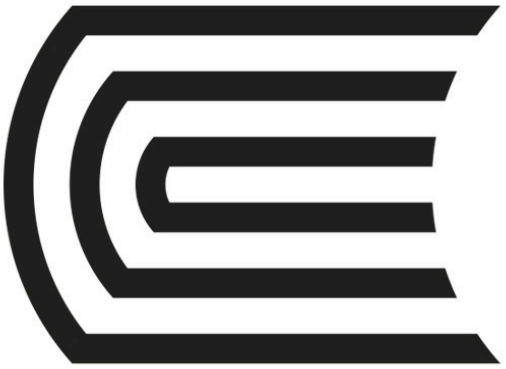


Universidad Continental

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

Resumen pendiente

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.

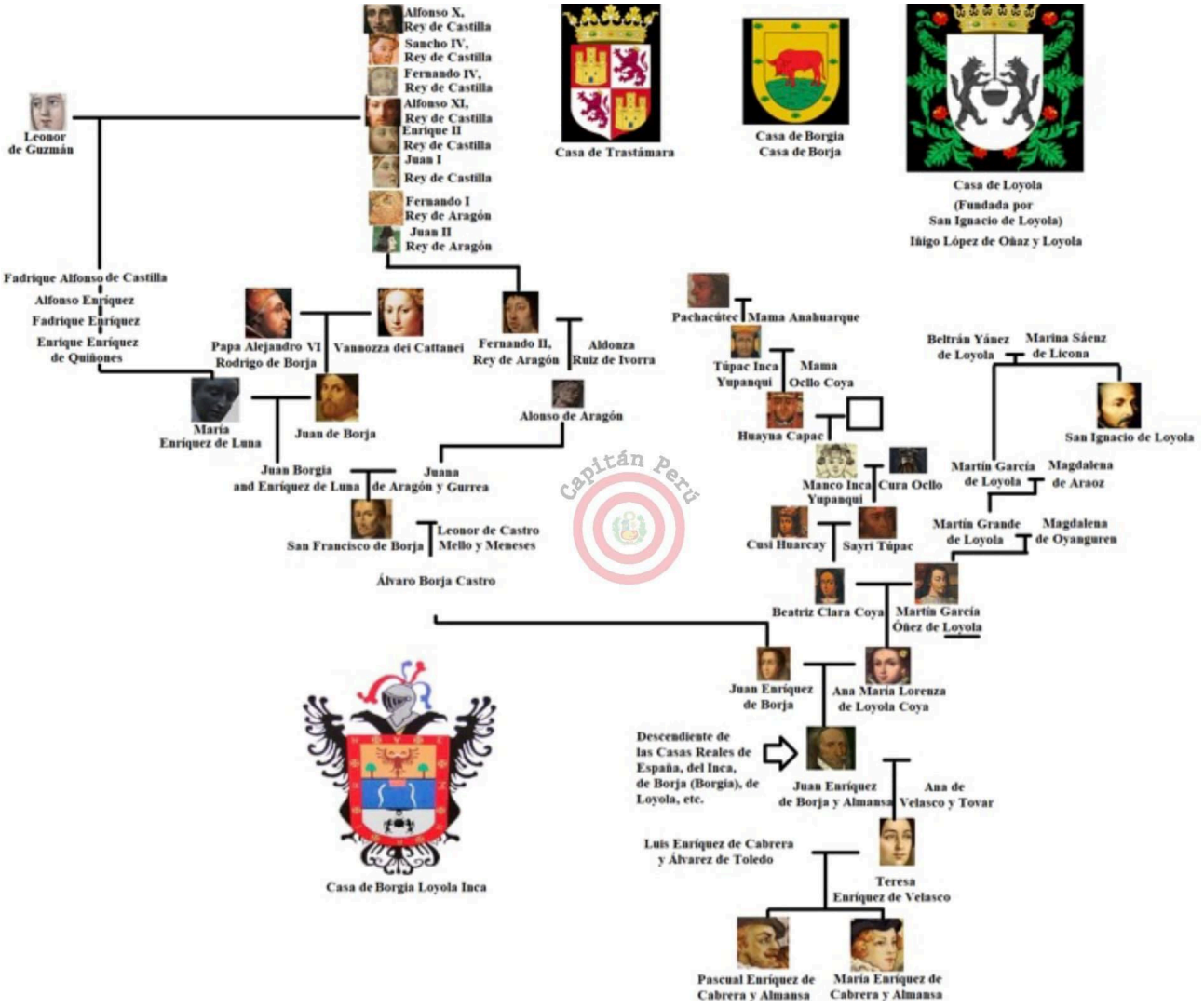
Aumentar información en esta sección

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?

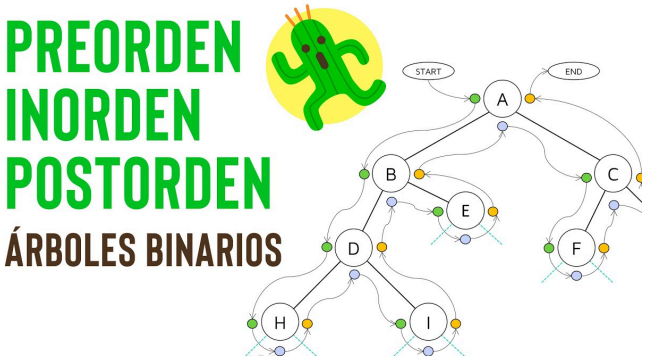
Inserción

- 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?

