

UNIVERSIDAD CONTINENTAL

FACULTAD DE INGENIERÍA



**Universidad
Continental**

INFORME DE ESTRUCTURA DE DATOS

NRC: 29901

Asignatura: Estructura de datos

Docente: Osorio Contreras Rosario Delia

Tema

“ÁRBOL GENEALÓGICO - ABR”

INTEGRANTES:

- García Betancourt Israel Jedidias
- Leon Armas Luis Aram
- Ramírez Quillatupa Juan Diego
- Veliz Durand Vieri Del Piero

2025 - 10

Huancayo - Perú

ÍNDICE

| | |
|--|----------|
| 1. Descripción de estructuras de datos y operaciones | 1 |
| 1.1. Pseudocódigo para crear un árbol binario | 1 |
| 1.2. Operaciones implementadas | 1 |
| 2. Algoritmos principales..... | 3 |
| 2.1. Pseudocódigo para crear un árbol binario | 3 |
| 2.2. Pseudocódigo para realizar el recorrido de un árbol..... | 4 |
| 3. Diagramas de flujo | 4 |
| 4. Avance del código fuente | 4 |
| 4.1. Definición de estructuras..... | 4 |
| 4.2. Funciones principales del árbol | 5 |
| 4.3. Operaciones del sistema | 6 |
| 4.4. Menú y ejecución principal | 8 |
| 4.5. Capturas de la ejecución | 8 |

ABR - PROTOTIPO

1. Descripción de estructuras de datos y operaciones

1.1. Pseudocódigo para crear un árbol binario

Se define una estructura llamada “Nodo”, que representa a cada persona dentro del árbol genealógico.

Cada nodo contiene los siguientes campos:

- id: una cadena de texto que actúa como identificador único de la persona.
- nombre: el nombre de la persona.
- apellido: el apellido de la persona.
- padre: puntero al nodo que representa al padre (antecesor inmediato).
- izq: puntero al hijo izquierdo (primer hijo).
- der: puntero al hijo derecho (segundo hijo).

Esta estructura permite construir un árbol binario con jerarquía familiar, donde cada persona puede tener como máximo dos hijos directos y un único padre. Este modelo es útil para representar genealogías de manera estructurada y organizada.

1.2. Operaciones implementadas

1.2.1. Búsqueda por ID (buscarPorID)

- Permite localizar a una persona dentro del árbol usando su identificador único (id).
- Utiliza un recorrido recursivo en preorden (raíz, hijo izquierdo, hijo derecho).
- Si el nodo con el ID buscado se encuentra, se retorna su puntero; de lo contrario, se retorna NULL.

1.2.2. Verificación de ancestro (esAncestro)

- Determina si un nodo dado es ancestro directo de otro.
- Recorre hacia arriba utilizando el puntero padre desde el nodo objetivo hasta llegar a la raíz.
- Esta función es esencial para prevenir la creación de ciclos genealógicos al insertar nuevos miembros en el árbol.

1.2.3. Inserción de persona (insertarPersona)

- Permite agregar una nueva persona al árbol como hijo de un nodo padre ya existente.
- Se validan las siguientes condiciones antes de la inserción:
 - Que el padre exista en el árbol.

- Que la persona nueva no tenga ya un padre asignado.
 - Que no se cree un ciclo genealógico (verificación mediante `esAncestro`).
 - Que el padre tenga espacio disponible (máximo dos hijos).
- Si todo es válido, la nueva persona se enlaza como hijo izquierdo (`izq`) o derecho (`der`) del padre.

1.2.4. Recorridos del árbol (Preorden, Inorden y Postorden)

- Preorden (`preorden`): primero muestra el nodo actual, luego recorre el hijo izquierdo y finalmente el derecho.
- Inorden (`inorden`): primero recorre el hijo izquierdo, luego muestra el nodo actual, y después recorre el hijo derecho.
- Postorden (`postorden`): recorre primero ambos hijos (izquierdo y derecho) y finalmente muestra el nodo actual.
- Estos recorridos se usan para visualizar el árbol en distintos órdenes según el análisis deseado.

