nuevo = new Nodo;
    nuevo->id = id;
    nuevo->nombre = nombre;
    nuevo->nacimiento = nacimiento;
    nuevo->estado = estado;
    nuevo->izq = nuevo->der = nuevo->padre = NULL;
    return nuevo;
}

// Buscar miembro por ID
Nodo* buscar(Nodo* actual, int id) {
    if (actual == NULL || actual->id == id) return actual;
    return (id < actual->id) ? buscar(actual->izq, id) : buscar(actual->der, id);
}

bool idExiste(int id) {
    return buscar(raiz, id) != NULL;
}

//validamos el ID
void pedirID(int& id) {
    do {
        cout << "\n\tID (DNI de 8 dígitos)\t: ";
        cin >> id;
        if (id < 10000000 || id > 99999999)
            cout << "\tERROR: El DNI debe tener 8 dígitos.\n";
        else if (idExiste(id))
            cout << "\tERROR: Ya existe un miembro con ese ID.\n";
        else
            break;
    } while (true);
}
```

```

Nodo* solicitarDatosMiembro() {
    int id, nacimiento;
    string nombre, estado;
    bool estadoValido=false; //Bandera para validar el estado de vida

    pedirID(id); //llamamos a nuestra funcion de pedirID llamando tambien idExiste

    cout << "\tNombre\t\t\t: ";
    cin >> nombre;

    do {
        cout << "\tAño de nacimiento\t: ";
        cin >> nacimiento;
        if (nacimiento <= 0 || nacimiento >= 2026)
            cout << "\tERROR: Año inválido. Debe ser mayor a 0 y menor a 2026.\n\n";
    } while (nacimiento <= 0 || nacimiento >= 2026);

    while (!estadoValido) {
        cout << "\t¿Está vivo o ya falleció? (vivo/fallecido): ";
        cin >> estado;

        if (estado != "vivo" && estado != "fallecido") {
            cout << "\tERROR: Debe ingresar 'vivo' o 'fallecido'.\n";
        } else if (nacimiento < 1925 && estado == "vivo") {
            cout << "\tERROR: Incoherente. Personas nacidas antes de 1925 deben haber fallecido.\n\n";
        } else {
            estadoValido = true; //Si pasa ambas condiciones, se acepta
        }
    }
    //Se retorna el nodo creado con los datos válidos
    return crearNodo(id, nombre, nacimiento, estado);
}

int calcularNivel(Nodo* nodo) {
    int nivel=1;
    while (nodo->padre != NULL) {
        nivel++;
        nodo = nodo->padre;
    }
    return nivel;
}

int calcularProfundidad(Nodo* nodo, int nivel) {
    if (nodo == NULL) return nivel - 1;
    int izq = calcularProfundidad(nodo->izq, nivel + 1);
    int der = calcularProfundidad(nodo->der, nivel + 1);
    return (izq > der) ? izq : der;
}

// Buscar Miembro por nombre
Nodo* buscarPorNombre(Nodo* nodo, string nombre) {
    if (!nodo) return NULL;//Caso base: si el nodo actual es NULL, el nombre no está aquí
    // Si el nombre del nodo actual coincide con el buscado, lo devolvemos
    if (nodo->nombre == nombre) return nodo;

    Nodo* izq = buscarPorNombre(nodo->izq, nombre);// Buscamos recursivamente en el subárbol izquierdo
    if (izq) return izq;// Si lo encontramos en el lado izquierdo, lo devolvemos

    return buscarPorNombre(nodo->der, nombre);//Si no lo encuentra en el izquierdo, busca en el derecho
}

```

```

bool validarAnio(Nodo* nuevo, Nodo* relativo, int relacion){
    int diferencia;
    switch (relacion) {
        case 1: { // Padre / madre
            diferencia = relativo->nacimiento - nuevo->nacimiento;
            if (diferencia < 12) { //si la diferencia de la edad del hijo(relativo) - la edad de la raiz(nuevo) es menor a 12..
                cout << "\nError: Es muy joven para ser padre/madre de " << relativo->nombre << ".\n";
                return false;
            }
            return true; //continua y valida la edad en caso de que el padre si sea mayor o igual a 12 años
        }
        //si la edad del nuevo ingresante - el del relativo es menor a 12 decimos que no es valida
        case 2: // Hijo izquierdo
        case 3: { // Hijo derecho
            diferencia = nuevo->nacimiento - relativo->nacimiento;
            if (diferencia < 12) {
                cout << "\n\nError: muy poca diferencia de edad para ser hijo/a de " << relativo->nombre << ".\n";
                return false;
            }
            return true; //si la edad es valida continuamos
        }
        //la diferencia de edad entre el nuevo miembro y la Raiz es mayor a 40
        case 4: // Hermano izquierdo
        case 5: { // Hermano derecho
            diferencia = nuevo->nacimiento - relativo->nacimiento;
            if (diferencia > 40) { //decimos que hay un error ya que no siempre la edad es tan diferente
                cout << "\n\nError: demasiada diferencia de edad para ser hermanos.\n";
                return false;
            }
            return true; //si está dentro del rango entonces continuamos
        }
        default: //si no cumple ninguno de esos casos decimos que no es valida la relación
            cout << "\nRelación no válida.\n";
            return false;
    }
}

void insertarFlexible(Nodo*& raiz, Nodo* nuevo, int generacionesMax, int idRelacionado, int relacion) {
    //Se busca al familiar con el que se establecerá la relación
    Nodo* relativo = buscar(raiz, idRelacionado);
    if (!relativo) {
        cout << "Miembro no encontrado.\n";
        return;
    }

    //Se evalúa el tipo de relación indicada (1 a 5)
    switch (relacion) {
        case 1: //Padre/Madre
            if (relativo == raiz) {
                //Si el relativo es la raíz, el nuevo se convierte en la nueva raíz
                nuevo->izq = raiz;      //El anterior raíz pasa a ser hijo menor del nuevo padre
                raiz->padre = nuevo;   //Se actualiza el puntero padre de la antigua raíz
                raiz = nuevo;           //Se actualiza la raíz del árbol
            } else {
                //Si el relativo no es la raíz
                Nodo* parentAnt = relativo->padre; //Se guarda su parent anterior
                nuevo->padre = parentAnt;           //El nuevo miembro apunta como parent al parent anterior
                relativo->padre = nuevo;           //El relativo ahora es hijo del nuevo
                if (parentAnt) {
                    //Se actualizan los punteros del parent anterior para enlazarlo con el nuevo
                    if (parentAnt->izq == relativo) parentAnt->izq = nuevo;
                    else parentAnt->der = nuevo;
                }
                nuevo->izq = relativo; //El nuevo miembro adopta al relativo como hijo menor
            }
            cout << "\nAsignado como parent/madre.\n";
            break;
    }
}

```

```

        case 2: //Hijo menor
            if (!relativo->izq) {
                //Si el relativo no tiene hijo menor
                relativo->izq = nuevo;      //Se asigna el nuevo como hijo menor
                nuevo->padre = relativo;    //Se enlaza al padre
                cout << "\tAsignado como hijo menor.\n";
            } else {
                cout << "\tYa tiene hijo menor.\n"; //Ya existe un hijo menor
            }break;

        case 3: //Hijo mayor
            if (!relativo->der) {
                //Si el relativo no tiene hijo mayor
                relativo->der = nuevo;      //Se asigna como hijo mayor
                nuevo->padre = relativo;    //Se enlaza al padre
                cout << "\tAsignado como hijo mayor.\n";
            } else {
                cout << "\tYa tiene hijo mayor.\n"; //Ya existe un hijo mayor
            }break;

        case 4: //Hermano menor
            if (!relativo->padre) {
                //Si el relativo no tiene parente registrado, no se puede agregar hermano
                cout << "\tNo tiene parente registrado, no se puede agregar hermano.\n";
                break;
            }
            if (relativo->padre->izq == NULL) {
                //Si el parente no tiene hijo menor aún
                relativo->padre->izq = nuevo;      //Se coloca al nuevo como hijo menor
                nuevo->padre = relativo->padre;    //Se enlaza al mismo parente
                cout << "\tAsignado como hermano menor.\n";
            } else {
                //Ya existe un hijo menor, por lo tanto ya hay hermano en ese lugar
                cout << "\tYa existe un hijo menor, no se puede agregar como hermano.\n";
            }
            break;

        case 5: //Hermano mayor
            if (!relativo->padre) {
                //Si el relativo no tiene parente registrado, no se puede agregar hermano
                cout << "\tNo tiene parente registrado, no se puede agregar hermano.\n";
                break;
            }
            if (relativo->padre->der == NULL) {
                //Si el parente no tiene hijo mayor aún
                relativo->padre->der = nuevo;      //Se coloca al nuevo como hijo mayor
                nuevo->padre = relativo->padre;    //Se enlaza al mismo parente
                cout << "\tAsignado como hermano mayor.\n";
            } else {
                //Ya existe un hijo mayor, por lo tanto ya hay hermano en ese lugar
                cout << "\tYa existe un hijo mayor, no se puede agregar como hermano.\n";
            }
            break;

        default: //En caso de que la opción de relación no sea válida
            cout << "\tRelación no válida.\n";
    }
}

void ingresarMiembro() {
    //Si aún no se ha definido el número de generaciones, se solicita al usuario
    if (generaciones == 0) {
        do {
            cout << "\n\tIngrese cuántas generaciones tendrá el árbol genealógico: ";
            cin >> generaciones;
            if (generaciones <= 0) {
                cout << "\tDebe ser mayor a 0.\n";
            }
        } while (generaciones <= 0);
    }
}

```

```

if (!raiz) {
    raiz = solicitarDatosMiembro();
    cout << "\tPrimer miembro insertado como raíz.\n";
}
else{ //Si ya hay miembros, se solicita ID del familiar relacionado
    int idRelacionado, relacion;
    cout << "\n\tIngrese el ID del familiar relacionado: ";
    cin >> idRelacionado;