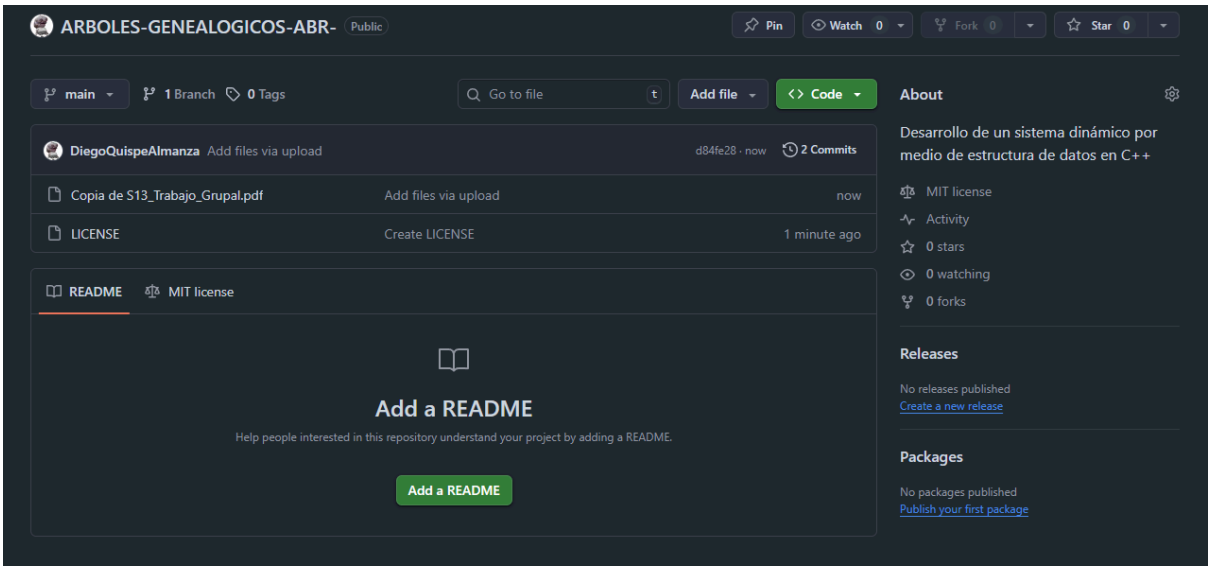
- Reorganización visual sin alterar relaciones
- Agrupar por generaciones

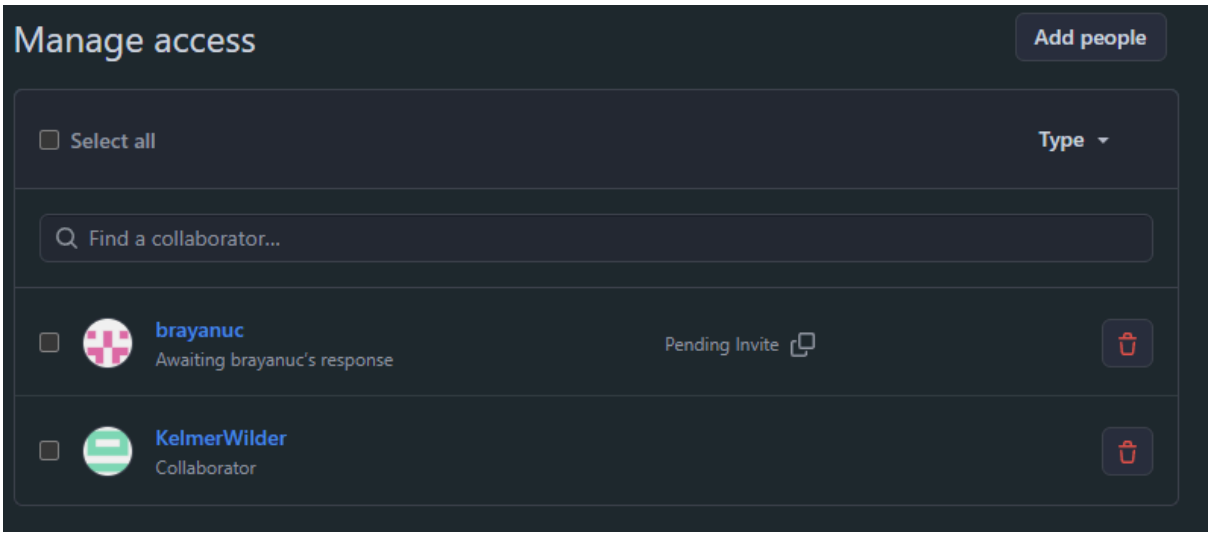
Se solicita responder a las preguntas guía con capturas de pantalla de la herramienta colaborativa en línea utilizada durante el desarrollo

4. Herramienta colaborativa

Enlace de la herramienta colaborativa utilizada y capturas de pantalla

GITHUB





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
        Según opcion Hacer
```

```
        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 '===================================='
    Escribir '      ?RBOL GENEAL?GICO COMPLETO'
    Escribir '===================================='
    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 '===================================='
            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 'Opci?n inv?lida. Intente nuevamente.'
FinSegún
FinMientras
FinAlgoritmo
```

Pseudocódigo para realizar el recorrido de un árbol

```
Algoritmo CasaBorjaLoyolaInca

    // DECLARACIÓN DE VARIABLES Y ARREGLOS
    Definir nombre, titulo, apellido Como Caracter
    Definir izq, der Como Entero
    // Dimensionamos los 5 arreglos
    Dimension titulo[10]
    Dimension nombre[10]
    Dimension apellido[10]
    Dimension izq[10]
    Dimension der[10]

    // DEFINICIÓN DEL ÁRBOL - Linaje Borja-Loyola Inca
    // [1] Don Fernando Borja Inca (Abuelo/Raíz) - Representa la unión de linajes
    titulo[1] = "Don"
    nombre[1] = "Fernando"
    apellido[1] = "Borja-Amaru" // Borja + Amaru (Serpiente, símbolo inca)
    izq[1] = 2
    der[1] = 3

    // [2] Doña Isabel (Hija) - Linaje fuerte Borja
    titulo[2] = "Doña"
    nombre[2] = "Isabel"
    apellido[2] = "Loyola"
    izq[2] = 4
    der[2] = 5

    // [3] Don Cristóbal (Hijo) - Linaje fuerte Inca
    titulo[3] = "Curaca" // Título tradicional andino
    nombre[3] = "Cristóbal"
    apellido[3] = "Tito Cusi"
    izq[3] = 6
    der[3] = 0