1.2.5. Mostrar ascendencia detallada (`mostrarAscendenciaDetallada`)

- Muestra la línea ascendente de una persona desde el nodo seleccionado hasta la raíz.
- Se presenta el nivel jerárquico de cada ancestro encontrado.
- Además, se detectan y muestran los hermanos del nodo en cada nivel (si existen).
- Esta operación proporciona una vista clara y estructurada del linaje familiar.

1.2.6. Añadir persona (`anadirPersona`)

- Solicita datos básicos como ID, nombre, apellido e ID del padre.
- Verifica que el ID sea único.
- Si el árbol está vacío, la persona se convierte en la raíz del árbol.
- Si no, se llama a `insertarPersona` para añadirla bajo el padre indicado.

1.2.7. Eliminación de persona (`eliminarPersona`)

- Permite eliminar un nodo del árbol solo si no tiene hijos.
- Si el nodo a eliminar es la raíz y no tiene hijos, el árbol queda vacío.
- Si el nodo tiene hijos, la eliminación es rechazada para mantener la integridad del árbol genealógico.

1.2.8. Submenú de métodos de búsqueda (`submenuMetodosBusqueda`)

- Ofrece un menú para que el usuario pueda ejecutar cualquiera de los tres recorridos del árbol.

- Después de mostrar los resultados, permite volver al menú principal.

1.2.9. Búsqueda de persona y ascendencia

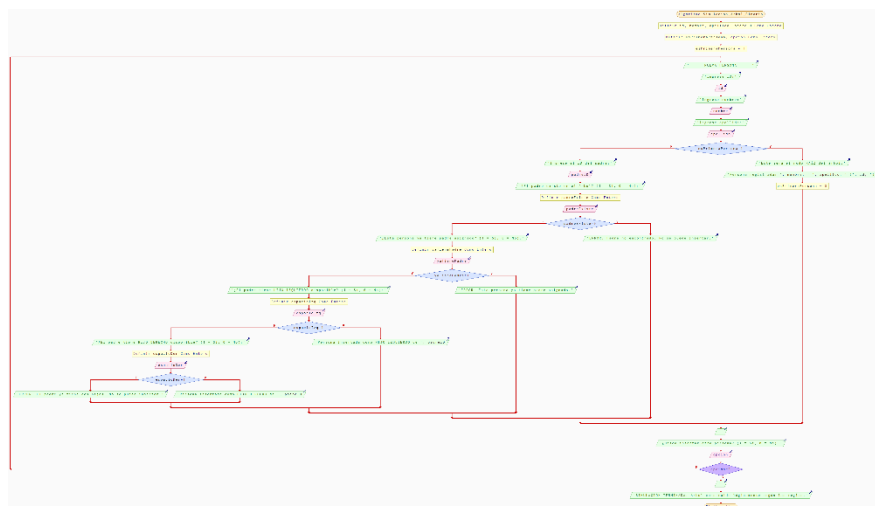
- Solicita un ID y utiliza buscarPorID para ubicar a la persona.
- Si se encuentra, se muestra la ascendencia detallada usando mostrarAscendenciaDetallada.

1.2.10. Menú principal (menu)

- Controla toda la interacción con el usuario mediante opciones por consola.
- Las opciones disponibles incluyen: agregar persona, eliminar persona, mostrar recorridos, buscar persona por ID y salir del programa.