    // Se busca el nodo relacionado en el árbol
    Nodo* relativo = buscar(raiz, idRelacionado);
    if (!relativo) { //Si no se encuentra, se cancela
        cout << "\tMiembro no encontrado.\n";
        return;
    }

    cout << "\n\t¿Qué relación tiene con " << relativo->nombre << "?\n";
    cout << "\t1-> Padre/Madre\n";
    cout << "\t2-> Hijo/a menor/a\n";
    cout << "\t3-> Hijo/a mayor/a\n";
    cout << "\t4-> Hermano/a menor/a\n";
    cout << "\t5-> Hermano/a mayor/a\n";
    cout << "\n\tSeleccione una opción: ";
    cin >> relacion;

    //Validación previa: no tiene padre, pero quiere insertar hermano
    if ((relacion==4||relacion==5) && relativo->padre == NULL) {
        cout << "\tError: no se puede insertar hermano/a, el miembro relacionado no tiene padre registrado.\n";
        return;
    }

    //Se calcula el nivel actual del nodo relacionado
    int nivelActual = calcularNivel(relativo);

    if ((relacion == 1 && nivelActual >= generaciones) ||
        ((relacion == 2 || relacion == 3) && nivelActual + 1 > generaciones) ||
        ((relacion == 4 || relacion == 5) && nivelActual > generaciones)) {
        cout << "\tNo se puede insertar: excede las generaciones permitidas.\n";
        return;
    }

    //Se solicita los datos del nuevo miembro
    Nodo* nuevo = solicitarDatosMiembro();

    //Se valida si los años de nacimiento son coherentes con la relación elegida
    if (!validarAnio(nuevo, relativo, relacion)) {
        delete nuevo; // Se libera memoria
        return;
    }
    //Si todo es válido, se inserta el nodo en el árbol con la relación adecuada
    insertarFlexible(raiz, nuevo, generaciones, idRelacionado, relacion);
}

// Mínimo del subárbol
Nodo* minimo(Nodo* actual) {
    while (actual && actual->izq != NULL)
        actual = actual->izq;
    return actual;
}

//ELIMINAR
void eliminarMiembro(Nodo*& nodo, int id) {
    if (nodo == NULL) return; // Si el nodo es nulo, no hay nada que eliminar

    // Recorrer primero el subárbol izquierdo y derecho (postorden)
    eliminarMiembro(nodo->izq, id); // Eliminar en subárbol izquierdo
    eliminarMiembro(nodo->der, id); // Eliminar en subárbol derecho
}

```

```

if (nodo->id == id) {

    // Caso 1: Nodo hoja (sin hijos)
    if (nodo->izq == NULL && nodo->der == NULL) {
        cout << "Se eliminó un nodo hoja (sin hijos): " << nodo->nombre << endl; // Mensaje informativo
        delete nodo; // Liberar memoria del nodo
        nodo = NULL; // Desconectar el puntero para evitar acceso a memoria liberada
    }

    // Caso 2: Nodo con solo hijo izquierdo
    else if (nodo->izq != NULL && nodo->der == NULL) {
        cout << "Se eliminó un nodo con solo hijo izquierdo: " << nodo->nombre << endl; // Mensaje
        Nodo* temp = nodo; // Guardar puntero temporal al nodo a eliminar
        nodo = nodo->izq; // Reemplazar el nodo actual por su hijo izquierdo
        if (nodo != NULL) nodo->padre = temp->padre; // Actualizar puntero padre del nuevo nodo
        delete temp; // Eliminar el nodo original
    }

    // Caso 3: Nodo con solo hijo derecho
    else if (nodo->izq == NULL && nodo->der != NULL) {
        cout << "Se eliminó un nodo con solo hijo derecho: " << nodo->nombre << endl; // Mensaje
        Nodo* temp = nodo; // Guardar puntero temporal
        nodo = nodo->der; // Reemplazar el nodo actual por su hijo derecho
        if (nodo != NULL) nodo->padre = temp->padre; // Actualizar padre
        delete temp; // Liberar memoria del nodo original
    }

    // Caso 4: Nodo con dos hijos
    else {
        cout << "Se eliminó un nodo con dos hijos (reemplazo por subárbol izquierdo): " << nodo->nombre << endl;
        Nodo* reemplazo = nodo->izq; // Buscar nodo máximo en subárbol izquierdo (predecesor)
        Nodo* padreReemplazo = nodo; // Puntero para seguir al padre del reemplazo
        while (reemplazo->der != NULL) { // Mientras haya hijo derecho
            padreReemplazo = reemplazo; // Avanzar padre
            reemplazo = reemplazo->der; // Avanzar al hijo derecho
        }

        nodo->id = reemplazo->id;
        nodo->nombre = reemplazo->nombre;
        // Eliminar recursivamente el nodo duplicado en subárbol izquierdo
        eliminarMiembro(nodo->izq, reemplazo->id);
    }
}
}

//ELIMINAR POR NIVELES
void eliminarPorNivel(Nodo*& raiz, int nivelObjetivo) {
    if (raiz == NULL) return; // Si el árbol está vacío, no hay nada que eliminar

    // Cola para recorrer el árbol en anchura (BFS), guarda pares <nodo, nivel>
    queue<pair<Nodo*, int>> cola; // Nota el espacio entre >
    cola.push(make_pair(raiz, 0)); // Insertar la raíz con nivel 0

    while (!cola.empty()) {
        Nodo* actual = cola.front().first; // Nodo actual a procesar
        int nivel = cola.front().second; // Nivel del nodo actual
        cola.pop(); // Sacar el nodo de la cola

        if (actual == NULL) continue; // Si el nodo es nulo, saltar

        // Encolar hijos con nivel incrementado
        if (actual->izq != NULL) cola.push(make_pair(actual->izq, nivel + 1));
        if (actual->der != NULL) cola.push(make_pair(actual->der, nivel + 1));

        // Si el nivel actual es el nivel objetivo, eliminar el nodo
        if (nivel == nivelObjetivo) {
            cout << "Eliminando miembro en nivel " << nivel << ":" << actual->nombre << endl;
            eliminarMiembro(actual, actual->id); // Llamar a la función para eliminar el nodo específico
        }
    }
}

```

```

void imprimirFicha(Nodo* nodo, const string& tipo) {
    cout << "\t-----\n";
    cout << "\tTipo : " << tipo << "\n";
    cout << "\tNombre : " << nodo->nombre << "\n";
    cout << "\tID (DNI) : " << nodo->id << "\n";
    cout << "\tNacimiento: " << nodo->nacimiento << "\n";
    cout << "\tFallecido : " << nodo->estado << "\n";
}

// Recorridos
void inorder(Nodo* nodo, string tipo = "RAIZ") {
    if (!nodo) return;
    inorder(nodo->izq, "Hijo Izquierdo");
    imprimirFicha(nodo, tipo);
    inorder(nodo->der, "Hijo Derecho");
}

void preorden(Nodo* nodo, string tipo = "RAÍZ") {
    if (!nodo) return;
    imprimirFicha(nodo, tipo);
    preorden(nodo->izq, "Hijo Izquierdo");
    preorden(nodo->der, "Hijo Derecho");
}

void postorden(Nodo* nodo, string tipo = "RAÍZ") {
    if (!nodo) return;
    postorden(nodo->izq, "Hijo Izquierdo");
    postorden(nodo->der, "Hijo Derecho");
    imprimirFicha(nodo, tipo);
}

void porNiveles(Nodo* raiz) {
    if (raiz == NULL) return;

    const int MAX = 100;
    Nodo* cola[MAX];
    int inicio = 0, fin = 0;

    // Encolar raíz
    cola[fin++] = raiz;

    while (inicio < fin) {
        Nodo* actual = cola[inicio++]; // Desencolar

        imprimirFicha(actual, "Pariente");

        // Encolar hijos
        if (actual->izq != NULL) {
            cola[fin++] = actual->izq;
        }
        if (actual->der != NULL) {
            cola[fin++] = actual->der;
        }
    }
}

//Ancestros
void mostrarAncestros(Nodo* ancestro, Nodo* hijoOriginal) {
    if (ancestro == NULL) { // Si no hay ancestro
        cout << "\n\tEste nodo no tiene ancestros registrados.\n";
        return;
    }

    int nivel = 1; // Contador de generaciones ascendentes

    // Recorremos hacia arriba por el árbol mientras existan padres
    while (ancestro) {
        // Mostramos el nivel generacional y los datos del ancestro actual
        cout << "\n\tNivel " << nivel << " - Ancestro:\n";
        imprimirFicha(ancestro, "Ancestro");

        // calculamos la diferencia de nacimiento entre el ancestro y el miembro consultado
        int diferencia = (hijoOriginal->nacimiento > ancestro->nacimiento)
                        ? (hijoOriginal->nacimiento - ancestro->nacimiento)
                        : (ancestro->nacimiento - hijoOriginal->nacimiento);
    }
}

```

```

        cout << "\tDiferencia de nacimiento con respecto a " << hijoOriginal->nombre
        << ":" << diferencia << " años\n";

    // Subimos un nivel más arriba en la línea de ancestros
    ancestro = ancestro->padre;
    nivel++; // Aumentamos el nivel generacional
}
}

// Descendientes (en preorden)
void mostrarDescendientes(Nodo* nodo) {

    if (!nodo) return;// Caso base: si el nodo es nulo, salimos de la función

    // Si el nodo es hoja, mostramos esto
    if (nodo->izq == NULL && nodo->der == NULL) {
        cout << "\n\tEste nodo no tiene hijos (hojas).\n";
        return;
    }

    // Si tiene hijo izquierdo, mostramos su información
    // y llamamos recursivamente para buscar más descendientes desde allí
    if (nodo->izq) {
        imprimirFicha(nodo->izq, "Descendiente (izquierdo)");
        mostrarDescendientes(nodo->izq);
    }