    // [4] Fray Francisco (Nieto) - Orientación religiosa
    titulo[4] = "Fray"
    nombre[4] = "Francisco"
    apellido[4] = "Borja"
    izq[4] = 0
```

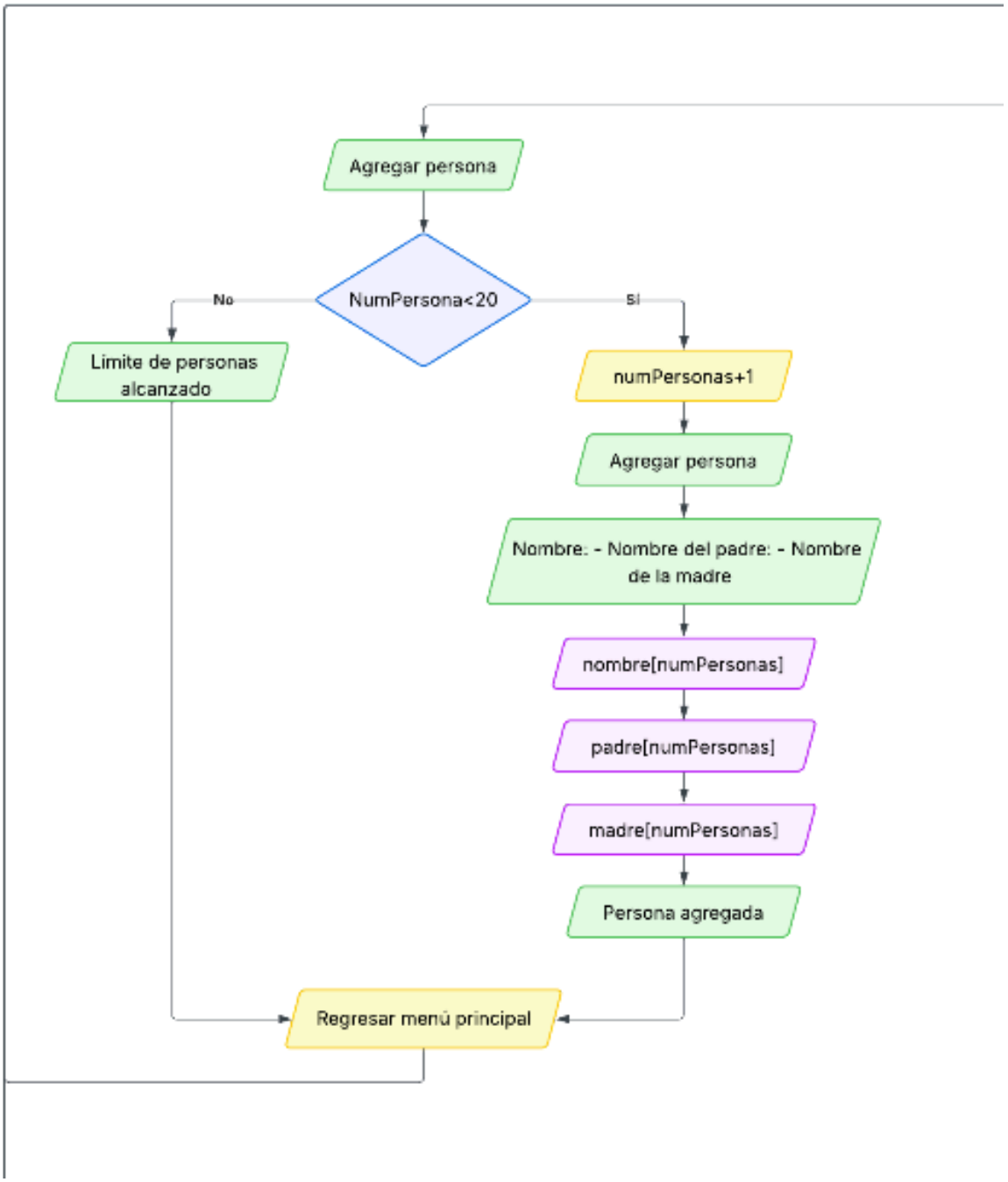
```
41
42      // [5] Doña Mencía (Nieta) - Linaje Loyola
43      titulo[5] = "Doña"
44      nombre[5] = "Mencía"
45      apellido[5] = "Loyola"
46      izq[5] = 0
47      der[5] = 0
48
49      // [6] Don Túpac (Nieta) - Linaje Túpac
50      titulo[6] = "Don"
51      nombre[6] = "Túpac"
52      apellido[6] = "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
```

```
FinAlgoritmo

// DEFINICIÓN DE SUBPROCESOS

SubProceso Preorden(n, tit, nom, ape, i, d)
    Si n ≠ 0 Entonces
        Escribir Sin Saltar tit[n], " ", nom[n], " ", ape[n], " | "
        Preorden(i[n], tit, nom, ape, i, d)
        Preorden(d[n], tit, nom, ape, i, d)
    FinSi
FinSubProceso


SubProceso Inorden(n, tit, nom, ape, i, d)
    Si n ≠ 0 Entonces
        Inorden(i[n], tit, nom, ape, i, d)
        Escribir Sin Saltar tit[n], " ", nom[n], " ", ape[n], " | "
        Inorden(d[n], tit, nom, ape, i, d)
    FinSi
FinSubProceso


