

Universidad Continental

Aprendizaje Basado en Retos - Desarrollo de un árbol genealógico

Kelmer W. Obregon, Brayddy B. Beltran y Diego A. Quispe

Facultad de Ingeniería, Universidad Continental

NRC 59098: Estructura de Datos

YESENIA CONCHA RAMOS

20 de diciembre de 2025

Tabla de contenido

RESUMEN	3
INTRODUCCIÓN	3
CAPÍTULO 1 - FASE DE IDEACIÓN	4
1. Descripción del problema	4
2. Requerimientos del sistema	5
3. Respuesta a las preguntas guías	5
4. Herramienta colaborativa	7
CAPÍTULO 2 - PROTOTIPO	8
1. Descripción de estructuras de datos y operaciones:	8
2. Algoritmos principales	9
Pseudocódigo para crear un árbol binario	9
Pseudocódigo para realizar el recorrido de un árbol	12
3. Diagramas de flujo del pseudocódigo	15
Pseudocódigo para crear un árbol binario.	15
Diagrama de ordenamiento	18
4. Avance del código fuente	20
CAPÍTULO 3 – SOLUCIÓN FINAL	25
1. Código limpio, bien comentado y estructurado.	25
2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos	25
CAPÍTULO 4 – EVIDENCIAS DE TRABAJO COLABORATIVO	25
Repositorio con Control de Versiones (Capturas de Pantalla)	25

Registro de commits claros y significativos que evidencien aportes individuales (proactividad).	25
Historial de ramas y fusiones si es aplicable	26
Evidencia por cada integrante del equipo	26
Enlace a la herramienta colaborativa	27
Evidencias trabajo colaborativo	27
Manual para usuario (= En progreso =)	28
Acta de compromiso (= En progreso =)	28
CONCLUSIONES	28
REFERENCIAS	28
ANEXOS (OPCIONAL)	29

RESUMEN

El informe desarrolla un sistema de árbol genealógico donde implementamos el C++, utilizando estructuras dinámicas basado en nodo enlazado, el objetivo del proyecto es modelar y representar de manera genealogía de la casa borja-loyola-inca, un linaje que se encuentra en el periodo virreinal del Perú

El sistema permite registrar miembros, establecer relaciones de parentesco como: padre, madre, hijos, hermanos. podremos ver la estructura genealógica, realizar búsquedas y recorrer la información en distintos sentidos, además para construir nuestro prototipo empleamos estructuras del tipo árbol, listas enlazadas horizontales, algunos algoritmos de inserción, recorrido, búsqueda y control de ciclos para evitar errores, asimismo utilizamos GitHub para el trabajo colaborativo, que nos permitió registrar y poner nuestras contribuciones, gestionar y documentar nuestro avances, aparte de la evidencia del trabajo

INTRODUCCIÓN

En esta sección se presenta una descripción general del reto, el objetivo del informe y la relevancia del uso de árboles binarios en la solución de problemas relacionados con estructuras jerárquicas.

Los árboles genealógicos son estructuras que utilizamos para representar las relaciones familiares y vínculos de ascendencia, desde el punto de vista de las estructuras de datos

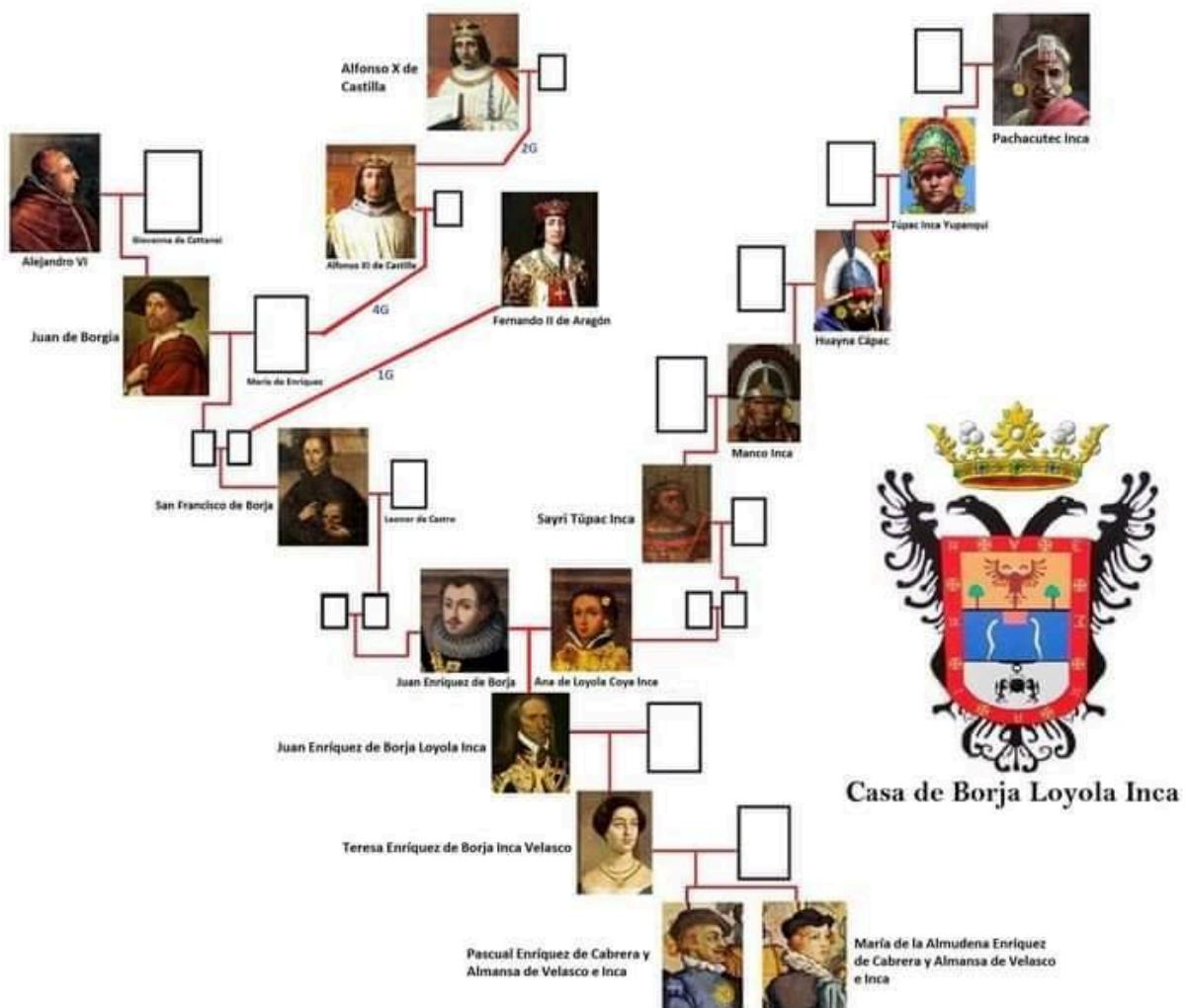
Nuestro proyecto, desarrollado en el curso de *Estructura de Datos*, tiene como objetivo implementar un sistema que permita modelar el árbol genealógico de la Casa *Borja-Loyola-Inca* utilizando C++. La propuesta utiliza un árbol general, donde cada miembro posee punteros hacia su padre, madre, primer hijo y siguiente hermano. Esto permite manejar descendencias y relaciones. Además, se desarrollaron algoritmos específicos para inserción, búsqueda, visualización y control de ciclos, garantizando recorridos seguros de la estructura.