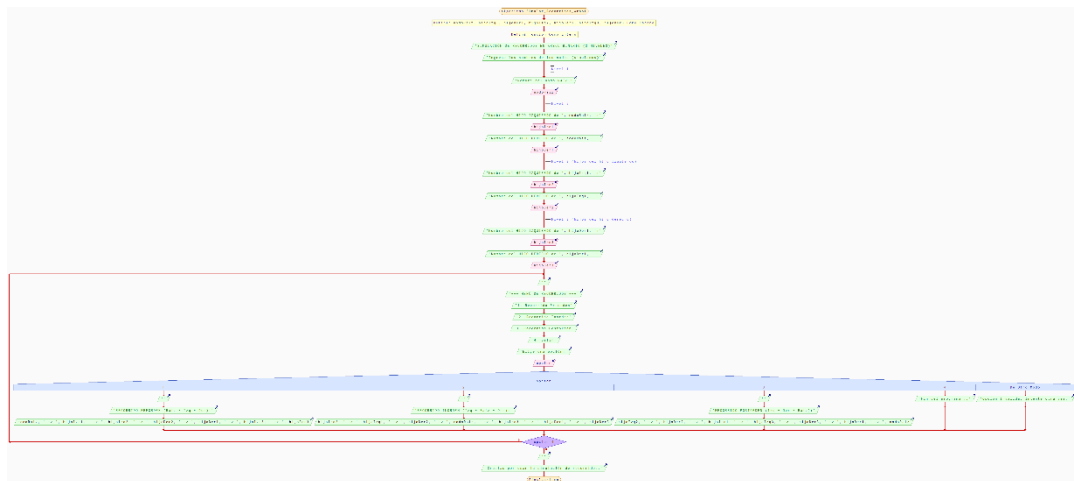
2. Algoritmos principales

2.1. Pseudocódigo para crear un árbol binario



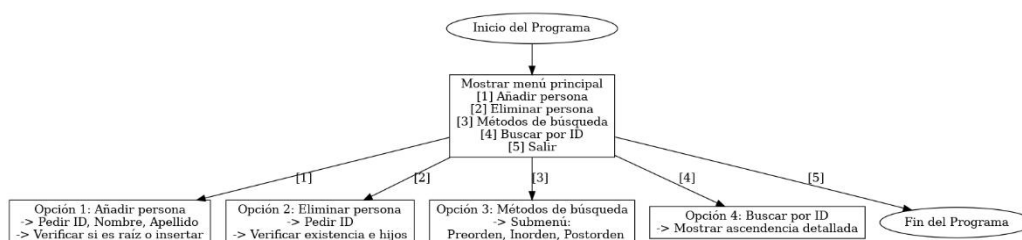
https://drive.google.com/drive/folders/1A_dJmSbkKeUvANefRnhPoYXFRTfMkJqz?usp=drive_link

2.2. Pseudocódigo para realizar el recorrido de un árbol



https://drive.google.com/drive/folders/1A_dJmSbkKeUvANefRnhPoYXFRTfMkJqz?usp=drive_link

3. Diagramas de flujo



4. Avance del código fuente

A continuación, se muestran capturas del código fuente desarrollado hasta el momento para el prototipo del árbol genealógico. Estas incluyen la estructura del nodo, funciones de inserción, eliminación, búsqueda, recorridos y el menú principal.

4.1. Definición de estructuras

```

1  #include <iostream>
2  #include <string>
3  #include <locale>
4  using namespace std;
5
6  // Definición de la estructura Nodo para representar a una persona en un árbol genealógico
7  struct Nodo {
8      string id;           // Identificador único de la persona
9      string nombre;       // Nombre de la persona
10     string apellido;      // Apellido de la persona
11     Nodo* padre;          // Puntero al nodo del padre
12     Nodo* izq;            // Hijo izquierdo (puede representar al primer hijo)
13     Nodo* der;            // Hijo derecho (puede representar al segundo hijo)
14 };
15

```

4.2. Funciones principales del árbol

4.2.1. Función para buscar persona por ID

```
16 // Función para buscar una persona en el árbol por su ID
17 Nodo* buscarPorID(Nodo* nodo, const string& id) {
18     if (nodo == NULL) return NULL; // Si el nodo es nulo, no se encontró
19     if (nodo->id == id) return nodo; // Si el ID coincide, se encontró
20     Nodo* encontrado = buscarPorID(nodo->izq, id); // Buscar recursivamente en el subárbol izquierdo
21     if (encontrado) return encontrado; // Si se encontró en el subárbol izquierdo, devolverlo
22     return buscarPorID(nodo->der, id); // Si no, buscar en el subárbol derecho
23 }
24
```