SubProceso Postorden(n, tit, nom, ape, i, d)
    Si n ≠ 0 Entonces
        Postorden(i[n], tit, nom, ape, i, d)
        Postorden(d[n], tit, nom, ape, i, d)
        Escribir Sin Saltar tit[n], " ", nom[n], " ", ape[n], " | "
    FinSi
FinSubProceso
```

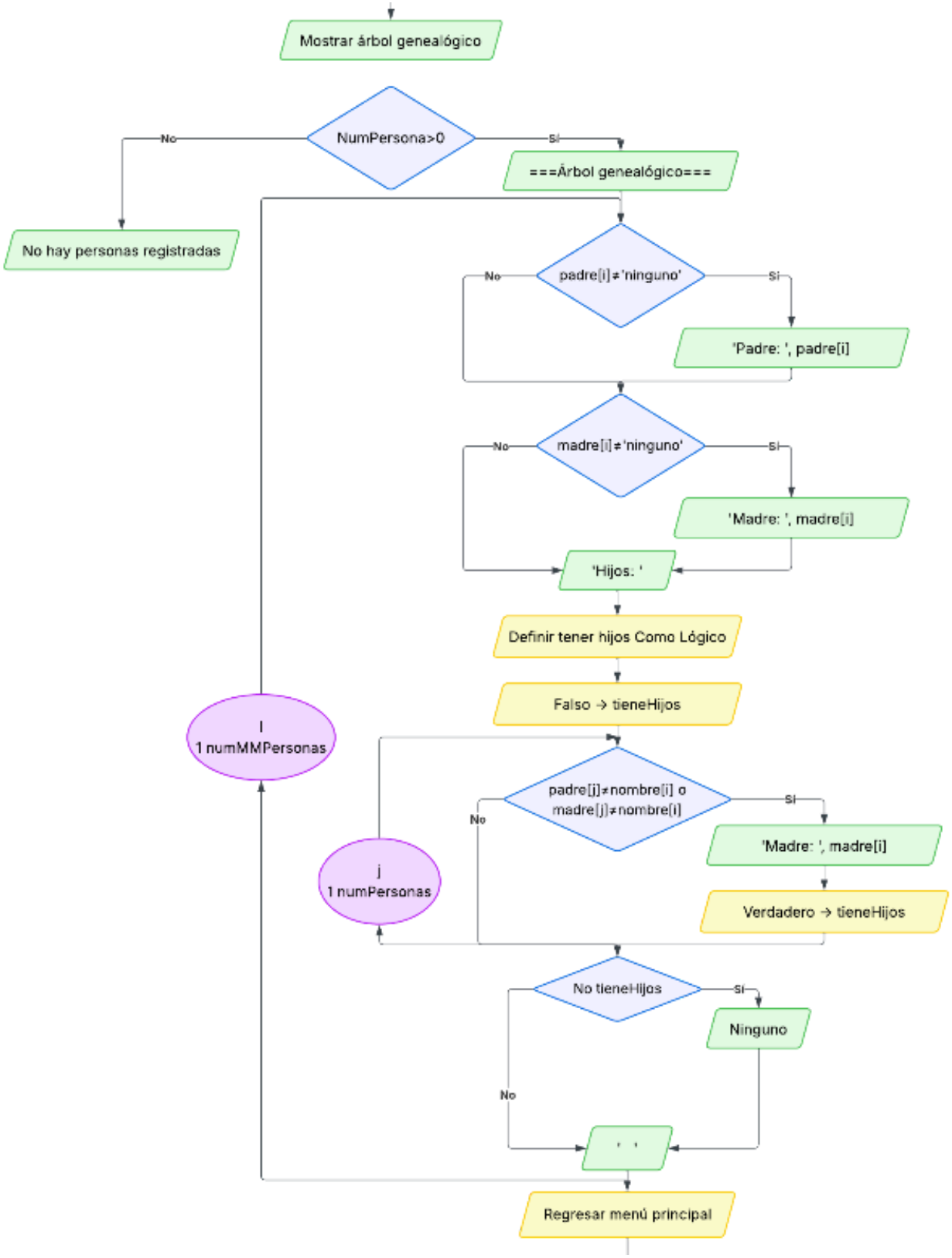
3. Diagramas de flujo del pseudocódigo

Pseudocódigo para crear un árbol binario.

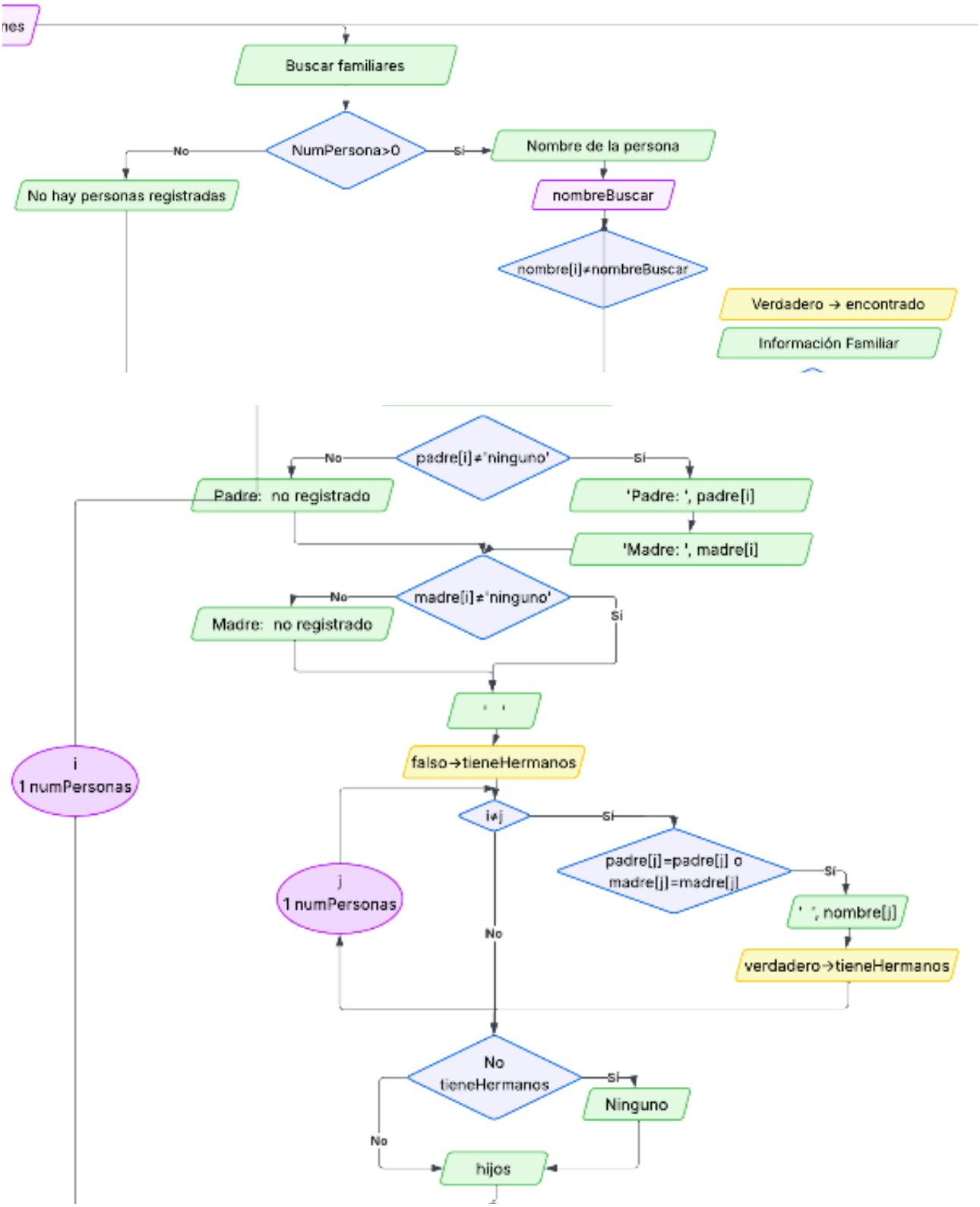
Agregar Persona



Mostrar Árbol Genealógico



Buscar familiares



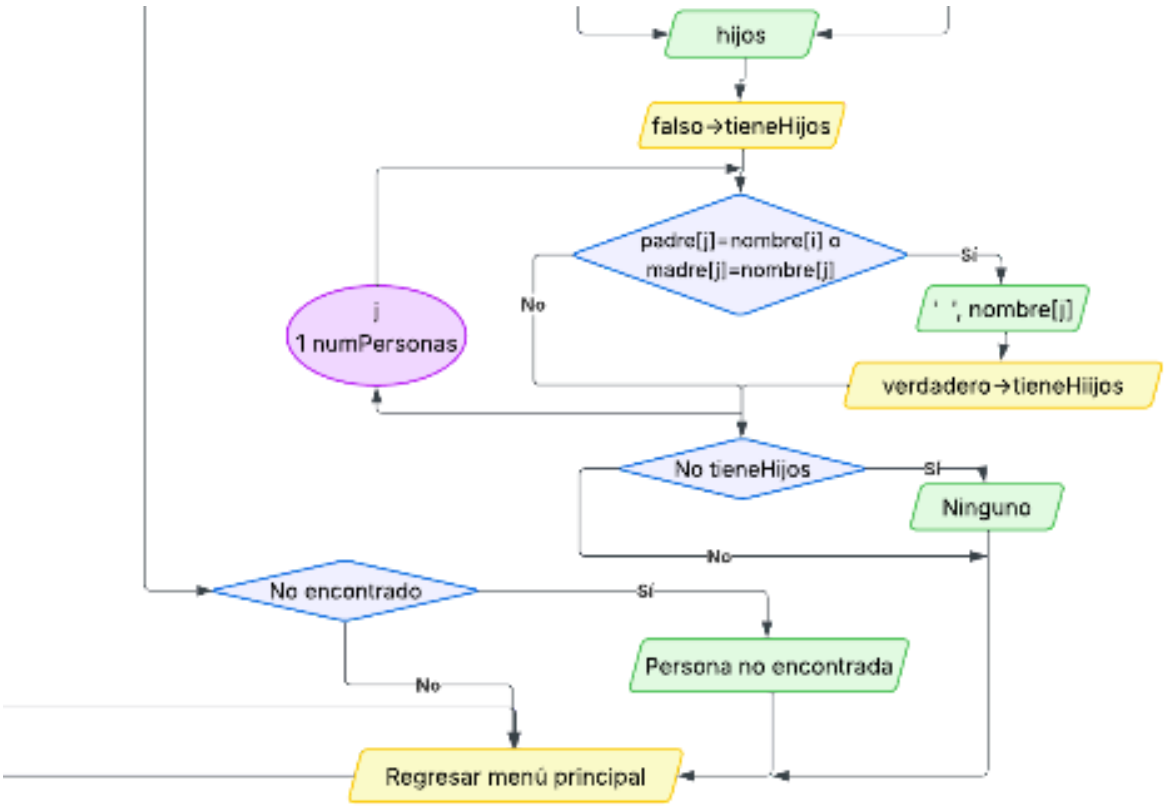