CAPÍTULO 1 - FASE DE IDEACIÓN

1. Descripción del problema

(Explicación del contexto del reto para generar el árbol genealógico de una civilización)

Se realizará una elaboración de un sistema sobre el árbol genealógico de la Casa Borja-Loyola Inca; para una mayor información, entendimiento y visualización de cómo era el vínculo del mestizaje en la nobleza durante la época virreinal en el Perú entre nobleza española e inca



2. Requerimientos del sistema

➤ Funcionales

- ➔ Insertar nuevos miembros (Nombre, Sexo, Edad, Observaciones, Puesto relacional)
- ➔ Eliminar miembros
- ➔ Actualizar y cambiar miembros
- ➔ Determinar relaciones entre miembros (Padre, Madre, Hijos)
- ➔ Generar una ID numérica (int) para cada nodo, esto será usado para realizar los recorridos dentro del árbol genealógico

➤ No funcionales

- ➔ Validaciones de datos para caracteres especiales (ASCII) en string o char
- ➔ Implementar modelo visual para graficar los nodos
- ➔ Optimizacion de codigo en base al tiempo
- ➔ Uso de funciones y tablas Hash

3. Respuesta a las preguntas guías

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

- En los nodos se deben almacenar la información sobre nombre, sexo, edad, observaciones y su puesto en la casa noble.
- Cada nodo tendrá un ID (int), para poder hacer los recorridos binarios

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

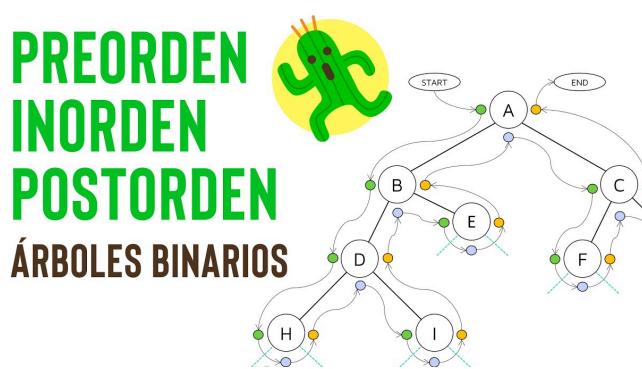
Inserción

- Creación del nodo con los datos del miembro
- La asignación de relaciones, con padre/madre y enlazar el nodo como hijo de ellos

Eliminación

- Si no se encuentra relación con los miembros de la casa se procede a la eliminación

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



- Recorridos clásicos (Recorrido Pre-Orden)

Muestra desde el primer miembro del linaje hasta el último, y en viceversa

- Recorridos genealógicos específicos

Desde un miembro específico hasta sus ascendientes

- Recorridos por Amplitud

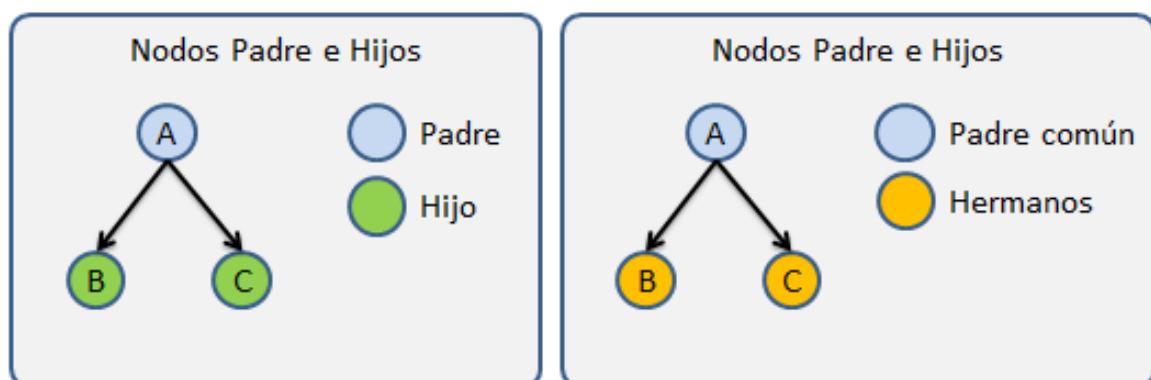
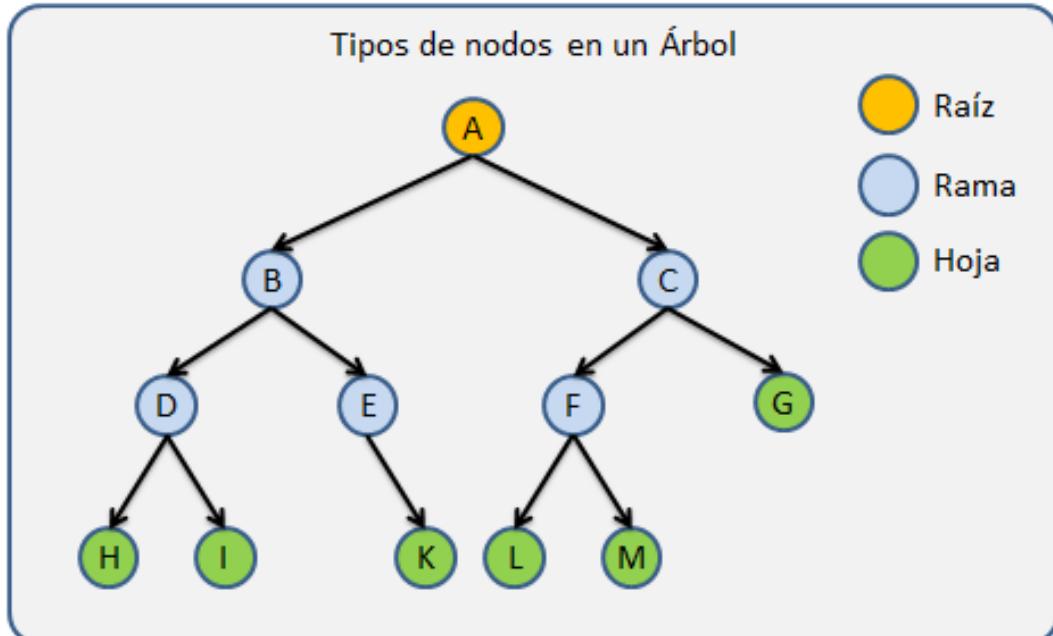
Desde un miembro específico hasta sus ascendientes

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

- Verificando si aparece el ancestro raíz de la rama (por ejemplo, un Inca o un Borja específico)
- Cada rama puede tener un ID o etiqueta, y cuando se agregan descendientes, heredan esa etiqueta

¿Cómo balancear el árbol si se vuelve demasiado profundo?

- Reorganización visual sin alterar relaciones
- Agrupar por generaciones



4. Herramienta colaborativa

Enlace de la herramienta colaborativa utilizada y capturas de pantalla

GITHUB

The screenshot shows a GitHub repository page. At the top, there's a navigation bar with options like 'Pin', 'Watch', 'Fork', and 'Star'. Below the header, it displays 'main' branch, 1 branch, 0 tags, and a search bar for 'Go to file'. A green 'Add file' button is visible. On the right, there's an 'About' section with a brief description: 'Desarrollo de un sistema dinámico por medio de estructura de datos en C++'. It also shows '2 Commits' from 'DiegoQuispeAlmanza' and file details for 'Copia de S13_Trabajo.Grupal.pdf' and 'LICENSE'. A red box highlights the 'README' section, which contains a 'Readme' icon, a 'Add a README' button, and a note: 'Help people interested in this repository understand your project by adding a README.' To the right of the main content area, there are sections for 'Releases' (no releases published) and 'Packages' (no packages published).

The screenshot shows the 'Manage access' page. It includes a 'Select all' checkbox, a 'Type' dropdown set to 'Pending Invite', and a search bar with placeholder text 'Find a collaborator...'. It lists two users: 'brayanuc' (pending invite) and 'KelmerWilder' (collaborator). Each user has a checkbox, a profile picture, a name, a status, and a trash can icon for removal.

CAPÍTULO 2 - PROTOTIPO

1. Descripción de estructuras de datos y operaciones:

struct persona

Tipo	Campo	Funcionalidad
int	id	ID del nodo
string	apenom	Almacenar nombre y apellido
char	sexo	Almacenar sexo ('M' o 'F')
string	obs	Almacenar observaciones de la persona

struct	fecha general	Almacenar la fecha de nacimiento y fallecimiento
struct	rol familia	Almacenar su posición o título dentro de la jerarquía familiar

struct fecha

Tipo	Campo	Funcionalidad	Tipo	Campo	Funcionalidad
int	diaN	Almacenar día Nac.	int	diaF	Almacenar día Fall.
int	mesN	Almacenar mes Nac.	int	mesF	Almacenar mes Fall.
int	anioN	Almacenar año Nac.	int	anioF	Almacenar anio Fall.

struct rol

Tipo	Campo	Funcionalidad
string	título	Almacenar título o rango de nobleza
string	etnia	Almacenar etnia a la que pertenece
string	ciudad	Almacenar ciudad de origen