    // Si tiene hijo derecho, lo mostramos e iniciamos recorrido desde ese hijo
    if (nodo->der) {
        imprimirFicha(nodo->der, "Descendiente (derecho)");
        mostrarDescendientes(nodo->der);
    }
}

int main() {
    setlocale(LC_CTYPE, "Spanish");

    int opcion;
    do {
        cout << "\n\t-----\n";
        cout << "\t|---- MENU ÁRBOL GENEALÓGICO ----|\n";
        cout << "\t-----\n";
        cout << "\n\t[1]. Insertar miembro\n";
        cout << "\t[2]. Buscar miembro\n";
        cout << "\t[3]. Eliminar miembro\n";
        cout << "\t[4]. Consultar ancestros\n";
        cout << "\t[5]. Consultar descendientes\n";
        cout << "\t[6]. Recorridos\n";
        cout << "\t[7]. Salir\n";
        cout << "\n\tSeleccione una opción: ";
        cin >> opcion;

        switch (opcion) {
            case 1:
                ingresarMiembro();
                break;
            case 2: {
                int id;
                cout << "\n\tIngrese ID a buscar: ";
                cin >> id;
                Nodo* resultado = buscar(raiz, id);
                if (resultado)
                    cout << "\tEncontrado: " << resultado->nombre << "(" << resultado->id << ")\n";
                else
                    cout << "\tNo encontrado.\n";
                break;
            }
        }
    } while (opcion != 7);
}

```

```

case 3: {
    int tipoEliminacion;
    cout << "\n\tTipo de eliminación:\n";
    cout << "\t[1] Eliminar por ID\n";
    cout << "\t[2] Eliminar por nivel\n";
    cout << "\tSeleccione: ";
    cin >> tipoEliminacion;

    if (tipoEliminacion == 1) {
        int id;
        cout << "\tID a eliminar: ";
        cin >> id;
        eliminarMiembro(raiz, id);
    }
    else if (tipoEliminacion == 2) {
        int nivel;
        cout << "\tNivel a eliminar: ";
        cin >> nivel;
        eliminarPorNivel(raiz, nivel);
    }
    else {
        cout << "\topción inválida\n";
    }
    break;
}

case 4: {
    int tipoBusqueda;
    cout << "\n\t¿Cómo desea buscar al miembro?\n";
    cout << "\t[1] Por ID\n";
    cout << "\t[2] Por nombre\n";
    cout << "\tSeleccione una opción: ";
    cin >> tipoBusqueda;

    Nodo* n = NULL;
    if (tipoBusqueda == 1) {
        int id;
        cout << "\tingrese ID para ver ancestros: ";
        cin >> id;
        n = buscar(raiz, id);
    } else if (tipoBusqueda == 2) {
        string nombre;
        cout << "\tingrese nombre exacto para ver ancestros: ";
        cin >> nombre;
        n = buscarPorNombre(raiz, nombre);
    } else {
        cout << "\topción inválida.\n";
        break;
    }

    if (n)
        mostrarAncestros(n->padre, n);
    else
        cout << "\tMiembro no encontrado.\n";

    break;
}

case 5: {
    int tipoBusqueda;
    cout << "\n\t¿Cómo desea buscar al miembro?\n";
    cout << "\t[1] Por ID\n";
    cout << "\t[2] Por nombre\n";
    cout << "\tSeleccione una opción: ";
    cin >> tipoBusqueda;

    Nodo* n = NULL;
    if (tipoBusqueda == 1) {
        int id;
        cout << "\tingrese ID para ver descendientes: ";
        cin >> id;
        n = buscar(raiz, id);
    } else if (tipoBusqueda == 2) {
        string nombre;
        cout << "\tingrese nombre exacto para ver descendientes: ";
        cin >> nombre;
        n = buscarPorNombre(raiz, nombre);
    } else {
        cout << "\topción inválida.\n";
        break;
    }
}

```

```

        if (n)
            mostrarDescendientes(n);
        else
            cout << "\tMiembro no encontrado.\n";

        break;
    }

    case 6: {
        int tipo;
        cout << "\n\t1. Inorden\n\t2. Preorden\n\t3. Postorden\n\t4. Por niveles\n\n\tOpción: ";
        cin >> tipo;
        switch (tipo) {
            case 1:
                cout << "\n\t===== RECORRIDO EN INORDEN =====\n";
                inorden(raiz); break;
            case 2:
                cout << "\n\t===== RECORRIDO EN PREORDEN =====\n";
                preorden(raiz); break;
            case 3:
                cout << "\n\t===== RECORRIDO EN POSTORDEN =====\n";
                postorden(raiz); break;
            case 4:
                cout << "\n\t===== RECORRIDO POR NIVELES =====\n";
                porNiveles(raiz); break;
            default: cout << "\n\tOpción no válida.\n"; break;
        }
        break;
    }

    case 7:
        cout << "\tPrograma finalizado.\n";
        break;

    case 7:
        cout << "\tPrograma finalizado.\n";
        break;

    default:
        cout << "\tOpción inválida. Intente de nuevo.\n";
        break;
    }

} while (opcion != 7);

return 0;
}

```

3.2 Capturas de pantalla de las ventanas de ejecución con pruebas de validación de datos

- Opción 1:

```
|---- MENU ÁRBOL GENEALÓGICO ----|  
  
[1]. Insertar miembro  
[2]. Buscar miembro  
[3]. Eliminar miembro  
[4]. Consultar ancestros  
[5]. Consultar descendientes  
[6]. Recorridos  
[7]. Salir  
  
Seleccione una opción: 1  
  
Ingrese cuántas generaciones tendrá el árbol genealógico: 3  
  
ID (DNI de 8 dígitos) : 1  
ERROR: El DNI debe tener 8 dígitos.  
  
ID (DNI de 8 dígitos) : 12345678  
Nombre : Paulina  
Año de nacimiento : 64512  
ERROR: Año inválido. Debe ser mayor a 0 y menor a 2026.  
  
Año de nacimiento : 1545  
¿Está vivo o ya falleció? (vivo/fallecido): vivo  
ERROR: Incoherente. Personas nacidas antes de 1925 deben haber fallecido.  
  
¿Está vivo o ya falleció? (vivo/fallecido): fallecido  
Primer miembro insertado como raíz.
```

Al ingresar más miembros:

```
|---- MENU ÁRBOL GENEALÓGICO ----|  
  
[1]. Insertar miembro  
[2]. Buscar miembro  
[3]. Eliminar miembro  
[4]. Consultar ancestros  
[5]. Consultar descendientes  
[6]. Recorridos  
[7]. Salir  
  
Seleccione una opción: 1  
  
Ingrese el ID del familiar relacionado: 45  
Miembro no encontrado.
```

```
[6]. Recorridos  
[7]. Salir  
  
Seleccione una opción: 1  
  
Ingrese el ID del familiar relacionado: 12345678  
  
¿Qué relación tiene con Paulina?  
1-> Padre/Madre  
2-> Hijo/a menor/a  
3-> Hijo/a mayor/a  
4-> Hermano/a menor/a  
5-> Hermano/a mayor/a  
  
Seleccione una opción: 4  
Error: no se puede insertar hermano/a, el miembro relacionado no tiene parente registrado.
```

```

[2]. Buscar miembro
[3]. Eliminar miembro
[4]. Consultar ancestros
[5]. Consultar descendientes
[6]. Recorridos
[7]. Salir

Seleccione una opción: 1

Ingrese el ID del familiar relacionado: 12345678

¿Qué relación tiene con Paulina?
1-> Padre/Madre
2-> Hijo/a menor/a
3-> Hijo/a mayor/a
4-> Hermano/a menor/a
5-> Hermano/a mayor/a

Seleccione una opción: 1

ID (DNI de 8 dígitos) : 12345678
ERROR: Ya existe un miembro con ese ID.

ID (DNI de 8 dígitos) : 23456789
Nombre : Luis
Año de nacimiento : 1543
¿Está vivo o ya falleció? (vivo/fallecido): fallecido
Error: Es muy joven para ser padre/madre de Paulina.

```

Cuando no hay errores:

```

Seleccione una opción: 1

Ingrese el ID del familiar relacionado: 12345678

¿Qué relación tiene con Paulina?
1-> Padre/Madre
2-> Hijo/a menor/a
3-> Hijo/a mayor/a
4-> Hermano/a menor/a
5-> Hermano/a mayor/a

Seleccione una opción: 1

ID (DNI de 8 dígitos) : 23456789
Nombre : Luis
Año de nacimiento : 1520
¿Está vivo o ya falleció? (vivo/fallecido): fallecido
Asignado como parentesco.

```

Validación si es parentesco menor a 12 años (la niña aún no es fértil)

```

Seleccione una opción: 1

Ingrese cuántas generaciones tendrá el árbol genealógico: 3

ID (DNI de 8 dígitos) : 21825020
Nombre : Gisella
Año de nacimiento : 2000
¿Está vivo o ya falleció? (vivo/fallecido): vivo
Primer miembro insertado como raíz.