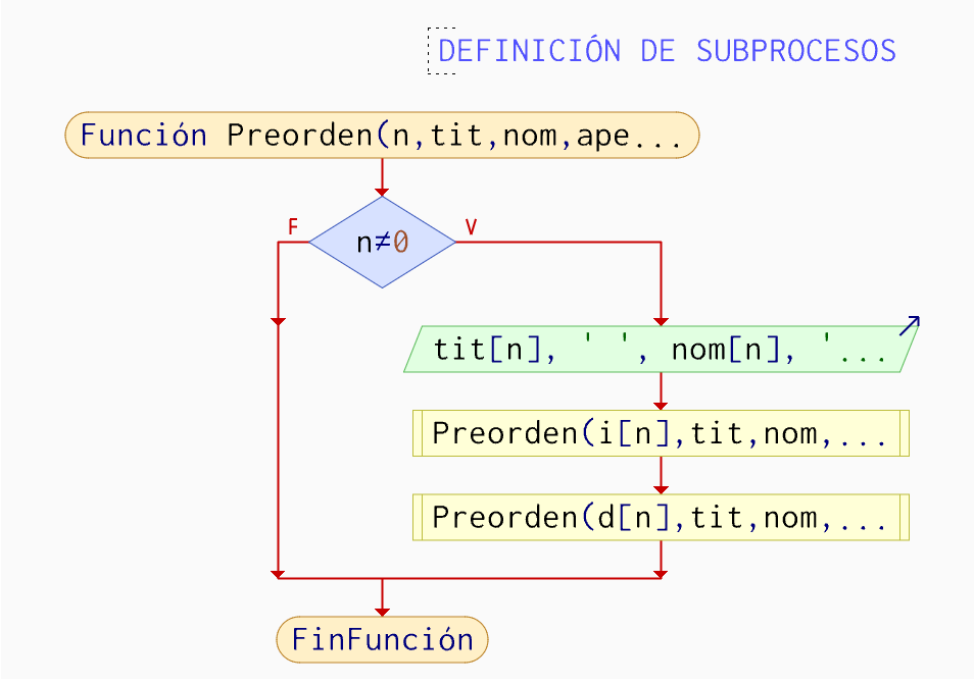
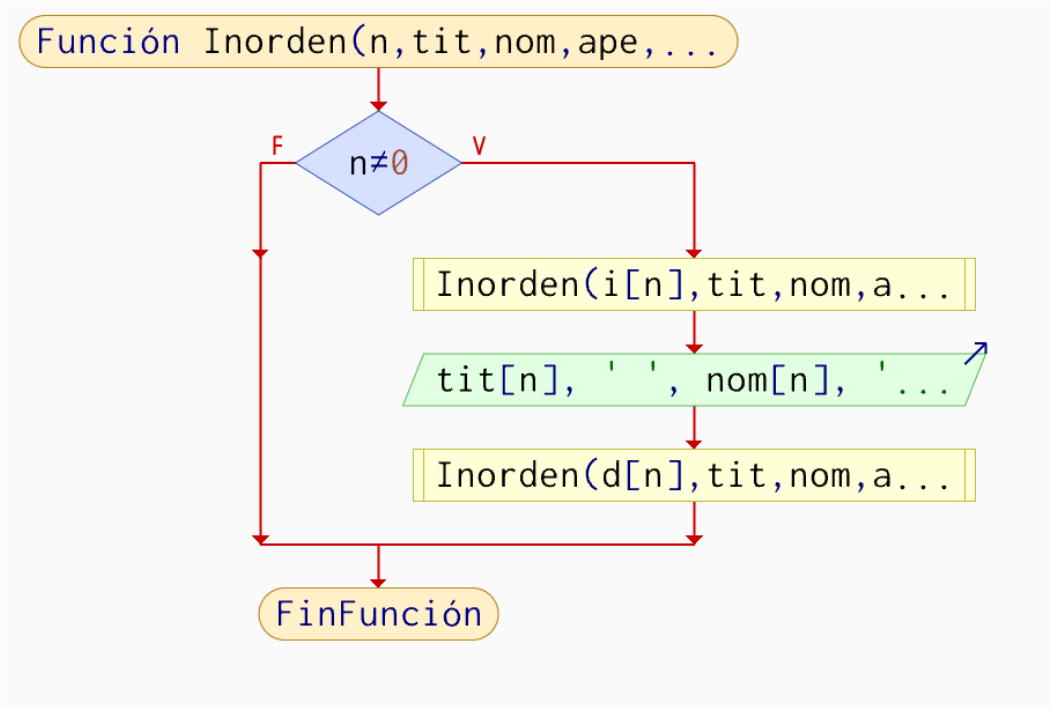


Diagrama de ordenamiento

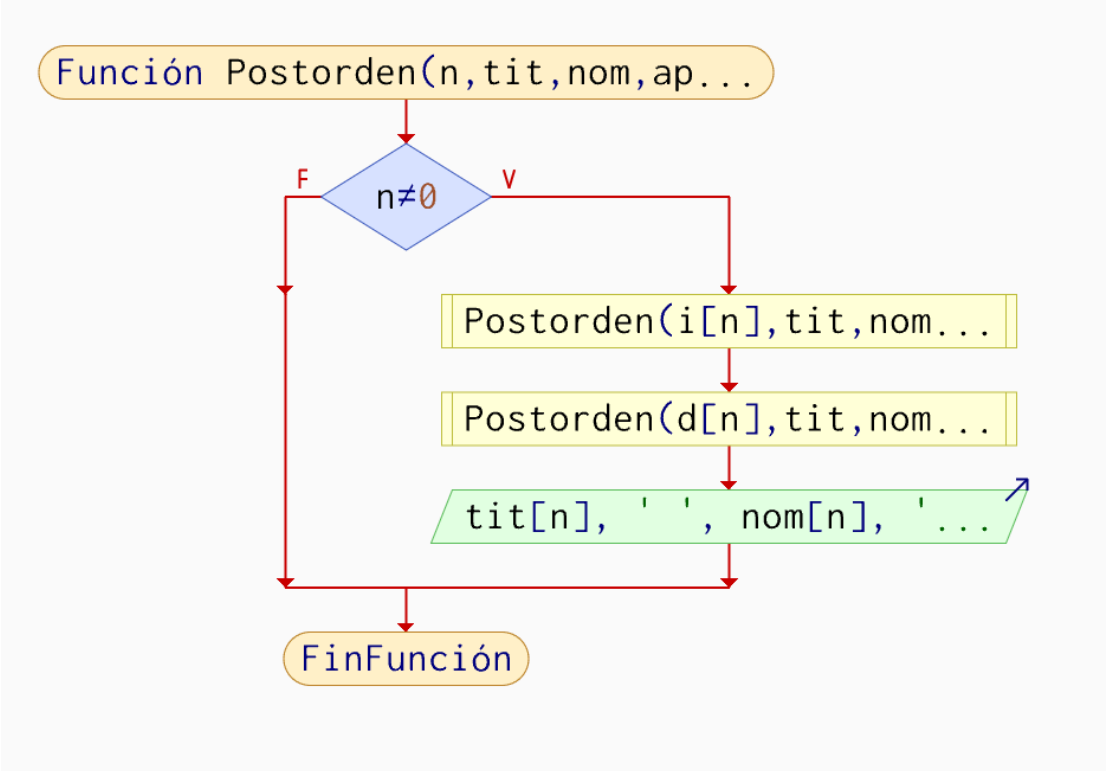
Preorden



In-order



PostOrden



4. Avance del código fuente

Avance del codigo fuente.cpp