2. Algoritmos principales

Pseudocódigo para crear un árbol binario

```

Algoritmo InsertarPersona
    Definir nombre, padre, madre Como Cadena
    Definir totalPersonas, i, j, opcion Como Entero
    Definir continuar Como Lógico
    Dimensionar nombre(20)
    Dimensionar padre(20)
    Dimensionar madre(20)
    totalPersonas ← 0
    continuar ← Verdadero
    Escribir '====='
    Escribir ' MODELO ?RBOL GENEAL?GICO'
    Escribir '====='
    Escribir ''
    Mientras continuar Hacer
        Escribir '==> MEN? PRINCIPAL ==>'
        Escribir '1. Agregar persona'
        Escribir '2. Mostrar ?rbol geneal?gico'
        Escribir '3. Buscar familiares de una persona'
        Escribir '4. Salir\n'
        Escribir 'Seleccione una opci?n: ' Sin Saltar
        Leer opcion
        Seg?n opcion Hacer
            Caso 1:
                Si numPersonas<20 Entonces
                    numPersonas ← numPersonas+1
                    Escribir ''
                    Escribir '--- AGREGAR PERSONA ', numPersonas, ' ---'
                    Escribir 'Nombre: ' Sin Saltar
                    Leer nombre[numPersonas]
                    Escribir 'Nombre del padre: ' Sin Saltar
                    Leer padre[numPersonas]
                    Escribir 'Nombre de la madre: ' Sin Saltar
                    Leer madre[numPersonas]
                    Escribir ''
                    Escribir 'Persona agregada exitosamente'
                SiNo
                    Escribir ''
                    Escribir 'Error: L?mite de personas alcanzado (m?ximo 20)'
            FinSi

```

```

-----  

2:  

  Si numPersonas>0 Entonces  

    Escribir ''  

    Escribir '===== ?RBOL GENEAL?GICO COMPLETO ====='  

    Escribir ''  

    Para i<1 Hasta numPersonas Hacer  

      Escribir nombre[i]  

      Si padre[i]!="ninguno" Y padre[i]!="Ninguno" Entonces  

        Escribir 'Padre: ', padre[i]  

      FinSi  

      Si madre[i]!="ninguno" Y madre[i]!="Ninguno" Entonces  

        Escribir ' Madre: ', madre[i]  

      FinSi  

      // Buscar hijos  

      Escribir ' Hijos:'  

      Definir tieneHijos Como Lógico  

      tieneHijos ← Falso  

      Para j<1 Hasta numPersonas Hacer  

        Si padre[j]=nombre[i] O madre[j]=nombre[i] Entonces  

          Escribir ' ', nombre[j]  

          tieneHijos ← Verdadero  

        FinSi  

      FinPara  

      Si NO tieneHijos Entonces  

        Escribir ' (ninguno)'  

      FinSi  

      Escribir ''  

    FinPara  

  SiNo  

    Escribir 'No hay personas registradas en el ?rbol'  

  FinSi  

-----  

3:  

  Si numPersonas>0 Entonces  

    Definir nombreBuscar Como Cadena  

    Definir encontrado Como Lógico  

    encontrado ← Falso  

    Escribir ''  

    Escribir 'Nombre de la persona a buscar: ' Sin Saltar  

    Leer nombreBuscar  

    Para i<1 Hasta numPersonas Hacer  

      Si nombre[i]=nombreBuscar Entonces  

        encontrado ← Verdadero  

        Escribir ''  

        Escribir '===== INFORMACI?N FAMILIAR DE ', nombreBuscar  

        Escribir '===== '  

        Escribir ''  

        // Mostrar padres  

        Si padre[i]!="ninguno" Y padre[i]!="Ninguno" Entonces  

          Escribir 'Padre: ', padre[i]  

          Escribir 'Madre: ', madre[i]  

        SiNo  

          Escribir 'Padre: (no registrado)'  

        FinSi  

        Si madre[i]!="ninguno" Y madre[i]!="Ninguno" Entonces  

        SiNo

```

```

    Para j<1 Hasta numPersonas Hacer
        Si i≠j Entonces
            Si (padre[i]=padre[j] Y padre[i]≠'ninguno') O (madre[i]
                Escribir ' ', nombre[j]
                tieneHermanos ← Verdadero
            FinSi
        FinSi
    FinPara
    Si NO tieneHermanos Entonces
        Escribir ' (ninguno)'
    FinSi
    // Mostrar hijos
    Escribir ''
    Escribir 'Hijos:'
    Definir tieneHijos Como Lógico
    tieneHijos ← Falso
    Para j<1 Hasta numPersonas Hacer
        Si padre[j]=nombre[i] O madre[j]=nombre[i] Entonces
            Escribir ' ', nombre[j]
            tieneHijos ← Verdadero
        FinSi
    FinPara
    Si NO tieneHijos Entonces
        Escribir ' (ninguno)'
    FinSi
    FinSi
    FinPara
    Si NO encontrado Entonces
        Escribir ''
        Escribir '? Persona no encontrada en el ?rbol geneal?gico'
    FinSi
    SiNo
        Escribir ''
        Escribir '? No hay personas registradas en el ?rbol'
    FinSi
4:
    Escribir ''
    Escribir '?Gracias por usar el sistema!'
    continuar ← Falso
De Otro Modo:
    Escribir ''
    Escribir 'Opc?n inv?lida. Intente nuevamente.'
FinSeg?n
FinMientras
FinAlgoritmo

```

Pseudocódigo para realizar el recorrido de un árbol

```

1 Algoritmo CasaBorjaLoyolaInca
2
3   // DECLARACIÓN DE VARIABLES Y ARREGLOS
4   Definir nombre, titulo, apellido Como Carácter
5   Definir izq, der Como Entero
6   // Dimensionamos los 5 arreglos
7   Dimension titulo[10]
8   Dimension nombre[10]
9   Dimension apellido[10]
10  Dimension izq[10]
11  Dimension der[10]
12
13  // DEFINICIÓN DEL ÁRBOL - Linaje Borja-Loyola Inca
14  // [1] Don Fernando Borja Inca (Abuelo/Raíz) - Representa la unión de linajes
15  titulo[1] = "Don"
16  nombre[1] = "Fernando"
17  apellido[1] = "Borja-Amaru" // Borja + Amaru (Serpiente, símbolo inca)
18  izq[1] = 2
19  der[1] = 3
20
21  // [2] Doña Isabel (Hija) - Linaje fuerte Borja
22  titulo[2] = "Doña"
23  nombre[2] = "Isabel"
24  apellido[2] = "Loyola"
25  izq[2] = 4
26  der[2] = 5
27
28  // [3] Don Cristóbal (Hijo) - Linaje fuerte Inca
29  titulo[3] = "Curaca" // Título tradicional andino
30  nombre[3] = "Cristóbal"
31  apellido[3] = "Tito Cusi"
32  izq[3] = 6
33  der[3] = 0
34
35  // [4] Fray Francisco (Nieto) - Orientación religiosa
36  titulo[4] = "Fray"
37  nombre[4] = "Francisco"
38  apellido[4] = "Borja"
39  izq[4] = 0