Ingrese el ID del familiar relacionado: 21825020

¿Qué relación tiene con Gisella?
1-> Padre/Madre
2-> Hijo/a menor/a
3-> Hijo/a mayor/a
4-> Hermano/a menor/a
5-> Hermano/a mayor/a

Seleccione una opción: 1

ID (DNI de 8 dígitos) : 72440876
Nombre : Luisana
Año de nacimiento : 2003
¿Está vivo o ya falleció? (vivo/fallecido): vivo
Error: Es muy joven para ser parentesco de Gisella.
No se puede insertar: años de nacimiento INCOHERENTES para la relación indicada.

```

- Opción 2:

<p> ---- MENU ÁRBOL GENEALÓGICO ---- </p> <hr/> <p>[1]. Insertar miembro [2]. Buscar miembro [3]. Eliminar miembro [4]. Consultar ancestros [5]. Consultar descendientes [6]. Recorridos [7]. Salir</p> <p>Seleccione una opción: 2</p> <p>Ingrese ID a buscar: 19784564 No encontrado.</p>	<p> ---- MENU ÁRBOL GENEALÓGICO ---- </p> <hr/> <p>[1]. Insertar miembro [2]. Buscar miembro [3]. Eliminar miembro [4]. Consultar ancestros [5]. Consultar descendientes [6]. Recorridos [7]. Salir</p> <p>Seleccione una opción: 2</p> <p>Ingrese ID a buscar: 12345678 Encontrado: Paulina (12345678)</p>
--	--

- Opción 3:

<p> ---- MENU ÁRBOL GENEALÓGICO ---- </p> <hr/> <p>[1]. Insertar miembro [2]. Buscar miembro [3]. Eliminar miembro [4]. Consultar ancestros [5]. Consultar descendientes [6]. Recorridos [7]. Salir</p> <p>Seleccione una opción: 3</p> <p>Tipo de eliminación: [1] Eliminar por ID [2] Eliminar por nivel Seleccione: 1 ID a eliminar: 45687912 Se eliminó un nodo hoja (sin hijos): jacinto</p>	<p> ---- MENU ÁRBOL GENEALÓGICO ---- </p> <hr/> <p>[1]. Insertar miembro [2]. Buscar miembro [3]. Eliminar miembro [4]. Consultar ancestros [5]. Consultar descendientes [6]. Recorridos [7]. Salir</p> <p>Seleccione una opción: 3</p> <p>Tipo de eliminación: [1] Eliminar por ID [2] Eliminar por nivel Seleccione: 2 Nivel a eliminar: 1 Eliminando miembro en nivel 1: luis Se eliminó un nodo hoja (sin hijos): luis</p>
--	--

- Opción 4:

<p> ---- MENU ÁRBOL GENEALÓGICO ---- </p> <hr/> <p>[1]. Insertar miembro [2]. Buscar miembro [3]. Eliminar miembro [4]. Consultar ancestros [5]. Consultar descendientes [6]. Recorridos [7]. Salir</p> <p>Seleccione una opción: 4</p> <p>¿Cómo desea buscar al miembro? [1] Por ID [2] Por nombre Seleccione una opción:</p>

Si se elige una forma no adecuada

```
Seleccione una opción: 4

¿Cómo desea buscar al miembro?
[1] Por ID
[2] Por nombre
Seleccione una opción: 3
Opción inválida.
```

Funcionamiento correcto, te muestra al ancestro

```
Seleccione una opción: 4

¿Cómo desea buscar al miembro?
[1] Por ID
[2] Por nombre
Seleccione una opción: 2
Ingrese nombre exacto para ver ancestros: yat

Nivel 1 - Ancestro:
-----
Tipo      : Ancestro
Nombre    : Wily
ID (DNI)  : 78945612
Nacimiento: 1975
Fallecido : vivo
Diferencia de nacimiento con respecto a yat: 32 años
```

- Opción 5:

```
[1]. Insertar miembro
[2]. Buscar miembro
[3]. Eliminar miembro
[4]. Consultar ancestros
[5]. Consultar descendientes
[6]. Recorridos
[7]. Salir

Seleccione una opción: 5

¿Cómo desea buscar al miembro?
[1] Por ID
[2] Por nombre
Seleccione una opción: 2
Ingrese nombre exacto para ver descendientes: Wily
```

Muestra los miembros descendientes

```
-----
Tipo      : Descendiente (izquierdo)
Nombre    : yat
ID (DNI)  : 61114889
Nacimiento: 2007
Fallecido : vivo

Este nodo no tiene hijos (hojas).

-----
Tipo      : Descendiente (derecho)
Nombre    : Gerald
ID (DNI)  : 12345678
Nacimiento: 2002
Fallecido : vivo

Este nodo no tiene hijos (hojas).
```

Menciona que el recorrido de parte izquierda como derecha acaba ahí por que no cuentan con nodos hojas (Hijos)

- Opción 6:

<pre>1. Inorden 2. Preorden 3. Postorden 4. Por niveles Opción: 1 ===== RECORRIDO EN INORDEN ====== ----- Tipo : Hijo Izquierdo Nombre : Luisana ID (DNI) : 72440876 Nacimiento: 2003 Fallecido : vivo ----- Tipo : RAÍZ Nombre : Gisell ID (DNI) : 21825020 Nacimiento: 1973 Fallecido : vivo ----- Tipo : Hijo Derecho Nombre : Adriana ID (DNI) : 72440875 Nacimiento: 2000 Fallecido : vivo</pre>	<pre>1. Inorden 2. Preorden 3. Postorden 4. Por niveles Opción: 2 ===== RECORRIDO EN PREORDEN ====== ----- Tipo : RAÍZ Nombre : Gisell ID (DNI) : 21825020 Nacimiento: 1973 Fallecido : vivo ----- Tipo : Hijo Izquierdo Nombre : Luisana ID (DNI) : 72440876 Nacimiento: 2003 Fallecido : vivo ----- Tipo : Hijo Derecho Nombre : Adriana ID (DNI) : 72440875 Nacimiento: 2000 Fallecido : vivo</pre>
<pre>1. Inorden 2. Preorden 3. Postorden 4. Por niveles Opción: 3 ===== RECORRIDO EN POSTORDEN ====== ----- Tipo : Hijo Izquierdo Nombre : Luisana ID (DNI) : 72440876 Nacimiento: 2003 Fallecido : vivo ----- Tipo : Hijo Derecho Nombre : Adriana ID (DNI) : 72440875 Nacimiento: 2000 Fallecido : vivo ----- Tipo : RAÍZ Nombre : Gisell ID (DNI) : 21825020 Nacimiento: 1973 Fallecido : vivo</pre>	<pre>1. Inorden 2. Preorden 3. Postorden 4. Por niveles Opción: 4 ===== RECORRIDO POR NIVELES ====== ----- Tipo : Pariente Nombre : Gisell ID (DNI) : 21825020 Nacimiento: 1973 Fallecido : vivo ----- Tipo : Pariente Nombre : Luisana ID (DNI) : 72440876 Nacimiento: 2003 Fallecido : vivo ----- Tipo : Pariente Nombre : Adriana ID (DNI) : 72440875 Nacimiento: 2000 Fallecido : vivo</pre>

CAPÍTULO 4 – EVIDENCIAS DE TRABAJO COLABORATIVO

4.1 Repositorio con Control de Versiones (Capturas de Pantalla)

- Registro de commits claros y significativos que evidencian aportes individuales (proactividad).

- Limaymanta Castro Dayana Evelin

The screenshot shows a Git commit history for a repository named 'ArbolGenealogico'. It displays two main branches of activity:

- Se compromete**:
 - Commit 1: Correcciones Insertar - Dayana (Verified) cd8a683
 - Commit 2: INSERTAR - Limaymanta Castro Dayana (Verified) aaaa027
- Se compromete el 19 de junio de 2025**:
 - Commit 1: INSERTAR - Limaymanta Castro Dayana (Verified) 7489321
 - Commit 2: INSERTAR - Limaymanta Castro Dayana (Verified) 2c84b1c
 - Commit 3: Corrección del Menú - Dayana (Verified) 0333148

Below this, there is a section titled "Compromiso 2c84b1c" which contains a single commit:

- Commit 1: INSERTAR - Limaymanta Castro Dayana (Verified)

The screenshot shows a code diff tool comparing two versions of a C++ file. Both versions are identical, showing the same code for inserting nodes into a binary search tree.

```
1 archivo cambiado +100 -23 líneas cambiadas
Estructura
29 - void insertar(Nodo*& actual, Nodo* nuevo, Nodo* padre = NULL) {
30 -     if (actual == NULL) {
31 -         nuevo->padre = padre;
32 -         actual = nuevo;
33 -         cout << "Miembro insertado correctamente.\n";
34 -     } else if (nuevo->id < actual->id) {
35 -         insertar(actual->izq, nuevo, actual);
36 -     } else {
37 -         insertar(actual->der, nuevo, actual);
29 + void insertarFlexible(Nodo*& raiz, Nodo* nuevo) { //NO CAMBIAR - DAYANA
30 +     if (!raiz) {
31 +         raiz = nuevo;
32 +         cout << "Primer miembro insertado como raiz.\n";
33 +         return;
34 +     }
35 +
36 +     int idRelacionado;
37 +     cout << "Ingrese el ID del familiar relacionado: ";
38 +     cin >> idRelacionado;
39 +     Nodo* relativo = buscar(raiz, idRelacionado);
40 +
41 +     if (!relativo) {
42 +         cout << "Miembro no encontrado.\n";
43 +         return;
44 +     }
45 +
46 +     cout << "¿Qué relación tiene con " << relativo->nombre << "?\n";
47 +     cout << "1. Padre/Madre\n";
48 +     cout << "2. Hijo/a izquierdo/a\n";
```

1 archivo cambiado +100 -23 líneas cambiadas

```

    ✓ Estructura □ ⌂

    79 +         } else cout << "Ya tiene hijo izquierdo.\n";
    80 +         break;
    81 +     case 3:
    82 +         if (!relativo->der) {
    83 +             nuevo->padre = relativo;
    84 +             cout << "Asignado como hijo derecho.\n";
    85 +         } else cout << "Ya tiene hijo derecho.\n";
    86 +         break;
    87 +     case 4:
    88 +     case 5:
    89 +         if (!relativo->padre) {
    90 +             cout << "No tiene padre registrado, no se puede agregar hermano.\n";
    91 +             break;
    92 +         }
    93 +     }
    94 +     if (relacion == 4 && (!relativo->padre->izq || relativo->padre->izq == relativo)) {
    95 +         relativo->padre = nuevo;
    96 +         nuevo->padre = relativo->padre;
    97 +         cout << "Asignado como hermano izquierdo.\n";
    98 +     } else if (relacion == 5 && (!relativo->padre->der || relativo->padre->der == relativo)) {
    99 +         relativo->padre = nuevo;
   100 +        nuevo->padre = relativo->padre;
   101 +        cout << "Asignado como hermano derecho.\n";
   102 +    } else {
   103 +        cout << "Ya existe un hermano en esa posición.\n";
   104 +    }
   105 +    break;
   106 + default:
   107 +     cout << "Relación no válida.\n";

```

Compromiso 7489321

 Dayana Limaymanta de autoría la semana pasada (Verificado)

 Explorar archivos

INSERTAR - Limaymanta Castro Dayana

 principal

1 padre 2c84b1ccomprometerse7489321

1 archivo cambiado +119 -72 líneas cambiadas