4.2.2. Verificación de ancestros

```
25 // Función para verificar si un nodo es ancestro de otro (para evitar ciclos)
26 bool esAncestro(Nodo* posibleAncestro, Nodo* nodo) {
27     while (nodo != NULL) {
28         if (nodo == posibleAncestro) // Si encontramos el nodo en la cadena ascendente, es ancestro
29             return true;
30         nodo = nodo->padre; // Subir al padre
31     }
32     return false; // Si se llega al final sin coincidencias, no es ancestro
33 }
```

4.2.3. Insertar persona al árbol

```
35 // Función para insertar una nueva persona en el árbol bajo un padre especificado
36 bool insertarPersona(Nodo* raiz, Nodo* nueva, const string& padreID) {
37     Nodo* padre = buscarPorID(raiz, padreID); // Buscar el nodo del padre
38     if (padre == NULL) {
39         cout << "Error: Padre no encontrado.\n"; // Validar existencia del padre
40         return false;
41     }
42     if (nueva->padre != NULL) {
43         // Validar que la nueva persona no tenga ya un padre asignado
44         cout << "Error: Esta persona ya tiene un padre asignado (" << nueva->padre->id << ").\n";
45         return false;
46     }
47     if (esAncestro(nueva, padre)) {
48         // Prevenir ciclos: no se puede insertar un ancestro como hijo de su descendiente
49         cout << "Error: No se puede insertar (crear a un ciclo).\n";
50         return false;
51     }
52     nueva->padre = padre; // Asignar el padre
53     if (padre->izq == NULL) {
54         padre->izq = nueva; // Insertar como hijo izquierdo si está disponible
55         return true;
56     } else if (padre->der == NULL) {
57         padre->der = nueva; // Insertar como hijo derecho si está disponible
58         return true;
59     } else {
60         cout << "Error: El padre ya tiene dos hijos.\n"; // Si ya tiene dos hijos, no se puede insertar
61         return false;
62     }
63 }
```

4.2.4. Recorridos del árbol

```
65 // Recorrido en preorden: visita raíz, luego hijo izquierdo y luego derecho
66 void preorden(Nodo* nodo) {
67     if (nodo != NULL) {
68         cout << nodo->nombre << " " << nodo->apellido << " (" << nodo->id << ")\n";
69         preorden(nodo->izq); // Visitar hijo izquierdo
70         preorden(nodo->der); // Visitar hijo derecho
71     }
72 }
73
74 // Recorrido en inorden: hijo izquierdo, luego raíz, luego derecho
75 void inorden(Nodo* nodo) {
76     if (nodo != NULL) {
77         inorden(nodo->izq); // Visitar hijo izquierdo
78         cout << nodo->nombre << " " << nodo->apellido << " (" << nodo->id << ")\n";
79         inorden(nodo->der); // Visitar hijo derecho
80     }
81 }
82
83 // Recorrido en postorden: hijo izquierdo, hijo derecho, luego raíz
84 void postorden(Nodo* nodo) {
85     if (nodo != NULL) {
86         postorden(nodo->izq); // Visitar hijo izquierdo
87         postorden(nodo->der); // Visitar hijo derecho
88         cout << nodo->nombre << " " << nodo->apellido << " (" << nodo->id << ")\n";
89     }
90 }
```