```

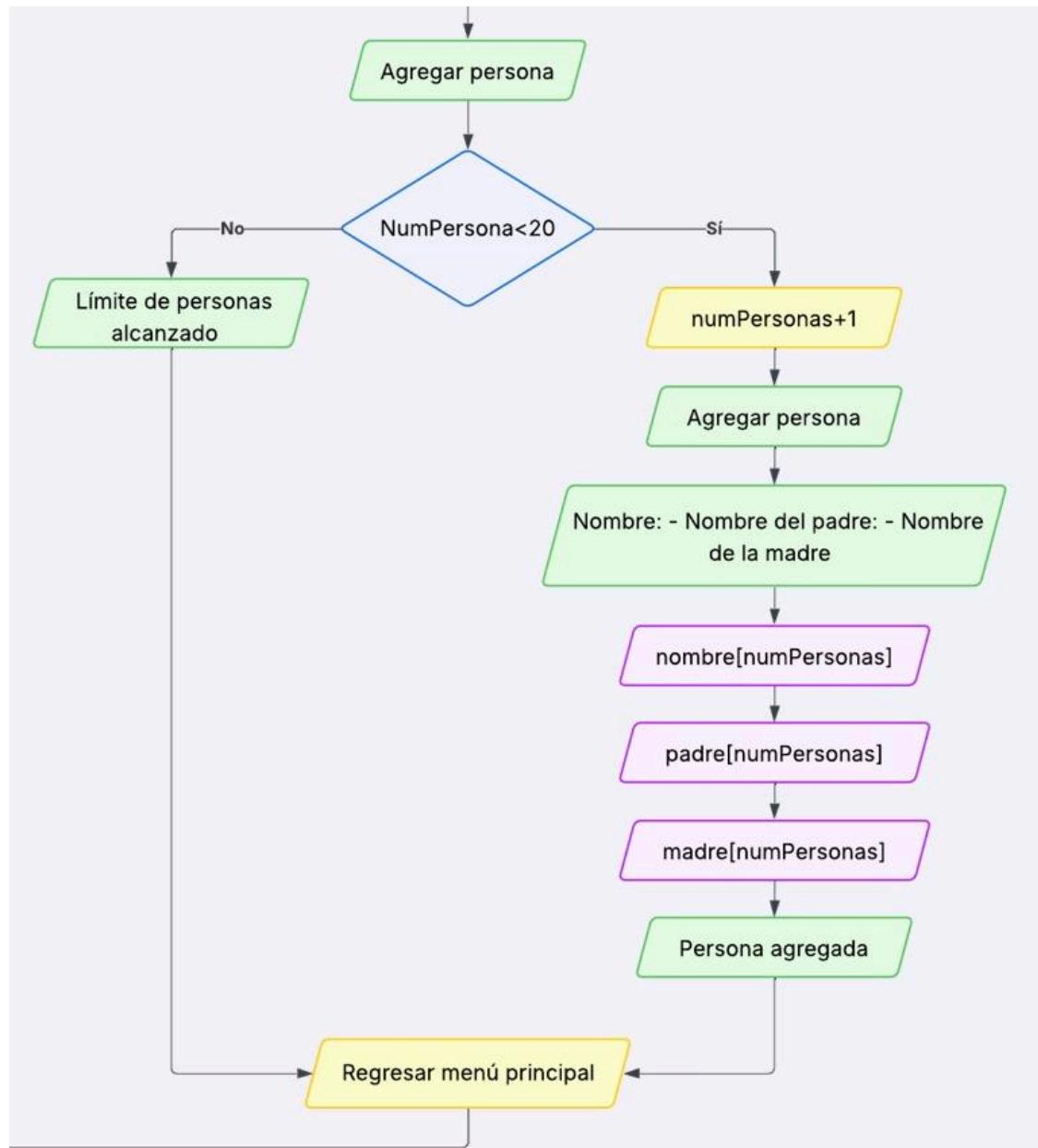
```
41 // [5] Doña Mencia (Nieta) - Linaje Loyola
42 titulo[5] = "Doña"
43 nombre[5] = "Mencia"
44 apellido[5] = "Loyola"
45 izq[5] = 0
46 der[5] = 0
47
48
49 // [6] Don Túpac (Nieta) - Linaje Túpac
50 titulo[6] = "Don"
51 nombre[6] = "Túpac"
52 apellido[6] = "Huari"
53 izq[6] = 0
54 der[6] = 0
55
56 // === RECORRIDOS ===
57 Escribir "--- RECORRIDO PREORDEN (Casa Borja-Loyola Inca) ---"
58 Preorden(1, titulo, nombre, apellido, izq, der)
59 Escribir ""
60
61 Escribir "--- RECORRIDO INORDEN (Casa Borja-Loyola Inca) ---"
62 Inorden(1, titulo, nombre, apellido, izq, der)
63 Escribir ""
64
65 Escribir "--- RECORRIDO POSTORDEN (Casa Borja-Loyola Inca) ---"
66 Postorden(1, titulo, nombre, apellido, izq, der)
67 Escribir ""
68
69 FinAlgoritmo
```

```
vv  * Lenguaje de alto nivel
70 // DEFINICIÓN DE SUBPROCESOS
71
72 SubProceso Preorden(n, tit, nom, ape, i, d)
73   Si n ≠ 0 Entonces
74     Escribir Sin Saltar tit[n], " ", nom[n], " ", ape[n], " | "
75     Preorden(i[n], tit, nom, ape, i, d)
76     Preorden(d[n], tit, nom, ape, i, d)
77   FinSi
78 FinSubProceso
79
80
81
82 SubProceso Inorden(n, tit, nom, ape, i, d)
83   Si n ≠ 0 Entonces
84     Inorden(i[n], tit, nom, ape, i, d)
85     Escribir Sin Saltar tit[n], " ", nom[n], " ", ape[n], " | "
86     Inorden(d[n], tit, nom, ape, i, d)
87   FinSi
88 FinSubProceso
89
90
91 SubProceso Postorden(n, tit, nom, ape, i, d)
92   Si n ≠ 0 Entonces
93     Postorden(i[n], tit, nom, ape, i, d)
94     Postorden(d[n], tit, nom, ape, i, d)
95     Escribir Sin Saltar tit[n], " ", nom[n], " ", ape[n], " | "
96   FinSi
97 FinSubProceso
```

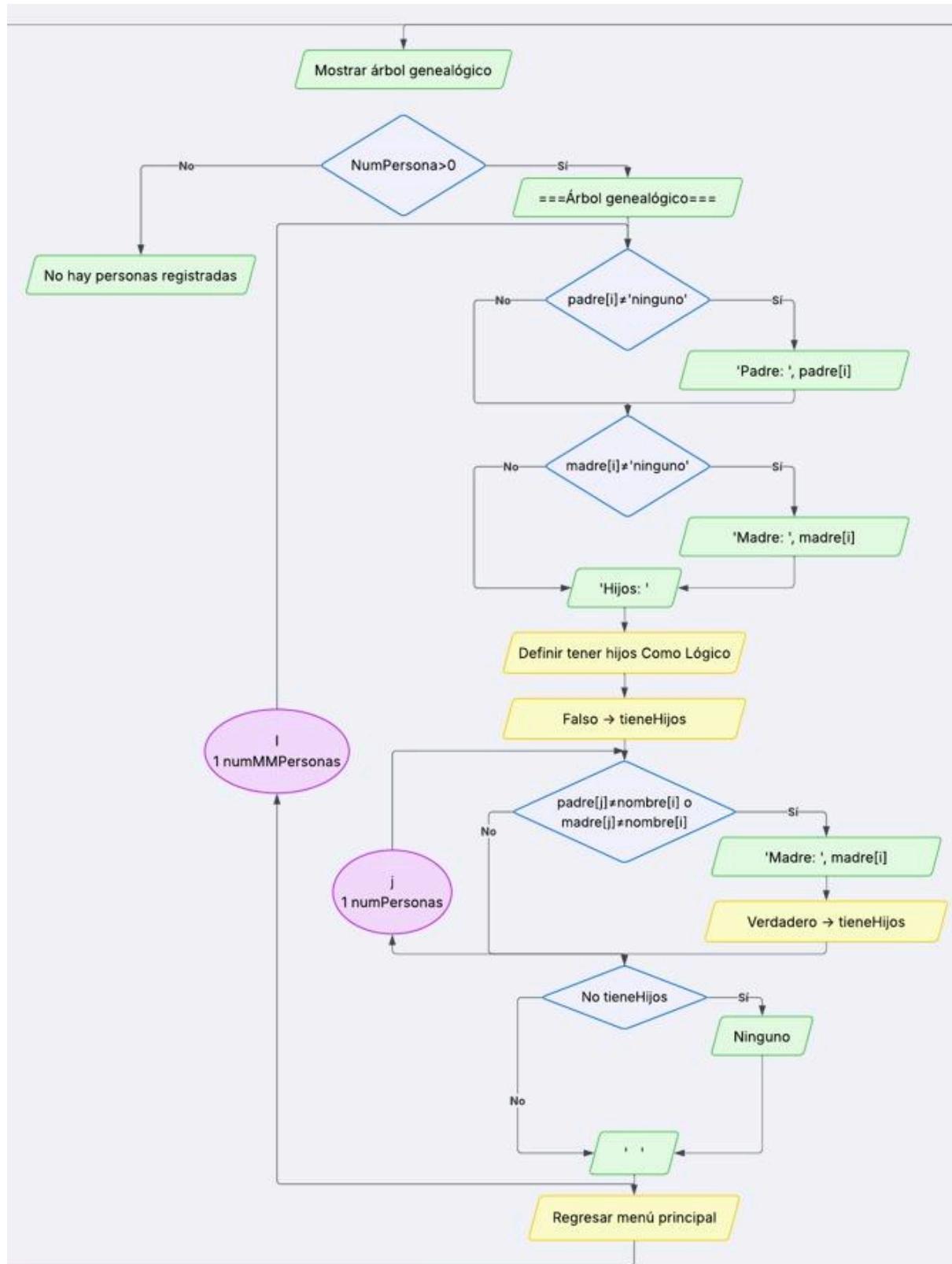
3. Diagramas de flujo del pseudocódigo

Pseudocódigo para crear un árbol binario.