```

    ✓ Estructura □ ⌂

    44 -     }
    45 -     cout << "¿Qué relación tiene con " << relativo->nombre << "?\n";
    46 -     cout << "1. Padre/Madre\n";
    47 -     cout << "2. Hijo/a izquierdo/a\n";
    48 -     cout << "3. Hijo/a derecho/a\n";
    49 -     cout << "4. Hermano/a izquierdo/a\n";
    50 -     cout << "5. Hermano/a derecho/a\n";
    51 -     cout << "Opción: ";
    52 -     int relacion;
    53 -     cin >> relacion;
    54 -
    55 * // Insertar miembro
    56 * void insertarFlexible(Nodo*& raiz, Nodo* nuevo, int generacionesMax, int idRelacionado, int relacion) { //NO CAMBIAR - DAYANA
    57 *     Nodo* relativo = buscar(raiz, idRelacionado);
    58 *     if (!relativo) {
    59 *         cout << "Miembro no encontrado.\n";
    60 *         return;
    61 *     }
    62 *
    63 *     switch (relacion) {
    64 *         case 1:
    65 *             cout << "Insertando hermano izquierdo...\n";
    66 *             if (relativo->izq == NULL) {
    67 *                 relativo->izq = nuevo;
    68 *                 nuevo->padre = relativo;
    69 *                 cout << "Insertado hermano izquierdo.\n";
    70 *             } else {
    71 *                 cout << "No tiene hermano izquierdo.\n";
    72 *             }
    73 *         case 2:
    74 *             cout << "Insertando hermano derecho...\n";
    75 *             if (relativo->der == NULL) {
    76 *                 relativo->der = nuevo;
    77 *                 nuevo->padre = relativo;
    78 *                 cout << "Insertado hermano derecho.\n";
    79 *             } else {
    80 *                 cout << "No tiene hermano derecho.\n";
    81 *             }
    82 *         case 3:
    83 *             cout << "Insertando hijo izquierdo...\n";
    84 *             if (relativo->izq == NULL) {
    85 *                 relativo->izq = nuevo;
    86 *                 nuevo->padre = relativo;
    87 *                 cout << "Insertado hijo izquierdo.\n";
    88 *             } else {
    89 *                 cout << "No tiene hijo izquierdo.\n";
    90 *             }
    91 *         case 4:
    92 *             cout << "Insertando hijo derecho...\n";
    93 *             if (relativo->der == NULL) {
    94 *                 relativo->der = nuevo;
    95 *                 nuevo->padre = relativo;
    96 *                 cout << "Insertado hijo derecho.\n";
    97 *             } else {
    98 *                 cout << "No tiene hijo derecho.\n";
    99 *             }
   100 *         default:
   101 *             cout << "Relación no válida.\n";

```

1 archivo cambiado +119 -72 líneas cambiadas

```

    ✓ Estructura □ ⌂

    105 -     break;
    106 -     case 4:
    107 -         if (!relativo->padre) {
    108 -             cout << "No tiene padre registrado, no se puede agregar hermano.\n";
    109 -             break;
    110 -         }
    111 -         if (relativo->padre->izq == NULL) {
    112 -             relativo->padre->izq = nuevo;
    113 -             nuevo->padre = relativo->padre;
    114 -             cout << "Asignado como hermano izquierdo.\n";
    115 -         } else {
    116 -             cout << "Ya existe un hermano izquierdo, no se puede agregar como hermano.\n";
    117 -         }
    118 -         break;
    119 -     case 5:
    120 -         if (!relativo->padre) {
    121 -             cout << "No tiene padre registrado, no se puede agregar hermano.\n";
    122 -             break;
    123 -         }
    124 -         if (relativo->padre->der == NULL) {
    125 -             relativo->padre->der = nuevo;
    126 -             nuevo->padre = relativo->padre;
    127 -             cout << "Asignado como hermano derecho.\n";
    128 -         } else {
    129 -             cout << "Ya existe un hermano derecho, no se puede agregar como hermano.\n";
    130 -         }
    131 -         break;
    132 -     default:
    133 -         cout << "Relación no válida.\n";

```

1 archivo cambiado +119 -72 líneas cambiadas

▼ Estructura ⌂ ⌃

```
119 189     while (actual && actual->izq != NULL)
120 190     @@ -213,12 +203,29 @@ void mostrarDescendientes(Nodo* nodo) {
121 191         ...
122 192         ...
123 193             mostrarDescendientes(nodo->der);
124 194         }
125 195
126 196         + int calcularNivel(Nodo* nodo) {
127 197             +     int nivel = 1;
128 198             +     while (nodo->padre != NULL) {
129 199                 +         nivel++;
130 200                 nodo = nodo->padre;
131 201             }
132 202             +     return nivel;
133 203         }
134 204
135 205         + int calcularProfundidad(Nodo* nodo, int nivel) {
136 206             +     if (nodo == NULL) return nivel - 1;
137 207             +     int izq = calcularProfundidad(nodo->izq, nivel + 1);
138 208             +     int der = calcularProfundidad(nodo->der, nivel + 1);
139 209             +     return (izq > der) ? izq : der;
140 210         }
141 211
142 212 // Menú principal
143 213 int main() {
144
145     -         int generaciones; //NO CAMBIAR - DAYANA
146
147     -         cout << "Ingrese cuántos miembros desea agregar al árbol genealógico: ";
148
149     -         cin >> generaciones;
150
151     -
```

Confirmación aaaa027

 Dayana Limaymanta de autoría hace 52 minutos Verificado

 Explorar archivos

INSERTAR - Limaymanta Castro Dayana

principal 1 padre 6bac45a comprometerse aaae027

1 archivo cambiado +240 -180 líneas cambiadas

```
1 Estructura + 
2 
3 9 +     string estado;
4 10 +    Nodo* izq; //hijo menor o hermano menor
5 11 +    Nodo* der; //hijo mayor o hermano mayor
6 
7 12 12    Nodo* padre;
8 13 13 };
9 14 14
10 15    Nodo* raiz = NULL;
11 16 + int generaciones=0;
12 17
13 18 // Crear nuevo nodo
14 19 - Nodo* crearNodo(int id, string nombre, int nacimiento, string fallecimiento) {
15 20 + Nodo* crearNodo(int id, string nombre, int nacimiento, string estado) {
16 21
17 22     Nodo* nuevo = new Nodo();
18 23     nuevo->id = id;
19 24     nuevo->nombre = nombre;
20 25     nuevo->nacimiento = nacimiento;
21 26     nuevo->fallecimiento = fallecimiento;
22 27     nuevo->estado = estado;
23 28
24 29     nuevo->izq = nuevo->der = nuevo->padre = NULL;
25 30     return nuevo;
26 31 }
27 32
28 33 + //Solicita y valida los datos de un nuevo miembro
29 34 + Nodo* solicitarDatosMiembro() {
30 35
31 36     int id, nacimiento;
32 37     string nombre, estado;
33 38     bool estadoValido=false; //Bandera para validar el estado de vida
34 39
35 40     do {
36 41         cout << "Introduzca el ID del miembro: ";
37 42         cin >> id;
38 43
39 44         cout << "Introduzca el nombre del miembro: ";
40 45         cin >> nombre;
41 46
42 47         cout << "Introduzca la fecha de nacimiento (dd/mm/aaaa): ";
43 48         cin >> nacimiento;
44 49
45 50         cout << "Introduzca el estado de vida (Vivo/Defunto): ";
46 51         cin >> estado;
47 52
48 53         if(estado=="Vivo" || estado=="Defunto") {
49 54             estadoValido=true;
50 55         } else {
51 56             cout << "Error: Estado de vida invalido. Introduzca 'Vivo' o 'Defunto'.\n";
52 57         }
53 58
54 59         if(!validarFecha(nacimiento)) {
55 60             cout << "Error: Fecha de nacimiento invalida.\n";
56 61         }
57 62
58 63     } while(!estadoValido);
59 64
60 65     return new Nodo(id, nombre, nacimiento, estado);
61 66 }
```

1 archivo cambiado +240 -180 líneas cambiadas

Estructura

```
34 +
35 +     do {
36 +         cout << "\tID (DNI de 8 dígitos)\t: ";
37 +         cin >> id;
38 +         if (id < 10000000 || id > 99999999)
39 +             cout << "\tERROR: El DNI debe tener 8 dígitos.\n";
40 +     } while (id < 10000000 || id > 99999999);
41 +
42 +     cout << "\tNombre\t\t\t: ";
43 +     cin >> nombre;
44 +
45 +     do {
46 +         cout << "\tAño de nacimiento\t: ";
47 +         cin >> nacimiento;
48 +         if (nacimiento <= 0 || nacimiento >= 2026)
49 +             cout << "\tERROR: Año inválido. Debe ser mayor a 0 y menor a 2026.\n";
50 +     } while (nacimiento <= 0 || nacimiento >= 2026);
51 +
52 +     while (!estadoValido) {
53 +         cout << "\t¿Está vivo o ya falleció? (vivo/fallecido): ";
54 +         cin >> estado;
55 +
56 +         if (estado != "vivo" && estado != "fallecido") {
57 +             cout << "\tERROR: Debe ingresar 'vivo' o 'fallecido'.\n";
58 +         } else if (nacimiento < 1925 && estado == "vivo") {
```

1 archivo cambiado +240 -180 líneas cambiadas