4.2.5. Mostrar ascendencia detallada

```
93 void mostrarAscendenciaDetallada(Nodo* persona) {
94     if (persona == NULL) {
95         cout << "Persona no encontrada.\n"; // Verifica si la persona existe
96         return;
97     }
98
99     cout << "\nPersona encontrada: " << persona->nombre << " " << persona->apellido << " (" << persona->id << ")\n";
100     Nodo* camino[100]; // Arreglo para guardar la ruta hasta la raíz
101     int nivel = 0;
102     Nodo* actual = persona;
103     while (actual != NULL) {
104         camino[nivel++] = actual; // Guarda cada nodo del camino hacia arriba
105         actual = actual->padre;
106     }
107
108     // Recorre el camino desde la raíz hasta la persona buscada
109     for (int i = nivel - 1; i >= 0; i--) {
110         actual = camino[i];
111         cout << "\nNivel " << (nivel - i) << ":\n ";
112
113         if (actual->padre != NULL) {
114             Nodo* hermano = NULL;
115
116             // Determina si existe un hermano (el otro hijo del padre)
117             if (actual->padre->izq == actual && actual->padre->der != NULL)
118                 hermano = actual->padre->der;
119             else if (actual->padre->der == actual && actual->padre->izq != NULL)
120                 hermano = actual->padre->izq;
121
122             // Imprime la información de la persona en ese nivel
123             cout << actual->nombre << " " << actual->apellido << " (" << actual->id;
124             if (i == nivel - 1) cout << ") (Raiz)"; // Es la raíz
125             else if (i == 0) cout << ") (buscado)"; // Es la persona buscada
126             else cout << ") (padre)"; // Es un ancestro intermedio
127
128             // Si hay hermano, mostrar su información también
129             if (hermano != NULL) {
130                 cout << " " << hermano->nombre << " " << hermano->apellido << " (" << hermano->id;
131                 if (i == 0) cout << ") (hermano)"; // Hermano del nodo buscado
132                 else cout << ")"; // Hermano de un ancestro
133             } else {
134                 cout << "\n"; // Nueva línea si no hay hermano
135             }
136         } else {
137             // Es la raíz del árbol (no tiene padre)
138             cout << actual->nombre << " " << actual->apellido << " (" << actual->id << ") (Raiz)\n";
139         }
140     }
141     cout << "\n";
}
```

4.3. Operaciones del sistema

4.3.1. Añadir persona

```
145 // Permite añadir una nueva persona al árbol
146 void anadirPersona(Nodo*& raiz) {
147     Nodo* nueva = new Nodo(); // Crear nuevo nodo
148
149     cout << "ID unico: ";
150     getline(cin, nueva->id);
151
152     // Verificar si ya existe una persona con ese ID
153     if (raiz != NULL && buscarPorID(raiz, nueva->id) != NULL) {
154         cout << "Error: Ya existe una persona con ese ID.\n";
155         delete nueva; // Liberar memoria si ya existe
156         return;
157     }
158
159     // Ingresar nombre y apellido
160     cout << "Nombre: ";
161     getline(cin, nueva->nombre);
162     cout << "Apellido: ";
163     getline(cin, nueva->apellido);
164
165     // Inicializar punteros
166     nueva->izq = nueva->der = nueva->padre = NULL;
167
168     if (raiz == NULL) {
169         raiz = nueva; // Si el árbol está vacío, esta persona es la raíz
170         cout << "Anadido como raíz del árbol.\n";
171     } else {
172         // Solicitar ID del padre y tratar de insertar
173         string padreID;
174         cout << "ID del padre: ";
175         getline(cin, padreID);
176         if (!insertarPersona(raiz, nueva, padreID)) {
177             delete nueva; // Si falla, liberar memoria
178         } else {
179             cout << "Persona anadida al árbol.\n";
180         }
181     }
182 }
183
```

4.3.2. Eliminar persona


```

184 // Permite eliminar una persona del árbol si no tiene hijos
185 void eliminarPersona(Nodo*& raiz, const string& id) {
186     Nodo* objetivo = buscarPorID(raiz, id); // Buscar a la persona
187     if (objetivo == NULL) {
188         cout << "Error: Persona no encontrada.\n";
189         return;
190     }
191
192     // No se permite eliminar personas con hijos
193     if (objetivo->izq != NULL || objetivo->der != NULL) {
194         cout << "Error: No se puede eliminar. Esta persona tiene hijos.\n";
195         return;
196     }
197
198     // Si es la raíz y no tiene hijos, eliminarla y vaciar el árbol
199     if (objetivo == raiz) {
200         delete raiz;
201         raiz = NULL;
202         cout << "Raiz eliminada. Arbol vacio.\n";
203         return;
204     }
205
206     // Si no es la raíz, eliminarla del padre
207     Nodo* padre = objetivo->padre;
208     if (padre->izq == objetivo)
209         padre->izq = NULL;
210     else if (padre->der == objetivo)
211         padre->der = NULL;
212
213     delete objetivo; // Liberar memoria del nodo
214     cout << "Persona eliminada correctamente.\n";
215 }