```cpp
#include <iostream>
using namespace std;

// Definición de tamaño máximo para las cadenas
#define MAX_STR_LEN 30

// 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;
};

// FUNCIONES ESENCIALES Y AUXILIARES

void limpiarBuffer() {
    // Limpieza simple del buffer de entrada después de cin
    cin.ignore(MAX_STR_LEN, '\n');
}

void obtenerLinea(char buffer[]) {
    // Simula getline: Lee una línea completa y limpia el buffer
    cin.get(buffer, MAX_STR_LEN, '\n');
    cin.ignore(MAX_STR_LEN, '\n');
}

// Función auxiliar para comparar cadenas (sustituye a <cstring>::strcmp)
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];
}

// Función auxiliar para copiar cadenas (sustituye a <cstring>::strcpy)
void copiarCadena(char* destino, const char* origen) {
    int i = 0;
```

```
    int i = 0;
    while ((destino[i] = origen[i]) != '\0') {
        i++;
    }
}

// Función para imprimir todos los detalles de un miembro
void imprimirDetalles(Nodo* nodo) {
    cout << "--------------------------------\n";
    cout << "ID: " << nodo->id << endl;
    cout << "Nombre: " << nodo->titulo << " " << nodo->nombre << endl;
    cout << "Nacimiento: " << nodo->fechaNacimiento << endl;
    cout << "Etnia: " << nodo->etnia << endl;
    cout << "--------------------------------\n";
}

// Función para inicializar un nuevo nodo
Nodo* crearNodo(const char* i, const char* n, const char* fn, const char* t, const char* e) {
    Nodo* nodo = new Nodo();
    copiarCadena(nodo->id, i);
    copiarCadena(nodo->nombre, n);
    copiarCadena(nodo->fechaNacimiento, fn);
    copiarCadena(nodo->titulo, t);
    copiarCadena(nodo->etnia, e);
    nodo->padre = NULL;
    nodo->madre = NULL;
    return nodo;
}

// ÚNICA FUNCIÓN DE BÚSQUEDA
Nodo* buscarNodo(Nodo* nodo, const char* nombreBuscar) {
    if (nodo == NULL) return NULL;

    // Búsqueda simplificada: debe coincidir el nombre exacto
    if (compararCadenas(nodo->nombre, nombreBuscar)) {
        return nodo;
    }

    Nodo* encontrado = buscarNodo(nodo->padre, nombreBuscar);
    if (encontrado != NULL) return encontrado;

    return buscarNodo(nodo->madre, nombreBuscar);
}


// 1. FUNCIONALIDAD PRINCIPAL: AGREGAR MIEMBRO
void agregarMiembro(Nodo* RAIZ) {
    char nId[MAX_STR_LEN], nNombre[MAX_STR_LEN], nFechaNac[MAX_STR_LEN];
    char nTitulo[MAX_STR_LEN], nEtnia[MAX_STR_LEN], nombreAncestro[MAX_STR_LEN];
    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     }
```

```
    if (*punteroDestino == NULL) {
        *punteroDestino = nuevoNodo;
        cout << nNombre << " agregado a " << ancestro->nombre << ".\n";
    } else {
        cout << "Error: Posicion ocupada por " << (*punteroDestino)->nombre << ".\n";
        delete nuevoNodo;
    }
}

// 2. FUNCIONALIDAD RESTAURADA: MOSTRAR MIEMBROS
void listarDetalles(Nodo* nodo) {
    if (nodo != NULL) {
        imprimirDetalles(nodo);
        listarDetalles(nodo->padre);
        listarDetalles(nodo->madre);
    }
}

void mostrarMiembros(Nodo* RAIZ) {
    cout << "\n--- 2. LISTA COMPLETA DE MIEMBROS ---\n";
    listarDetalles(RAIZ);
}

// 3. FUNCIONALIDAD RESTAURADA: BUSCAR MIEMBRO
void buscarMiembro(Nodo* RAIZ) {
    char nombreBuscar[MAX_STR_LEN];
    cout << "\n--- 3. BUSCAR MIEMBRO ---\n";
    cout << "Nombre a buscar: ";
    obtenerLinea(nombreBuscar);

    Nodo* resultado = buscarNodo(RAIZ, nombreBuscar);

    if (resultado != NULL) {
        cout << "\n¡Miembro Encontrado!\n";
        imprimirDetalles(resultado);
    } else {
        cout << "\nMiembro no encontrado.\n";
    }
}

// CREACIÓN DEL ÁRBOL INICIAL Y MAIN
Nodo* crearArbolBorjaLoyolaInca() {
    Nodo* FranciscoBorja = crearNodo("B1", "Francisco de Borja", "1510", "Don", "Virreinal");
    Nodo* Carlos = crearNodo("M2.1", "Carlos de Borja", "1540", "Don", "Mestizo");

    FranciscoBorja->padre = Carlos;

    return FranciscoBorja;
}
```

```
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
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============================\n";
188         cout << " ARBOL GENEALOGICO COMPLETO\n";
189         cout << "============================\n";
190         cout << "1. Agregar Miembro\n";
191         cout << "2. Mostrar Miembros\n";
192         cout << "3. Buscar Miembro\n";
193         cout << "4. Salir\n";
194         cout << "Seleccione una opcion: ";
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

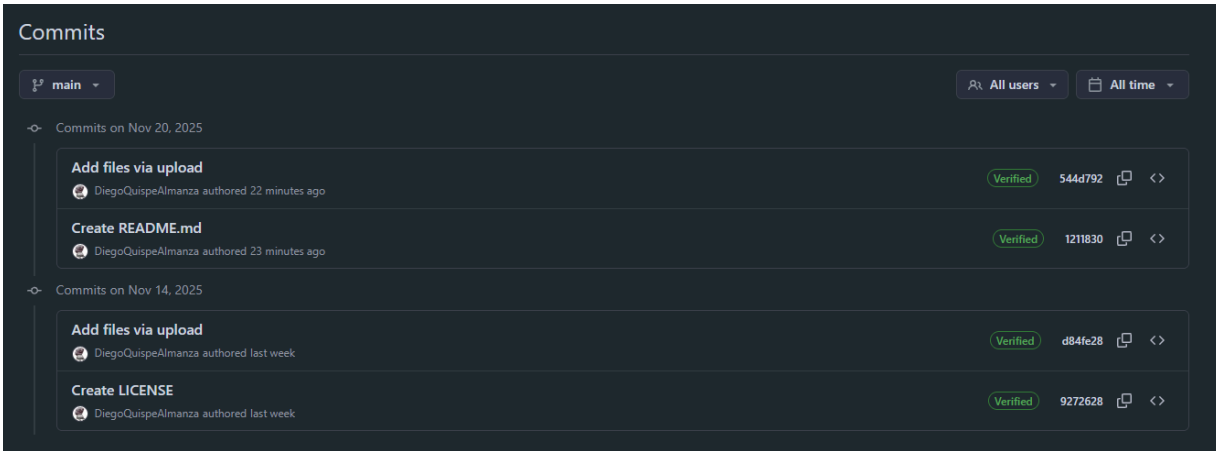
1. Código limpio, bien comentado y estructurado.

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

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).