```

Estructura □ +:
57 +     cout << "\tTERROR: Debe ingresar 'vivo' o 'fallecido'.\n";
58 + } else if (nacimiento < 1925 && estado == "vivo") {
59 +     cout << "\tTERROR: Incoherente. Personas nacidas antes de 1925 deben haber fallecido.\n";
60 + } else {
61 +     estadoValido = true; //Si pasa ambas condiciones, se acepta
62 + }
63 + }
64 + //Se retorna el nodo creado con los datos válidos
65 + return crearNodo(id, nombre, nacimiento, estado);
66 +
67 +
68 + //Calcula el nivel de un nodo desde la raíz (raíz=nivel 1)
69 + int calcularNivel(Nodo* nodo) {
70 +     int nivel=1;
71 +     while (nodo->padre != NULL) {
72 +         nivel++;
73 +         nodo = nodo->padre;
74 +     }
75 +     return nivel;
76 +
77 +
78 + int calcularProfundidad(Nodo* nodo, int nivel) {
79 +     if (nodo == NULL) return nivel - 1;
80 +     int izq = calcularProfundidad(nodo->izq, nivel + 1);
81 +     int der = calcularProfundidad(nodo->der, nivel + 1);
82 +
83 + }
84 +

```

1 archivo cambiado +240 -180 líneas cambiadas

```

Estructura □ +:
11b L22
233 + void ingresarMiembro() {
234 +     //Si aún no se ha definido el número de generaciones, se solicita al usuario
235 +     if (generaciones == 0) {
236 +         do {
237 +             cout << "\n\tIngrese cuántas generaciones tendrá el árbol genealógico: ";
238 +             cin >> generaciones;
239 +             if (generaciones < 0) {
240 +                 cout << "\tDebe ser mayor a 0.\n";
241 +             }
242 +         } while (generaciones < 0);
243 +     }
244 +     //Si la raíz no existe, se inserta el primer miembro como raíz
245 +     if (!raiz) {
246 +         raiz = solicitarDatosMiembro();
247 +         cout << "\tPrimer miembro insertado como raíz.\n";
248 +     }
249 +     //Si ya hay miembros, se solicita ID del familiar relacionado
250 +     int idRelacionado, relacion;
251 +     cout << "\n\tIngrese el ID del familiar relacionado: ";
252 +     cin >> idRelacionado;
253 +
254 +     // Se busca el nodo relacionado en el árbol
255 +     Nodo* relativo = buscar(raiz, idRelacionado);
256 +     if (!relativo) { //Si no se encuentra, se cancela
257 +         cout << "\tMiembro no encontrado.\n";
258 +         return;
259 +     }
260 +

```

1 archivo cambiado +240 -180 líneas cambiadas

```

Estructura □ +:
261 +     cout << "\n\tQué relación tiene con " << relativo->nombre << "?\n";
262 +     cout << "\t1-> Padre/Madre\n";
263 +     cout << "\t2-> Hijo/a menor/a\n";
264 +     cout << "\t3-> Hijo/a mayor/a\n";
265 +     cout << "\t4-> Hermano/a menor/a\n";
266 +     cout << "\t5-> Hermano/a mayor/a\n";
267 +     cout << "\n\tSeleccione una opción: ";
268 +     cin >> relacion;
269 +
270 +     // Se calcula el nivel actual del nodo relacionado
271 +     int nivelActual = calcularNivel(relativo);
272 +
273 +     //Validaciones para no exceder el límite de generaciones según el tipo de relación
274 +     if ((relacion == 1 && nivelActual >= generaciones) ||
275 +         ((relacion == 2 || relacion == 3) && nivelActual + 1 > generaciones) ||
276 +         ((relacion == 4 || relacion == 5) && nivelActual > generaciones)) {
277 +         cout << "\tNo se puede insertar: excede las generaciones permitidas.\n";
278 +         return;
279 +     }
280 +
281 +     //Se solicita los datos del nuevo miembro
282 +     Nodo* nuevo = solicitarDatosMiembro();
283 +
284 +     //Se valida si los años de nacimiento son coherentes con la relación elegida
285 +     if (!validarAnio(nuevo, relativo, relacion)) {
286 +         cout << "\tNo se puede insertar: años de nacimiento INCOHERENTES para la relación indicada.\n";
287 +         delete nuevo; // Se libera memoria
288 +         return;
289 +     }
290 +
291 +     //Si todo es válido, se inserta el nodo en el árbol con la relación adecuada
292 +     insertarFlexible(raiz, nuevo, generaciones, idRelacionado, relacion);
293 +

```

1 archivo cambiado +240 -180 líneas cambiadas

```

Estructura □ +:
271 +     int nivelActual = calcularNivel(relativo);
272 +
273 +     //Validaciones para no exceder el límite de generaciones según el tipo de relación
274 +     if ((relacion == 1 && nivelActual >= generaciones) ||
275 +         ((relacion == 2 || relacion == 3) && nivelActual + 1 > generaciones) ||
276 +         ((relacion == 4 || relacion == 5) && nivelActual > generaciones)) {
277 +         cout << "\tNo se puede insertar: excede las generaciones permitidas.\n";
278 +         return;
279 +     }
280 +
281 +     //Se solicita los datos del nuevo miembro
282 +     Nodo* nuevo = solicitarDatosMiembro();
283 +
284 +     //Se valida si los años de nacimiento son coherentes con la relación elegida
285 +     if (!validarAnio(nuevo, relativo, relacion)) {
286 +         cout << "\tNo se puede insertar: años de nacimiento INCOHERENTES para la relación indicada.\n";
287 +         delete nuevo; // Se libera memoria
288 +         return;
289 +     }
290 +
291 +     //Si todo es válido, se inserta el nodo en el árbol con la relación adecuada
292 +     insertarFlexible(raiz, nuevo, generaciones, idRelacionado, relacion);
293 +

```

- Napan Hernández Luisana Carolina

<p>Commits on Jun 25, 2025</p> <div style="border: 1px solid #ccc; padding: 5px;"> Evitar Duplicidad de ID Estructura LuisanaNapan authored 10 minutes ago Verified 41e5d4f Copy View </div> <p>Commits on Jun 22, 2025</p> <div style="border: 1px solid #ccc; padding: 5px;"> Ordenando Estructura LuisanaNapan authored 4 days ago Verified 8688668 Copy View </div> <div style="border: 1px solid #ccc; padding: 5px;"> Validando relacion LuisanaNapan authored 4 days ago Verified c4d0b38 Copy View </div> <p>Commits on Jun 21, 2025</p> <div style="border: 1px solid #ccc; padding: 5px;"> Update Estructura LuisanaNapan authored 4 days ago Verified 9564cee Copy View </div>	<p>Commits on Jun 19, 2025</p> <div style="border: 1px solid #ccc; padding: 5px;"> Delete README.md LuisanaNapan authored last week Verified 1b54b4f Copy View </div> <div style="border: 1px solid #ccc; padding: 5px;"> I Estructura LuisanaNapan authored last week Verified 2d95e50 Copy View </div> <p>Commits on Jun 14, 2025</p> <div style="border: 1px solid #ccc; padding: 5px;"> Rename EstructuralInicial to Estructura LuisanaNapan authored 2 weeks ago Verified 17e10b5 Copy View </div> <div style="border: 1px solid #ccc; padding: 5px;"> Luisana LuisanaNapan authored 2 weeks ago Verified a0f01ab Copy View </div> <div style="border: 1px solid #ccc; padding: 5px;"> Initial commit LuisanaNapan authored 2 weeks ago Verified f04c6cc Copy View </div>
---	---

```

diff --git a/Estructura.cpp b/Estructura.cpp
@@ -261,32 +261,32 @@ int main() {
    int id, nac;
    string nombre, fallecido;
    cout << "ID (DNI): "; cin >> id;
    cout << "Nombre: "; cin >> nombre;
    cout << "Año de nacimiento: "; cin >> nac;
    cout << "Fallecimiento (o 'Vivo'): "; cin >> fallecido;
    cout << "\tID (DNI): "; cin >> id;
    cout << "\tNombre: "; cin >> nombre;
    cout << "\tAño de nacimiento: "; cin >> nac;
    cout << "\tFallecimiento (o 'Vivo'): "; cin >> fallecido;
    Nodo* nuevo = crearNodo(id, nombre, nac, fallecido);
    raiz = nuevo;
    cout << "Primer miembro insertado como raiz.\n";
    cout << "\tPrimer miembro insertado como raiz.\n";
} else {
    int idRelacionado, relacion;
    cout << "Ingrese el ID del familiar relacionado: ";
    cout << "\tIngrese el ID del familiar relacionado: ";
    cin >> idRelacionado;
    cout << "\n\tQué relación tiene con " << relativo->nombre << "?\n";
    cout << "\t1. Padre/Madre\n";
    cout << "\t2. Hijo/a izquierdo/a\n";
    cout << "\t3. Hijo/a derecho/a\n";
    cout << "\t4. Hermano/a izquierdo/a\n";
    cout << "\t5. Hermano/a derecho/a\n";
    cout << "Opción: ";
    cin >> relacion;
} else {
    int id, nac;
    string nombre, fallecido;
    cout << "ID (DNI): "; cin >> id;
    cout << "Nombre: "; cin >> nombre;
    cout << "Año de nacimiento: "; cin >> nac;
    cout << "Fallecimiento (o 'Vivo'): "; cin >> fallecido;
    cout << "\tID (DNI): "; cin >> id;
    cout << "\tNombre: "; cin >> nombre;
    cout << "\tAño de nacimiento: "; cin >> nac;
    cout << "\tFallecimiento (o 'Vivo'): "; cin >> fallecido;
}

```