```

4.3.3. Submenú de recorridos

```

218 void submenuMetodosBusqueda(Nodo* raiz) {
219     int subopcion;
220     do {
221         cout << "\n--- METODOS DE BUSQUEDA ---\n";
222         cout << "[1]: Mostrar Preorden\n";
223         cout << "[2]: Mostrar Inorden\n";
224         cout << "[3]: Mostrar Postorden\n";
225         cout << "[4]: Volver al menu principal\n";
226
227         // Validar que se elija una opción válida
228         do {
229             cout << "Elegir opcion: ";
230             cin >> subopcion;
231             if (subopcion < 1 || subopcion > 4) {
232                 cout << "ERROR, opcion incorrecta.\n";
233             }
234         } while (subopcion < 1 || subopcion > 4);
235
236         cin.ignore(); // Limpiar buffer de entrada
237
238         switch (subopcion) {
239             case 1:
240                 // Mostrar recorrido preorden
241                 cout << "\n-- Recorrido Preorden --\n";
242                 preorden(raiz);
243                 break;
244             case 2:
245                 // Mostrar recorrido inorden
246                 cout << "\n-- Recorrido Inorden --\n";
247                 inorden(raiz);
248                 break;
249             case 3:
250                 // Mostrar recorrido postorden
251                 cout << "\n-- Recorrido Postorden --\n";
252                 postorden(raiz);
253                 break;
254             case 4:
255                 // Volver al menú principal
256                 cout << "Regresando al menu principal...\n";
257                 break;
258         }
259
260         // Pausar y limpiar pantalla si no se sale
261         if (subopcion != 4) {
262             system("pause");
263             system("cls");
264         }
265     } while (subopcion != 4);
266 }

```

4.3.4. Buscar por ID y mostrar ascendencia

```
269 // Permite buscar una persona por su ID y mostrar su ascendencia detallada
270 void buscarPersonaYAscendencia(Nodo* raiz) {
271     string id;
272     cout << "Ingrese el ID de la persona: ";
273     getline(cin, id); // Leer el ID de la persona
274     Nodo* persona = buscarPorID(raiz, id); // Buscar persona
275     mostrarAscendenciaDetallada(persona); // Mostrar su ascendencia
276 }
```

4.4. Menú y ejecución principal

4.4.1. Menú principal

```
279 // Función para el menú principal, que genera como es, maneja, muestra, genera y genera
280 void menu() {
281     Nodo* raiz = NULL; // Arbol inicialmente vacío
282     int op;
283     do {
284         // Mostrar opciones del menú principal
285         cout << "*** MENU ARBOL GENEALOGICO ***\n";
286         cout << "[1]: Anadir persona\n";
287         cout << "[2]: Eliminar persona\n";
288         cout << "[3]: Metodos de busqueda (Preorden, Inorden, Postorden)\n";
289         cout << "[4]: Buscar persona por ID (ascendencia)\n";
290         cout << "[5]: Salir\n";
291
292         // Validar opción ingresada
293         do {
294             cout << "Elegir opcion: ";
295             cin >> op;
296             if (op < 1 || op > 5) {
297                 cout << "ERROR, opcion incorrecta\n";
298             }
299         } while (op < 1 || op > 5);
300
301         cin.ignore(); // Limpiar buffer
302
303         // Ejecutar opción elegida
304         switch (op) {
305             case 1:
306                 // Añadir nueva persona al árbol
307                 anadirPersona(raiz);
308                 break;
309             case 2:
310                 // Eliminar persona del árbol
311                 string id;
312                 cout << "ID de la persona a eliminar: ";
313                 getline(cin, id);
314                 eliminarPersona(raiz, id);
315                 break;
316             case 3:
317                 // Acceder a submenú de métodos de recorrido
318                 submenuMetodosBusqueda(raiz);
319                 break;
320             case 4:
321                 // Buscar persona y mostrar su ascendencia
322                 buscarPersonaYAscendencia(raiz);
323                 break;
324             case 5:
325                 // Salir del programa
326                 break;
327         }
328
329         // Pausar y limpiar pantalla si no se ha elegido salir
330         if (op != 5) {
331             system("pause");
332             system("cls");
333         }
334     } while (op != 5);
335 }
```