Agregar Persona



Mostrar Árbol Genealógico



Buscar familiares



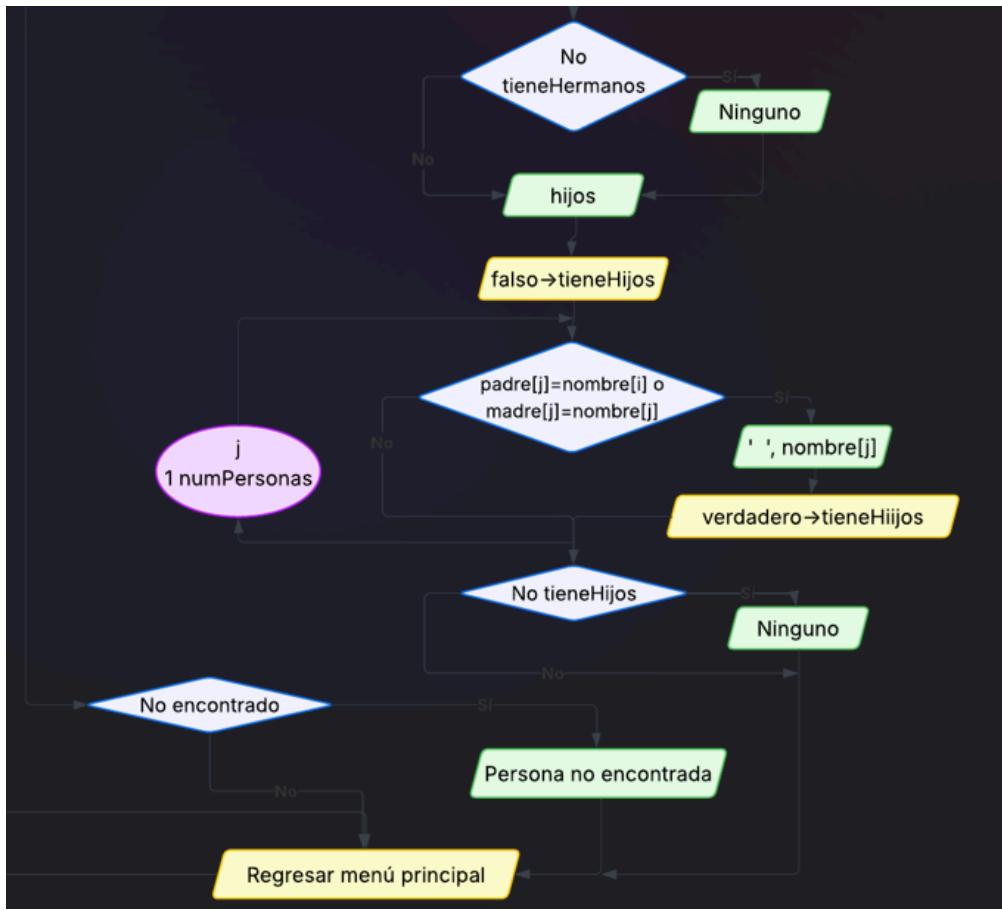
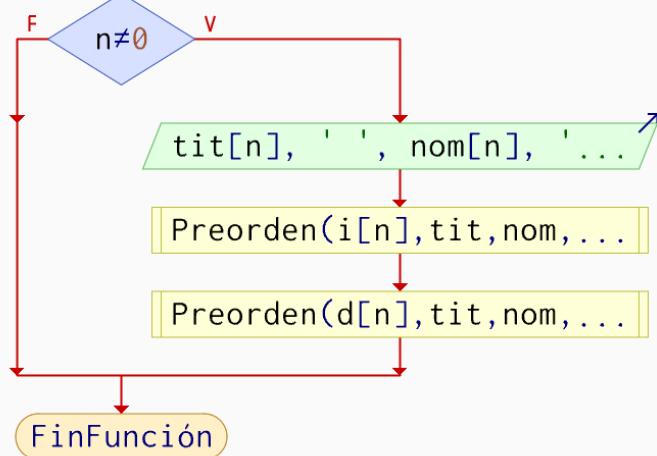


Diagrama de ordenamiento

Preorden

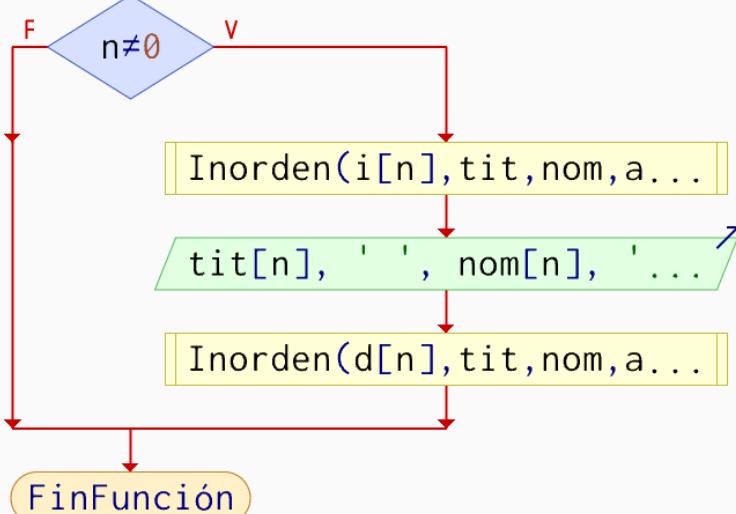
DEFINICIÓN DE SUBPROCESOS

Función Preorden(n , tit, nom, ape...)

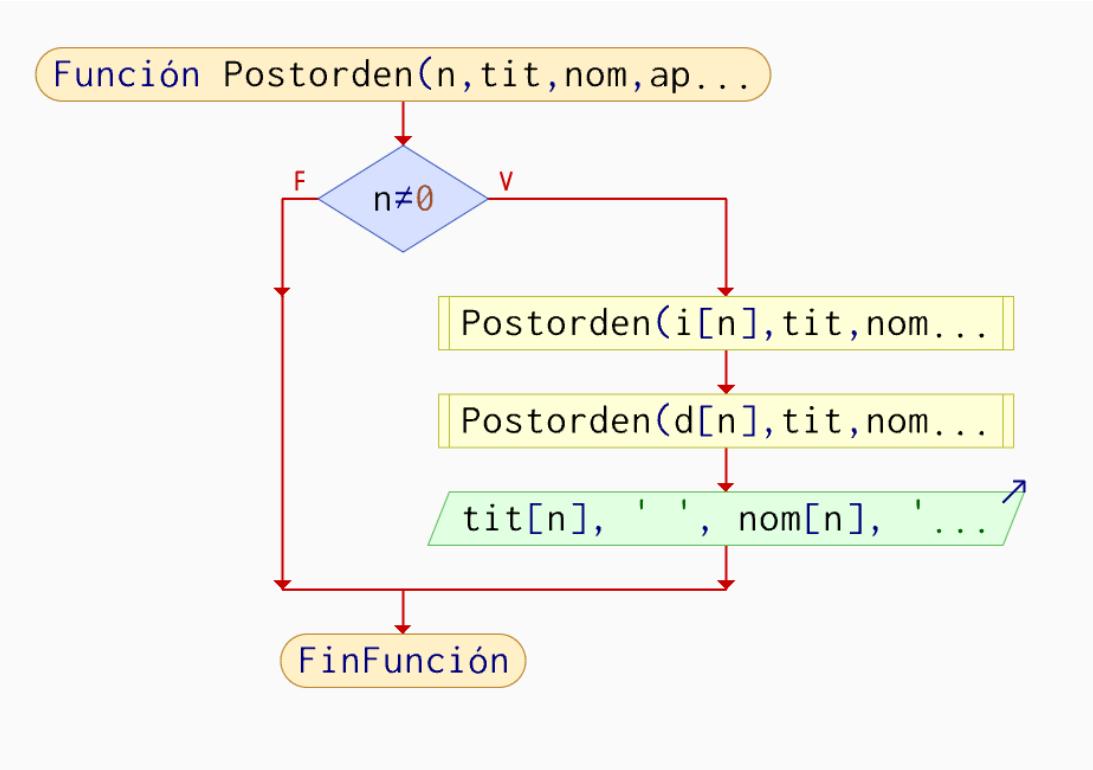


In-order

Función Inorden(n , tit, nom, ape, ...)