```

158 + //imprimir recorrido con datos completos
159 + void imprimirFicha(Nodo* nodo, const string& tipo) {
160 +     cout << "\t-----\n";
161 +     cout << "\tTipo : " << tipo << "\n";
162 +     cout << "\tNombre : " << nodo->nombre << "\n";
163 +     cout << "\tID (DNI) : " << nodo->id << "\n";
164 +     cout << "\tNacimiento: " << nodo->nacimiento << "\n";
165 +     cout << "\tFallecido : " << nodo->fallecimiento << "\n";
166 +
167
159 168     // Recorridos
160 - void inorden(Nodo* nodo) {
161 +     void inorden(Nodo* nodo, string tipo = "RAIZ") {
162 +         if (!nodo) return;
163 -         inorden(nodo->izq);
164 -         cout << nodo->nombre << " (" << nodo->id << ")\n";
165 -         inorden(nodo->der);
166 +         inorden(nodo->izq, "Hijo Izquierdo");
167 +         imprimirFicha(nodo, tipo);
168 +         inorden(nodo->der, "Hijo Derecho");
169 }
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247 + bool validarAnio(Nodo* nuevo, Nodo* relativo, int relacion){
248 +     int diferencia;
249 +
250 +     switch (relacion) {
251 +         case 1: { // Padre / madre
252 +             diferencia = relativo->nacimiento - nuevo->nacimiento;
253 +             if (diferencia < 12) { //si la diferencia de la edad del hijo(relativo) - la edad de la raiz(nuevo) es menor a 12...
254 +                 cout << "\tError: Es muy joven para ser padre/madre de " << relativo->nombre << ".\n";
255 +                 return false;
256 +             }
257 +             return true; //continua y valida la edad en caso de que el padre si sea mayor o igual a 12 años
258 +         }
259 +         //si la edad del nuevo ingresante - el del relativo es menor a 12 también decimos que al diferencia de edad no es valida
260 +         case 2: // Hijo izquierdo
261 +         case 3: { // Hijo derecho
262 +             diferencia = nuevo->nacimiento - relativo->nacimiento;
263 +             if (diferencia < 12) {
264 +                 cout << "\n\tError: muy poca diferencia de edad para ser hijo/a de " << relativo->nombre << ".\n";
265 +                 return false;
266 +             }
267 +             return true; //si la edad es valida continuamos
268 +         }
269 +         //la diferencia de edad entre el nuevo miembro y la Raiz es mayor a 48
270 +         case 4: // Hermano izquierdo
271 +         case 5: { // Hermano derecho
272 +             diferencia = nuevo->nacimiento - relativo->nacimiento;
273 +             if (diferencia > 48) { //decimos que hay un error ya que no siempre la edad es tan diferente
274 +                 cout << "\n\tError: demasiada diferencia de edad para ser hermanos.\n";
275 +                 return false;
276 +             }
277 +             return true; //si está dentro del rango entonces continuamos
278 +         }
279 +
280 +         default: //si no cumple ninguno de esos casos decimos que no es valida la relacion
281 +             cout << "\tRelación no válida.\n";
282 +             return false;
283 +     }
284 + }

```

```

335 +             cout << "\n\t1 -> Padre/Madre\n";
336 +             cout << "\t2 -> Hijo/a izquierdo/a\n";
337 +             cout << "\t3 -> Hijo/a derecho/a\n";
338 +             cout << "\t4 -> Hermano/a izquierdo/a\n";
339 +             cout << "\t5 -> Hermano/a derecho/a\n";
340 +             cout << "\n\tDígitte un número de opción: ";
341 341             cin >> relacion;
342 342
343 343             // Verifica si excede generaciones antes de pedir datos
344 344             int nivelActual = calcularNivel(relativo);
345 345             if ((relacion == 1 && nivelActual >= generaciones) || // Insertar padre sube una generación
346 346                 ((relacion == 2 || relacion == 3) && nivelActual + 1 > generaciones) || // Insertar hijo si sube
347 347                 ((relacion == 4 || relacion == 5) && nivelActual > generaciones)) { // Hermano está al mismo nivel
348 -             cout << "No se puede insertar: excede las generaciones permitidas.\n";
348 +             cout << "\tNo se puede insertar: excede las generaciones permitidas.\n";
349 349             break;
350 350         }
351 351
352 352             // Ahora si pide los datos del nuevo miembro
353 353             int id, nac;
354 354             string nombre, fallecido;
355 355
356 -             cout << "\tID (DNI): "; cin >> id;
357 -             cout << "\tNombre: "; cin >> nombre;
358 -             cout << "\tAño de nacimiento: "; cin >> nac;
356 +             cout << "\tID (DNI)\t\t: "; cin >> id;
357 +             cout << "\tNombre\t\t\t: "; cin >> nombre;
358 +             cout << "\tAño de nacimiento\t\t: "; cin >> nac;
359 359             cout << "\tFallecimiento (o 'Vivo'): "; cin >> fallecido;
360 360
361 361             Nodo* nuevo = crearNodo(id, nombre, nac, fallecido);
362 +             if (!validarAnio(nuevo, relativo, relacion)) {
363 +                 cout << "\tNo se puede insertar: años de nacimiento INCOHERENTES para la relación indicada.\n";
364 +                 break;
365 +             }
366 --- -----
413 -             case 1: inorden(raiz); break;
414 -             case 2: preorden(raiz); break;
415 -             case 3: postorden(raiz); break;
416 -             case 4: porNiveles(raiz); break;
417 -             default: cout << "Opción no válida.\n"; break;
416 +
417 +             case 1:
418 +                 cout << "\n\t----- RECORRIDO EN INORDEN -----";
419 +                 inorden(raiz); break;
420 +             case 2:
421 +                 cout << "\n\t----- RECORRIDO EN PREORDEN -----";
422 +                 preorden(raiz); break;
423 +             case 3:
424 +                 cout << "\n\t----- RECORRIDO EN POSTORDEN -----";
425 +                 postorden(raiz); break;
426 +             case 4:
427 +                 cout << "\n\t----- RECORRIDO POR NIVELES -----";
428 +                 porNiveles(raiz); break;
428 +             default: cout << "\n\tOpción no válida.\n"; break;
38 + // Buscar miembro por ID
39 + Nodo* buscar(Nodo* actual, int id) {
40 +     if (actual == NULL || actual->id == id) return actual;
41 +     return (id < actual->id) ? buscar(actual->izq, id) : buscar(actual->der, id);
42 + }
43 +
44 + //si el ID existe, primero buscamos el id
45 + bool idExiste(int id) {
46 +     return buscar(raiz, id) != NULL;
47 + }
48 +
49 + //Validamos el ID
50 + void pedirID(int& id) {
51 +     do {
52 +         cout << "\n\tID (DNI de 8 dígitos)\t: ";
53 +         cin >> id;
54 +         if (id < 10000000 || id > 99999999)
55 +             cout << "\tERROR: El DNI debe tener 8 dígitos.\n";
56 +         else if (!idExiste(id))
57 +             cout << "\tERROR: Ya existe un miembro con ese ID.\n";
58 +         else
59 +             break;
60 +     } while (true);
61 + }

```

- Valdivia Blas Yatzuri Xiomara

Se compromete

The screenshot shows a GitHub repository interface. At the top, there's a dropdown menu set to 'principal' and a filter button 'Todo el tiempo'. Below this, a commit history is displayed under two sections: 'Se compromete el 25 de junio de 2025' and 'Se compromete el 16 de junio de 2025'. Each section contains several commits from the user 'YatzuriVB'. The commits are listed with their commit hash, author, date, and a snippet of the commit message.

Section	Commit Hash	Author	Date	Message Snippet
Se compromete el 25 de junio de 2025	6869634	YatzuriVB	hace 8 minutos	Comentarios-Descendientes
	716c518	YatzuriVB	hace 3 minutos	Comentarios de los antepasados
	88b0ede	YatzuriVB	hace 8 minutos	Comentarios Buscar por nombres
	6bac45a	YatzuriVB	hace 14 horas	Mejorando Consultas
Se compromete el 16 de junio de 2025	57ffd83	YatzuriVB	la semana pasada	AVANCE
	384a882	YatzuriVB	la semana pasada	Avance

The screenshot shows a GitHub pull request with a code diff view. The code is written in C++ and includes various annotations such as '+' for additions, '-' for deletions, and '++' for renames. The code is organized into several functions: 'Avance', 'Mejorando Consultas', 'InsertarFlexible', 'mostrarAncestros', and 'mostrarAncestros'. The code is annotated with comments like '//Buscar Miembro por nombre', '// Insertar miembro', and '// Ancestros'.

```

class Avance {
public:
    void buscarPorNombre(Nodo* nodo, string nombre) {
        if (!nodo) return NULL;
        if (nodo->nombre == nombre) return nodo;
        Nodo* izq = buscarPorNombre(nodo->izq, nombre);
        if (izq) devuelve izq;
        return buscarPorNombre(nodo->der, nombre);
    }
};

void insertarFlexible(Nodo*& raiz, Nodo* nuevo, int generacionesMax, int idRelacionado, int relacion) {
    if (!raiz) {
        raiz = nuevo;
        return;
    }
    if (nuevo->id < raiz->id) {
        insertarFlexible(raiz->izq, nuevo, generacionesMax, idRelacionado, relacion);
    } else {
        insertarFlexible(raiz->der, nuevo, generacionesMax, idRelacionado, relacion);
    }
}

void mostrarAncestros(Nodo* nodo) {
    while (nodo) {
        cout << nodo->nombre << " (" << nodo->id << ")\n";
        nodo = nodo->padre;
    }
}

void mostrarAncestros(Nodo* ancestro, Nodo* hijoOriginal) {
    if (ancestro == NULL) {
        cout << "\n\tEste nodo no tiene ancestros.\n";
        return;
    }
    int nivel = 1;
    while (ancestro) {
        cout << "\n\tNivel " << nivel << " - Ancestro:\n";
        imprimirFicha(ancestro, "Ancestro");
        nivel++;
        ancestro = ancestro->padre;
    }
    int diferencia = (hijoOriginal->nacimiento > ancestro->nacimiento)
                    ? (hijoOriginal->nacimiento - ancestro->nacimiento)
                    : (ancestro->nacimiento - hijoOriginal->nacimiento);
    cout << "\tDiferencia de nacimiento con respecto a " << hijoOriginal->nombre
        << ": " << diferencia << " años\n";
}

```