4.4.2. Función main

```
339 // Función principal del programa
340 int main() {
341     setlocale(LC_ALL, "Spanish"); // Configura la codificación para admitir caracteres en español
342     menu(); // Llama al menú principal
343     return 0; // Fin del programa
344 }
```

4.5. Capturas de la ejecución

4.5.1. Menú

```
*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 1
```

4.5.2. Añadir Persona

```
*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 1
ID unico: 1235
Nombre: María Alberta
Apellido: Valde Tapia
ID del padre: 1234
Persona anadida al arbol.
Presione una tecla para continuar . . .
```

4.5.3. Eliminar Persona

```
*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 2
ID de la persona a eliminar: 1235
Persona eliminada correctamente.
Presione una tecla para continuar . . .
```

4.5.4. Métodos de búsqueda

```
*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 3

--- METODOS DE BUSQUEDA ---
[1]: Mostrar Preorden
[2]: Mostrar Inorden
[3]: Mostrar Postorden
[4]: Volver al menu principal
Elegir opcion: 1

-- Recorrido Preorden --
Juan Eduardo Rojas Rivas (1234)
Juan Ramirez (1236)
Pedro Quillatupa (1239)
Presione una tecla para continuar . . .
```

```

--- METODOS DE BUSQUEDA ---
[1]: Mostrar Preorden
[2]: Mostrar Inorden
[3]: Mostrar Postorden
[4]: Volver al menu principal
Elegir opcion: 2

-- Recorrido Inorden --
Juan  Ramirez (1236)
Juan Eduardo Rojas Rivas (1234)
Pedro Quillatupa (1239)
Presione una tecla para continuar . . .

```

```

--- METODOS DE BUSQUEDA ---
[1]: Mostrar Preorden
[2]: Mostrar Inorden
[3]: Mostrar Postorden
[4]: Volver al menu principal
Elegir opcion: 3

-- Recorrido Postorden --
Juan  Ramirez (1236)
Pedro Quillatupa (1239)
Juan Eduardo Rojas Rivas (1234)
Presione una tecla para continuar . . .

```

4.5.5. Buscar por ID

```

*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 4
Ingrese el ID de la persona: 1234

Persona encontrada: Juan Eduardo Rojas Rivas (1234)

Nivel 1:
    Juan Eduardo Rojas Rivas (1234) (Raiz)

Presione una tecla para continuar . . .

```

4.5.6. Rebalancear árbol

```
*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 5
Arbol rebalanceado correctamente.
Presione una tecla para continuar . . .
```

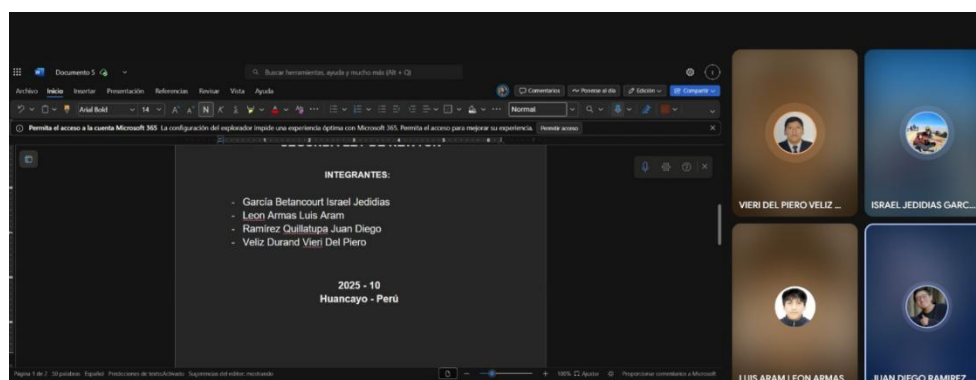
4.5.6. Salir

```
*** MENU ARBOL GENEALOGICO ***
[1]: Anadir persona
[2]: Eliminar persona
[3]: Metodos de busqueda (Preorden, Inorden, Postorden)
[4]: Buscar persona por ID (ascendencia)
[5]: Rebalancear arbol
[6]: Salir
Elegir opcion: 6

-----
Process exited after 459.5 seconds with return value 0
Presione una tecla para continuar . . .
```