PostOrden



4. Avance del código fuente

```
Avance del codigo fuente.cpp

1 #include <iostream>
2 using namespace std;
3
4 // Definición de tamaño máximo para las cadenas
5 #define MAX_STR_LEN 30
6
7 // ESTRUCTURA DEL NODO
8 struct Nodo {
9     char id[MAX_STR_LEN];
10    char nombre[MAX_STR_LEN];
11    char fechaNacimiento[MAX_STR_LEN];
12    char titulo[MAX_STR_LEN];
13    char etnia[MAX_STR_LEN];
14    Nodo* padre;
15    Nodo* madre;
16 };
17
18 // FUNCIONES ESENCIALES Y AUXILIARES
19
20 void limpiarBuffer() {
21     // Limpieza simple del buffer de entrada después de cin
22     cin.ignore(MAX_STR_LEN, '\n');
23 }
24
25 void obtenerLinea(char buffer[]) {
26     // Simula getline: Lee una línea completa y limpia el buffer
27     cin.get(buffer, MAX_STR_LEN, '\n');
28     cin.ignore(MAX_STR_LEN, '\n');
29 }
30
31 // Función auxiliar para comparar cadenas (sustituye a <cstring>::strcmp)
32 bool compararCadenas(const char* a, const char* b) {
33     int i = 0;
34     while (a[i] != '\0' && b[i] != '\0') {
35         if (a[i] != b[i]) return false;
36         i++;
37     }
38     return a[i] == b[i];
39 }
40
41 // Función auxiliar para copiar cadenas (sustituye a <cstring>::strcpy)
42 void copiarCadena(char* destino, const char* origen) {
43     int i = 0;
```

```
43     int i = 0;
44     while ((destino[i] = origen[i]) != '\0') {
45         i++;
46     }
47 }
48
49 // Función para imprimir todos los detalles de un miembro
50 void imprimirDetalles(Nodo* nodo) {
51     cout << "-----\n";
52     cout << "ID: " << nodo->id << endl;
53     cout << "Nombre: " << nodo->titulo << " " << nodo->nombre << endl;
54     cout << "Nacimiento: " << nodo->fechaNacimiento << endl;
55     cout << "Etnia: " << nodo->etnia << endl;
56     cout << "-----\n";
57 }
58
59 // Función para inicializar un nuevo nodo
60 Nodo* crearNodo(const char* i, const char* n, const char* fn, const char* t, const char* e) {
61     Nodo* nodo = new Nodo();
62     copiarCadena(nodo->id, i);
63     copiarCadena(nodo->nombre, n);
64     copiarCadena(nodo->fechaNacimiento, fn);
65     copiarCadena(nodo->titulo, t);
66     copiarCadena(nodo->etnia, e);
67     nodo->padre = NULL;
68     nodo->madre = NULL;
69     return nodo;
70 }
71
72 // ÚNICA FUNCIÓN DE BÚSQUEDA
73 Nodo* buscarNodo(Nodo* nodo, const char* nombreBuscar) {
74     if (nodo == NULL) return NULL;
75
76     // Búsqueda simplificada: debe coincidir el nombre exacto
77     if (compararCadenas(nodo->nombre, nombreBuscar)) {
78         return nodo;
79     }
80
81     Nodo* encontrado = buscarNodo(nodo->padre, nombreBuscar);
82     if (encontrado != NULL) return encontrado;
83
84     return buscarNodo(nodo->madre, nombreBuscar);
85 }
86
87
88 // 1. FUNCIONALIDAD PRINCIPAL: AGREGAR MIEMBRO
89 void agregarMiembro(Nodo* RAIZ) {
90     char nId[MAX_STR_LEN], nNombre[MAX_STR_LEN], nFechaNac[MAX_STR_LEN];
91     char nTitulo[MAX_STR_LEN], nEtnia[MAX_STR_LEN], nombreAncestro[MAX_STR_LEN];
92     char relacion[2];
```

```
88 // 1. FUNCIONALIDAD PRINCIPAL: AGREGAR MIEMBRO
89 void agregarMiembro(Nodo* RAIZ) {
90     char nId[MAX_STR_LEN], nNombre[MAX_STR_LEN], nFechaNac[MAX_STR_LEN];
91     char nTitulo[MAX_STR_LEN], nEtnia[MAX_STR_LEN], nombreAncestro[MAX_STR_LEN];
92     char relacion[2];
93
94     cout << "\n--- 1. AGREGAR NUEVO MIEMBRO ---\n";
95
96     cout << "ID: "; obtenerLinea(nId);
97     cout << "Nombre: "; obtenerLinea(nNombre);
98     cout << "Nacimiento: "; obtenerLinea(nFechaNac);
99     cout << "Titulo: "; obtenerLinea(nTitulo);
100    cout << "Etnia: "; obtenerLinea(nEtnia);
101
102    Nodo* nuevoNodo = crearNodo(nId, nNombre, nFechaNac, nTitulo, nEtnia);
103
104    cout << "Ancestro: "; obtenerLinea(nombreAncestro);
105    Nodo* ancestro = buscarNodo(RAIZ, nombreAncestro);
106
107    if (ancestro == NULL) {
108        cout << "Error: Ancestro no encontrado. Miembro no insertado.\n";
109        delete nuevoNodo; return;
110    }
111
112    cout << "Conectar a [P]adre/[M]adre: "; cin >> relacion; limpiarBuffer();
113
114    // Lógica de Inserción Simplificada (Usando puntero a puntero conceptual)
115    Nodo** punteroDestino = NULL;
116    char opcion = relacion[0];
117
118    if (opcion == 'P' || opcion == 'p') {
119        punteroDestino = &ancestro->padre;
120    } else if (opcion == 'M' || opcion == 'm') {
121        punteroDestino = &ancestro->madre;
122    }
123
124    if (punteroDestino == NULL) {
125        cout << "Error: Opcion no valida.\n";
126        delete nuevoNodo;
127        return;
128    }
129
130    if (*punteroDestino == NULL) {
131        *punteroDestino = nuevoNodo;
132        cout << nNombre << " agregado a " << ancestro->nombre << ".\n";
133    } else {
134        cout << "Error: Posicion ocupada por " << (*punteroDestino)->nombre << ".\n";
135        delete nuevoNodo;
136    }
137 }
```

```

130     if (*punteroDestino == NULL) {
131         *punteroDestino = nuevoNodo;
132         cout << nNombre << " agregado a " << ancestro->nombre << ".\n";
133     } else {
134         cout << "Error: Posicion ocupada por " << (*punteroDestino)->nombre << ".\n";
135         delete nuevoNodo;
136     }
137 }
138
139 // 2. FUNCIONALIDAD RESTAURADA: MOSTRAR MIEMBROS
140 void listarDetalles(Nodo* nodo) {
141     if (nodo != NULL) {
142         imprimirDetalles(nodo);
143         listarDetalles(nodo->padre);
144         listarDetalles(nodo->madre);
145     }
146 }
147
148 void mostrarMiembros(Nodo* RAIZ) {
149     cout << "\n--- 2. LISTA COMPLETA DE MIEMBROS ---\n";
150     listarDetalles(RAIZ);
151 }
152
153 // 3. FUNCIONALIDAD RESTAURADA: BUSCAR MIEMBRO
154 void buscarMiembro(Nodo* RAIZ) {
155     char nombreBuscar[MAX_STR_LEN];
156     cout << "\n--- 3. BUSCAR MIEMBRO ---\n";
157     cout << "Nombre a buscar: ";
158     obtenerLinea(nombreBuscar);
159
160     Nodo* resultado = buscarNodo(RAIZ, nombreBuscar);
161
162     if (resultado != NULL) {
163         cout << "\n¡Miembro Encontrado!\n";
164         imprimirDetalles(resultado);
165     } else {
166         cout << "\nMiembro no encontrado.\n";
167     }
168 }
169
170 // CREACIÓN DEL ÁRBOL INICIAL Y MAIN
171 Nodo* crearArbolBorjaLoyolaInca() {
172     Nodo* FranciscoBorja = crearNodo("B1", "Francisco de Borja", "1510", "Don", "Virreinal");
173     Nodo* Carlos = crearNodo("M2.1", "Carlos de Borja", "1540", "Don", "Mestizo");
174
175     FranciscoBorja->padre = Carlos;
176
177     return FranciscoBorja;
178 }
179