```

242 +     }
243 +     ancestro = ancestro->padre;
244 +     nivel++;
220 245     }
221 246   }
222 247
223 248   // Descendientes (en preorden)
224 249   void mostrarDescendientes(Nodo* nodo) {
225 250     if (!nodo) return;
226 -     cout << nodo->nombre << "(" << nodo->id << ")\n";
227 -     mostrarDescendientes(nodo->iizq);
228 -     mostrarDescendientes(nodo->der);
251 +
252 +     // Verificamos si el nodo no tiene hijos
253 +     if (nodo->iizq == NULL && nodo->der == NULL) {
254 +       cout << "\n\tEste nodo no tiene hijos (hojas).\n";
255 +       return;
256 +     }
257 +
258 +     // Si tiene hijos, los mostramos con sus respectivos recorridos
259 +     if (nodo->iizq) {
260 +       imprimirFicha(nodo->iizq, "Descendiente (izquierdo)");
261 +       mostrarDescendientes(nodo->iizq);
262 +     }
263 +     if (nodo->der) {
264 +       imprimirFicha(nodo->der, "Descendiente (derecho)");
265 +       mostrarDescendientes(nodo->der);
266 +     }
457 +
398 458           break;
399 459         }
400 460
401 461         case 5: {
402 -           int id;
403 +           cout << "\tID para ver descendientes: ";
404 -           cin >> id;
405 -           Nodo* n = buscar(raiz, id);
406 -           if (n) mostrarDescendientes(n);
407 -           else cout << "\tNo encontrado.\n";
462 +
463 +           int tipoBusqueda;
464 +           cout << "\n\tCómo desea buscar al miembro?\n";
465 +           cout << "\t[1] Por ID\n";
466 +           cout << "\t[2] Por nombre\n";
467 +           cout << "\tSeleccione una opción: ";
468 +           cin >> tipoBusqueda;
469 +
470 +           Nodo* n = NULL;
471 +           if (tipoBusqueda == 1) {
472 +             int id;
473 +             cout << "\tIngrese ID para ver descendientes: ";
474 +             cin >> id;
475 +             n = buscar(raiz, id);
476 +           } else if (tipoBusqueda == 2) {
477 +             string nombre;
478 +             cout << "\tIngrese nombre exacto para ver descendientes: ";
479 +             cin >> nombre;
480 +             n = buscarPorNombre(raiz, nombre);
481 +           } else {
482 +             cout << "\tOpción inválida.\n";
483 +             break;
484 +
485 +             if (n)
486 +               mostrarDescendientes(n);
487 +             else
488 +               cout << "\tmíembro no encontrado.\n";
489 +
488 490           break;
489 491         }
410 492

```

1 archivo cambiado + 8 - 4 líneas cambiadas

```
▼ Estructura □ ⌂
```

...	@@ -488,21 +488,25 @@ void mostrarAncestros(Nodo* ancestro, Nodo* hijoOriginal) {
488 488	}
489 489	}
490 490	
491	- // Descendientes (en preorder)
491	+ //Descendientes (en preorder)
492 492	void mostrarDescendientes(Nodo* nodo) {
493	- if (!nodo) return;
493	+
494	+ if (!nodo) return; // Caso base: si el nodo es nulo, salimos de la función
494 495	
495	- // Verificamos si el nodo no tiene hijos
496	+ // Si el nodo es hoja, mostramos esto
496 497	if (nodo->izq == NULL && nodo->der == NULL) {
497 498	cout << "\n\tEste nodo no tiene hijos (hojas).\n";
498 499	return;
499 500	+ }
500 501	
501	- // Si tiene hijos, los mostramos con sus respectivos recorridos
502	+ // Si tiene hijo izquierdo, mostramos su información
503	+ // y llamamos recursivamente para buscar más descendientes desde allí
502 504	if (nodo->izq) {
503 505	imprimirFicha(nodo->izq, "Descendiente (izquierdo);
504 506	mostrarDescendientes(nodo->izq);
505 507	}
508	+
509	+ // Si tiene hijo derecho, lo mostramos e iniciamos recorrido desde ese hijo
461	- // Ancestros
461	+ //Ancestros
462 462	void mostrarAncestros(Nodo* ancestro, Nodo* hijoOriginal) {
463	- if (ancestro == NULL) {
463	+ if (ancestro == NULL) {// Si no hay ancestro
464 464	cout << "\n\tEste nodo no tiene ancestros registrados.\n";
465 465	return;
466 466	}
467 467	
468	- int nivel = 1;
468	+ int nivel = 1; // Contador de generaciones ascendentes
469	+
470	+ // Recorremos hacia arriba por el árbol mientras existan padres
469 471	while (ancestro) {
472	+ // Mostramos el nivel generacional y los datos del ancestro actual
470 473	cout << "\n\tNivel " << nivel << " - Ancestro:\n";
471 474	imprimirFicha(ancestro, "Ancestro");
472 475	
476	+ // Calculamos la diferencia de nacimiento entre el ancestro y el miembro consultado
473 477	int diferencia = (hijoOriginal->nacimiento > ancestro->nacimiento)
474	- ? (hijoOriginal->nacimiento - ancestro->nacimiento)
475	- : (ancestro->nacimiento - hijoOriginal->nacimiento);
478	+ ? (hijoOriginal->nacimiento - ancestro->nacimiento)
479	+ : (ancestro->nacimiento - hijoOriginal->nacimiento);
480	+
481	+ // Mostramos esa diferencia de años

- Yauri Ancassi Rosa

Commits

main

Commits on Jun 25, 2025

Mejora de eliminación

riksa authored 22 minutes ago

Verified 51bd1e0 ⌂ ↗

Commits on Jun 21, 2025

Eliminar Miembro Genealogico -Yauri Anccasi Rosa

riksa authored 4 days ago

Verified 831d3a0 ⌂ ↗

estructura.h

```
112 113 }  
114  
115 // Eliminar nodo  
116 void* eliminarNodo(void* raiz, int id) {  
117     if (!raiz) return NULL;  
118     if (id == raiz->id) {  
119         raiz->sig = eliminarNodo(raiz->sig, id);  
120     } else if (id > raiz->id) {  
121         raiz->sig = eliminarNodo(raiz->sig, id);  
122     } else if (id < raiz->id) {  
123         raiz->sig = eliminarNodo(raiz->sig, id);  
124     }  
125     void* actual = insertarRaiz(raiz, id); // 1. Buscar al miembro a eliminar por su ID  
126     if (actual) {  
127         cout << "Miembro no encontrado.\n"; // 2. Si no lo encuentra, muestra mensaje  
128         return; // 3.  
129     }  
130  
131     Nodo* padre = actual->padre; // 3. Guardemos al padre del nodo a eliminar (si tiene)  
132  
133     // 4. Si tiene un padre (NO es la raiz)  
134     if (padre) {  
135         if (actual->id == padre->id) { // 5. Si el nodo es el hijo izquierdo del actualizado como nuevo hijo del padre  
136             if (actual->sig == actual->sig->actual->id) { // 6. Se coloca en la posicion de su hermano  
137                 padre->sig = actual->sig->actual->id; // 7. Se coloca en la posicion de su hermano  
138                 if (actual->id == padre->id) { // 8. Si el hijo derecho  
139                     padre->id = actual->id; // 9. Se coloca en la posicion de su hermano  
140                     padre->id = actual->id; // 10. Se coloca en la posicion de su hermano  
141                     if (actual->id == padre->id) { // 11. Si el hermano es el mismo  
142                         if (actual->id == padre->id) { // 12. Lo colocamos alli  
143                             if (actual->id == actual->id->id) { // 13. Si tiene hermano izquierdo, el hermano sera la raiz  
144                                 if (actual->id == actual->id) { // 14. Si tiene hermano derecho, lo colocamos como hermano izquierdo  
145                                     if (actual->id == actual->id) { // 15. Finalmente, eliminamos el nodo  
146                                         cout << "Miembro eliminado correctamente (con reorganizacion).\n";  
147                                         delete actual; // 16. Si no tiene hijos, el actual queda vacio  
148                                         raiz = NULL; // 16. Si no tiene hijos, el actual queda vacio  
149                                         return raiz;  
150         }  
151         delete actual; // 15. Finalmente, eliminamos el nodo  
152         cout << "Miembro eliminado correctamente (con reorganizacion).\n";  
153     }  
154 }  
155  
156 // Recuperar  
157 void* insertarRaiz(void* nodo) {  
158     if (!nodo) return;  
159  
160     if (nodo->id > id) {  
161         if (nodo->sig) {  
162             insertarRaiz(nodo->sig);  
163         }  
164         nodo->id = id; // 10. Si no tiene hijos, el actual queda vacio  
165         cout << "ID a eliminar: ";  
166         cin >> id;  
167         cout << "Nuevo ID: ";  
168         raiz = eliminarNodo(raiz, id);  
169         cout << "Eliminado el elemento.\n";  
170         return raiz;  
171     }  
172     if (nodo->id < id) {  
173         if (nodo->sig) {  
174             insertarRaiz(nodo->sig);  
175         }  
176         nodo->id = id; // 11. Si no tiene hijos, el actual queda vacio  
177         cout << "ID a eliminar (por rellenable): ";  
178         cin >> id;  
179         cout << "Nuevo ID: ";  
180         eliminarNodoGenealogico(raiz, id);  
181         return;  
182     }  
183 }
```

```
518 +         case 3: {
519 +             int tipoEliminacion;
520 +             cout << "Introduce el tipo de eliminación:\n";
521 +             cout << "\t[1] Eliminar por ID\n";
522 +             cout << "\t[2] Eliminar por nivel\n";
523 +             cout << "\t[3] Eliminar por nombre\n";
524 +             cin >> tipoEliminacion;
525 +
526 +             if (tipoEliminacion == 1) {
527 +                 int id;
528 +                 cout << "\tID a eliminar: ";
529 +                 cin >> id;
530 +                 eliminarMiembro(raiz, id);
531 +             }
532 +             else if (tipoEliminacion == 2) {
533 +                 int nivel;
534 +                 cout << "\tNivel a eliminar: ";
535 +                 cin >> nivel;
536 +                 eliminarPorNivel(raiz, nivel);
537 +             }
538 +             else {
539 +                 cout << "\tOpción inválida\n";
540 +             }
541 +         }
542 +     }
```

· Enlace a la herramienta colaborativa:

GitHub:

<https://github.com/LuisanaNapan/ArbolGenealogico.git>

Presentación en Canva:

https://www.canva.com/design/DAGpHx0bKrc/DhV355nrDJIAU4BKCI_Pgg/view?utm_content=DAGpHx0bKrc&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlld=h6adaed0774