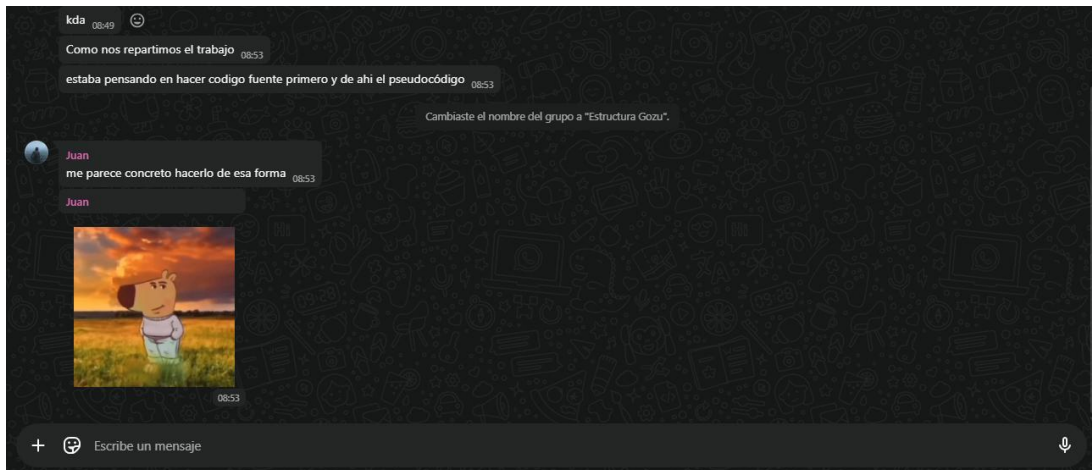
5. Anexos

Anexo 1



En esta imagen se observa la reunión que tuvo el grupo, para coordinar el ABR-PROTOTIPO

Anexo 2



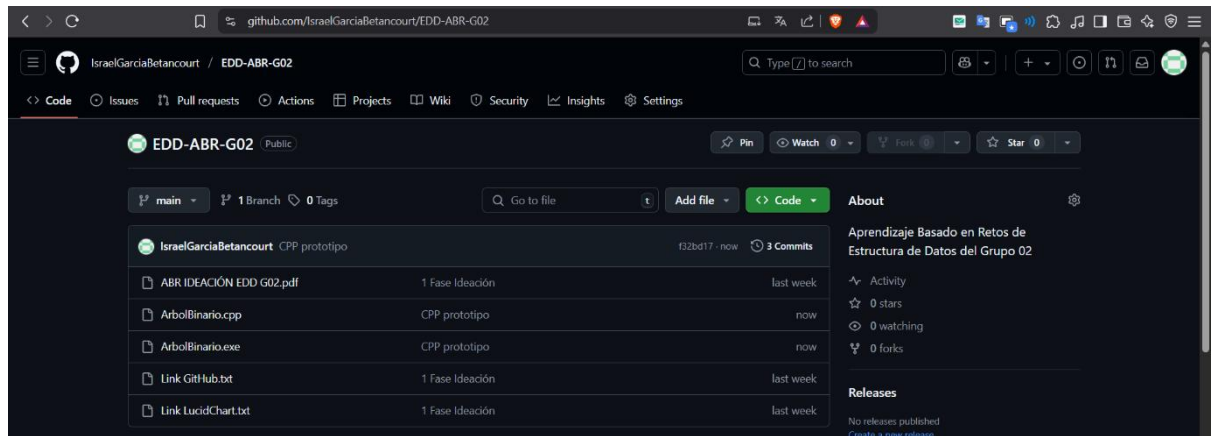
En esta imagen se observa la coordinación mediante un grupo en WhatsApp

Anexo 3



En esta imagen se observa la coordinación en el aula de clases para coordinar el ABR – Prototipo

Anexo 4



En esta imagen se observa el repositorio donde se almacenará todos los archivos correspondientes al ABR

<https://github.com/IsraelGarciaBetancourt/EDD-ABR-G02>