```

```
166     cout << "\nMiembro no encontrado.\n";
167 }
168
169
170 // CREACIÓN DEL ÁRBOL INICIAL Y MAIN
171 Nodo* crearArbolBorjaLoyolaInca() {
172     Nodo* FranciscoBorja = crearNodo("B1", "Francisco de Borja", "1510", "Don", "Virreinal");
173     Nodo* Carlos = crearNodo("M2.1", "Carlos de Borja", "1540", "Don", "Nestizo");
174
175     FranciscoBorja->padre = Carlos;
176
177     return FranciscoBorja;
178 }
179
180 int main() {
181     Nodo* RAIZ = crearArbolBorjaLoyolaInca();
182     int opcion;
183
184     cout << "ADVERTENCIA: La busqueda y la insercion requieren el nombre exacto." << endl;
185
186     do {
187         cout << "\n===== ARBOL GENEALOGICO COMPLETO =====\n";
188         cout << "===== =====\n";
189         cout << "1. Agregar Miembro\n";
190         cout << "2. Mostrar Miembros\n";
191         cout << "3. Buscar Miembro\n";
192         cout << "4. Salir\n";
193         cout << "Seleccione una opcion: ";
194
195
196         if (!(cin >> opcion)) {
197             cin.clear(); limpiarBuffer(); opcion = 0;
198         } else {
199             cout << "\n";
200         }
201
202         switch (opcion) {
203             case 1: agregarMiembro(RAIZ); break;
204             case 2: mostrarMiembros(RAIZ); break;
205             case 3: buscarMiembro(RAIZ); break;
206
207             case 4: cout << "Saliendo del programa.\n"; break;
208
209             default: if (opcion != 0) cout << "Opcion no valida.\n"; break;
210         }
211     } while (opcion != 4);
212
213     return 0;
214 }
```

CAPÍTULO 3 – SOLUCIÓN FINAL

1. Código limpio, bien comentado y estructurado.

```
//estructura del nodo
] struct Nodo {
    char id[MAX_STR_LEN];
    char nombre[MAX_STR_LEN];
    char fechaNacimiento[MAX_STR_LEN];
    char titulo[MAX_STR_LEN];
    char etnia[MAX_STR_LEN];
    Nodo* padre;
    Nodo* madre;
    Nodo* hijoPrimero;
    Nodo* siguienteHermano;
    bool visitado;
};

//limpiador del teclado
] void limpiarBuffer() {
    cin.ignore(10000, '\n');
}

// paa leer el texto
] void obtenerLinea(char buffer[]) {
    cin.getline(buffer, MAX_STR_LEN);
    if (cin.fail()) {
        cin.clear();
        limpiarBuffer();
        buffer[0] = '\0';
    }
}

//funciones
//comparar las cadasnas caracter por caracter
] bool compararCadenas(const char* a, const char* b) {
    int i = 0;
    while (a[i] != '\0' && b[i] != '\0') {
        if (a[i] != b[i]) return false;
        i++;
    }
    return a[i] == b[i];
}
```

```

// copia una cadena a otra
void copiarCadena(char* destino, const char* origen) {
    int i = 0;
    while ((destino[i] = origen[i]) != '\0') {
        i++;
    }
}

// convierte de entero a texto
void intToChar(int n, char buffer[]) {
    if (n == 0) {
        buffer[0] = '0';
        buffer[1] = '\0';
        return;
    }
    char temp[MAX_STR_LEN];
    int i = 0;
    while (n > 0) {
        temp[i++] = (n % 10) + '0';
        n /= 10;
    }
    int j = 0;
    while (i > 0) buffer[j++] = temp[--i];
    buffer[j] = '\0';
}

// crear e imprimir

// construye un nuevo miembro de la familia
Nodo* crearNodo(const char* n, const char* fn, const char* t, const char* e) {
    Nodo* nodo = new Nodo();
    intToChar(next_member_id++, nodo->id);
    copiarCadena(nodo->nombre, n);
    copiarCadena(nodo->fechaNacimiento, fn);
    copiarCadena(nodo->titulo, t);
    copiarCadena(nodo->etnia, e);
    nodo->padre = nullptr;
    nodo->madre = nullptr;
    nodo->hijoPrimero = nullptr;
    nodo->siguienteHermano = nullptr;
    nodo->visitado = false;
    return nodo;
}

```

2. Capturas de pantalla de las ventanas de ejecución con las diversas pruebas de validación de datos

```

MENU PRINCIPAL
1. Agregar Miembro
2. Mostrar Todos los Miembros
3. Buscar Miembro
4. Visualizar Arbol Genealogico
5. Salir
Seleccione una opcion: 1

--- AGREGAR NUEVO MIEMBRO ---
Nombre: juan
fecha de nacimiento: 1525
titulo (Don/Doña): don
etnia: inca

selecciona: [H] hijo@ | [P] padre o madre
Opcion (H/P): h
nombre del antecesor existente: JUANA
indica si es 'P' (padre) o 'M' (madre): M
exito agregado como hijo (madre: JUANA) [ID: 3]

```

la parte 1, agregar miembro se encarga de registrar a un nuevo miembro donde nos pedirá poner el nombre, fecha de nacimiento, título, etnia, después nos pedía seleccionar si es padre o hijo, el nombre de su antecesor, después indicar si el antecesor elegido es padre o madre, finalmente nos mandara el mensaje de que el registro fue completado con éxito

```

----- LISTA COMPLETA DE MIEMBROS -----

ID: 1
Nombre: Don FRANCISCO
Nacimiento: 1510
Etnia: Virreinal
Madre: JUANA

-----
ID: 2
Nombre: Doña JUANA
Nacimiento: 1520
Etnia: Inca

-----
ID: 3
Nombre: don juan
Nacimiento: 1525
Etnia: inca
Madre: JUANA

```

en la segunda opción nos devolverá todos los miembros registrados con éxito, mostrando sus id, nombre, nacimiento, etnia, madre o padre, o descendencia

```
----- BUSCAR MIEMBRO -----
nombre para buscar : JUANA
miembro encontrado:
```

```
-----
ID: 2
Nombre: Doña JUANA
Nacimiento: 1520
Etnia: Inca
```

en la tercera opción, tendremos que poner el numero del miembro que desamor buscar, después si lo encontramos nos mostrará los datos como el id, nombre, nacimiento, etnia, caso contrario nos mandará un mensaje que indica que no se encontró al miembro

```
--- ARBOL GENEALOGICO (VISUAL) ---
RAIZ: Don FRANCISCO (1510) [ID:1]
M: Doña JUANA (1520) [ID:2]
H: don juan (1525) [ID:3]
```

en la cuarta opción nos mostrará el árbol genealógico que se ha creado en base a los datos que se registraron

CAPÍTULO 4 – EVIDENCIAS DE TRABAJO COLABORATIVO

Repositorio con Control de Versiones (Capturas de Pantalla)

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

The screenshot shows a GitHub repository interface. At the top, there's a dropdown for the branch 'main'. Below it, there are two sections of commits:

- Commits on Nov 20, 2025:**
 - Add files via upload** by DiegoQuispeAlmanza authored 22 minutes ago. Status: Verified, SHA: 544d792.
 - Create README.md** by DiegoQuispeAlmanza authored 23 minutes ago. Status: Verified, SHA: 1211830.
- Commits on Nov 14, 2025:**
 - Add files via upload** by DiegoQuispeAlmanza authored last week. Status: Verified, SHA: d84fe28.
 - Create LICENSE** by DiegoQuispeAlmanza authored last week. Status: Verified, SHA: 9272628.