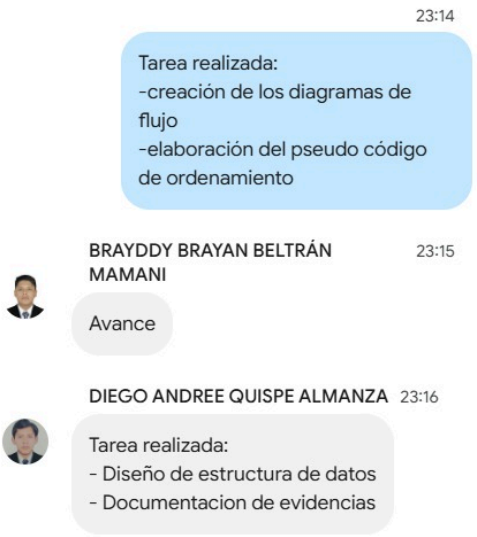


(Nota: Se presentaron problemas técnicos para la subida de los archivos, uno de los integrantes se encargó de subir el archivo)

Historial de ramas y fusiones si es aplicable

Evidencia por cada integrante del equipo

Responsable	Roles
Kelmer	Programador principal Arquitecto de software
Brayan	Gerente de producto Analista de requerimientos
Diego	Documentación del proyecto QA Tester



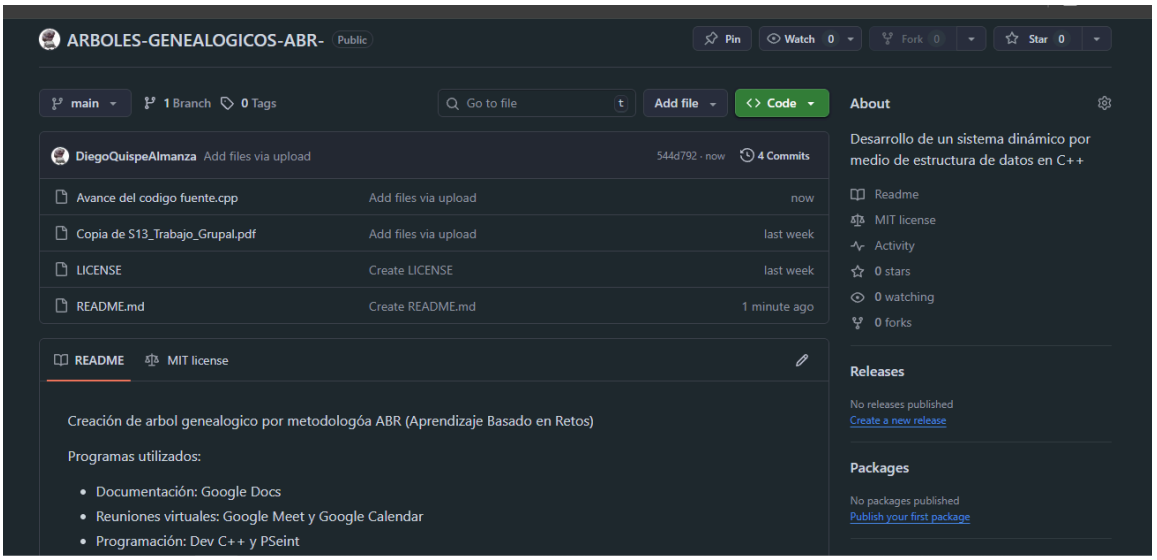
Semana 14

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

Enlace a la herramienta colaborativa

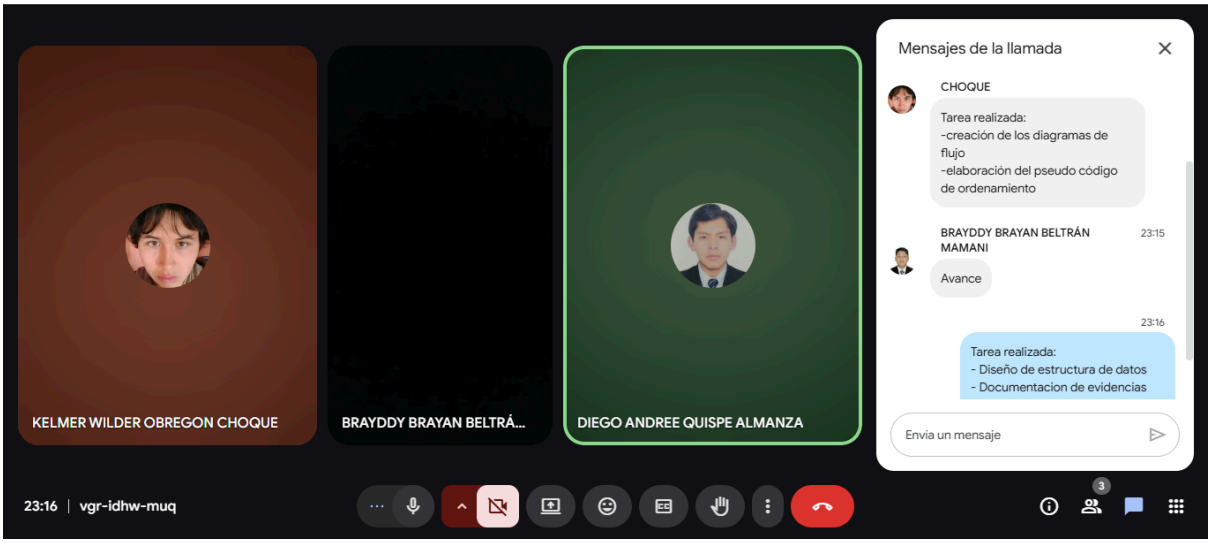
Semana 14



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

Evidencias trabajo colaborativo

Semana 14



Manual para usuario (= En progreso =)

Manual de Usuario - ABR

https://docs.google.com/document/d/1hS3Sr2z76dp2ssFXPdA_8Jt5tX3YsKKT8fkRAawl4ll/edit?tab=t.0

Acta de compromiso (= En progreso =)

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

Lista de fuentes consultadas (libros, artículos, sitios web), citadas según el formato ISO-690 Numérico

ANEXOS (OPCIONAL)

Código completo, resultados adicionales, diagramas complementarios u otro material relevante.