(Nota: Se presentaron problemas técnicos para la subida de los archivos, uno de los integrantes se encargó de subir el archivo)

Historial de ramas y fusiones si es aplicable

Evidencia por cada integrante del equipo

Responsable	Roles
Kelmer	Programador principal Arquitecto de software

Tarea realizada:
 -creación de los diagramas de flujo
 -elaboración del pseudo código de ordenamiento



BRAYDDY BRAYAN BELTRÁN
MAMANI

23:14

Avance



DIEGO ANDREE QUISPE ALMANZA 23:16

Tarea realizada:
 - Diseño de estructura de datos
 - Documentacion de evidencias

Brayan	Gerente de producto Analista de requerimientos
Diego	Documentación del proyecto QA Tester

Semana 14

Encargado	Responsabilidades
Kelmer	+ Creación de los diagramas de flujo + Elaboración del pseudo código de ordenamiento + Desarrollo del código fuente
Brayan	+ Asistente de programación del código fuente + Análisis funcional de los requerimientos
Diego	+ Diseño de estructura de datos + Documentación de evidencias

Semana 15

Encargado	Responsabilidades
Kelmer	+ Rediseño de los diagramas de flujo + Desarrollo del código fuente
Brayan	+ Asistente de programación del código fuente + Creación de validaciones

Diego	<ul style="list-style-type: none"> + Actualización manual de usuario + Documentación de evidencias
-------	--

Enlace a la herramienta colaborativa

<https://github.com/DiegoQuispeAlmanza/ARBOLES-GENEALOGICOS-ABR->

Semana 14

The screenshot shows the GitHub repository page for 'ARBOLES-GENEALOGICOS-ABR-' (Public). The repository has 1 branch and 0 tags. The commit history shows the following activity:

- DiegoQuispeAlmanza Add files via upload (544d792 · now)
- Avance del codigo fuente.cpp (Add files via upload · now)
- Copia de S13_Trabajo_Grupal.pdf (Add files via upload · last week)
- LICENSE (Create LICENSE · last week)
- README.md (Create README.md · 1 minute ago)

The README file contains the following content:

```
Creación de arbol genealogico por metodología ABR (Aprendizaje Basado en Retos)

Programas utilizados:
• Documentación: Google Docs
• Reuniones virtuales: Google Meet y Google Calendar
• Programación: Dev C++ y PSeint
```

The repository details section includes:

- About: Desarrollo de un sistema dinámico por medio de estructura de datos en C++.
- Readme
- MIT license
- Activity
- 0 stars
- 0 watching
- 0 forks
- Releases: No releases published. Create a new release
- Packages: No packages published. Publish your first package

Semana 15

The screenshot shows the GitHub repository page for 'ARBOLES-GENEALOGICOS-ABR-' (Public). The repository has 1 branch and 0 tags. The commit history shows the following activity:

- brayanuc Update Avance del codigo fuente.cpp (91c6850 · 42 minutes ago)
- Avance codigo v3.cpp (código V3 · 5 days ago)
- Avance del codigo fuente.cpp (Update Avance del codigo fuente.cpp · 45 minutes ago)
- Copia de S13_Trabajo_Grupal.pdf (Add files via upload · 2 weeks ago)
- LICENSE (Create LICENSE · 2 weeks ago)
- README.md (Update README.md · last week)
- S13_Trabajo_Grupal F-2.pdf (Add files via upload · last week)

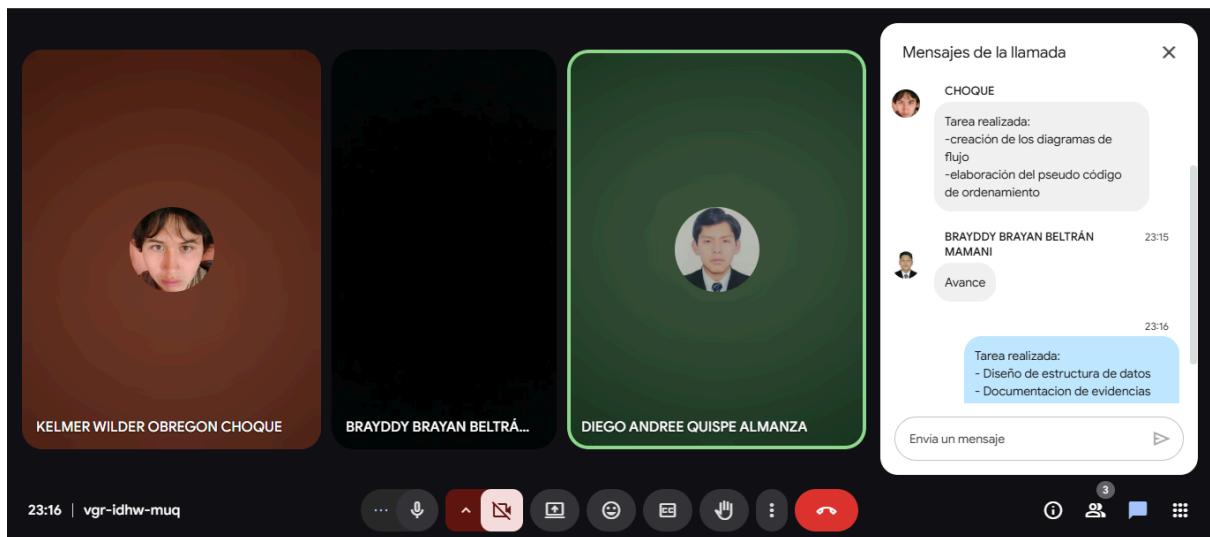
The repository details section includes:

- About: Desarrollo de un sistema dinámico por medio de estructura de datos en C++.
- Readme
- MIT license
- Activity
- 0 stars
- 0 watching
- 0 forks
- Releases: No releases published

Evidencias trabajo colaborativo

Reuniones Meet

Semana 14



X Desarrollo de Estructura de Datos - ABR

16 nov de 2025 6:00pm a 7:00pm 16 de nov de 2025 (GMT-05:00) Hora estándar de Perú Zona horaria

Todo el día Cada semana el domingo, jueves, hasta: 29 de nov de 2026

Confirmar asistencia: si Agregar nota o invitados

Detalles del evento Encuentra un hueco

Únete con Google Meet meet.google.com/urim-cwqz-dcbh-baq

Añade una ubicación

Notificación: 10 minutos

Añadir una notificación

DIEGO ANDREE QUISPE ALMANZA

No disponible Visibilidad predeterminada

La disponibilidad podría mostrarse en otras aplicaciones de Google

Añadir invitados

3 invitados

1 ya 2 en espera

DIEGO ANDREE QUISPE ALMANZA Organizador

KELMER WILDER OBREGON CHOQUE

BRAYDDY BRAYAN BELTRÁN MAMANI

Permisos de invitados

Editar el evento Invitar a otros Ver la lista de invitados

Crear notas de la reunión

Reunión de Estructura de Datos

≡ Calendar Hoy < > Noviembre de 2025

+ Crear

DOM	LUN	MAR	MIE	JUE
16	17	18	19	20

Noviembre de 2025

0 L M X J V S

26 27 28 29 30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30 1 2 3 4 5 6

Reunirse con... [Buscar a gente](#)

Páginas de reserva +

30 min con DIEGO

Estadísticas de tiempo 16 - 22 de NOV DE 2025 Tiempo en reuniones: 3 h (2.3 h de media)

Más información

Mis calendarios

DIEGO ANDREE QUISPE A., Cumpleaños, Tareas

Desarrollo de Estructura de Datos 6 - Típico

Desarrollo de Estructura de Datos 6 - Típico

Desarrollo de Estructura de Datos 6 - Típico

Manual para usuario

≡ Manual de Usuario - ABR

https://docs.google.com/document/d/1hS3Sr2z76dp2ssFXPdA_8Jt5tX3YsKKT8fkRAawl4II/edit?tab=t.0

Acta de compromiso

w Acta de compromiso (Grupo F) - Actualizado.docx

https://docs.google.com/document/d/1gdBmcfmS_7SzXApqJG9UyMNf45dcMgHx/edit

CONCLUSIONES

Reflexión sobre los aprendizajes alcanzados, dificultades enfrentadas durante la implementación, y posibles mejoras al sistema desarrollado.

REFERENCIAS

CEDEÑO ZAMBRANO, K. C.; LUCAS ALAY, D. D.; VERÁ POLANCO, L. A.; VERA ZAMBRANO, J. M.; ROMERO CASTRO, V. F. *Importancia de árboles binarios en la programación*. Journal TechInnovation, 2022, vol. 1, n.º 2, p. 33-42.

DOI: <https://doi.org/10.47230/Journal.TechInnovation.v1.n2.2022.33-42>.

Disponible en: <https://revistas.unesum.edu.ec/JTI/index.php/JTI/article/view/17>