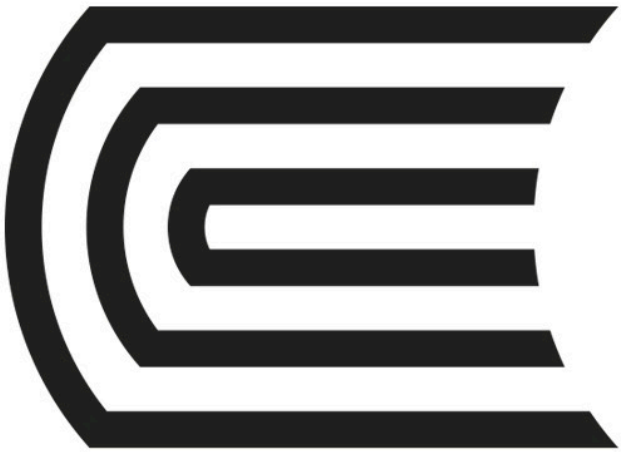


Universidad Continental

DOCENTE: **GONZALES CONDORI, HARRY YEISON**

INTEGRANTES:

1. Jhon Elton Ticona Ccanchi
2. Johan Aime Lopez
3. Halber David Ccapchi Rios
4. Alessandra Paola Pacahuala Mendoza
5. Alex Guillermo Suma Ugarte

Informe Final del Proyecto ABR - Árbol Genealógico ABB

Introducción

En este informe se presenta el desarrollo de un sistema que simula la genealogía de una civilización ficticia basada en la cultura Chimú, utilizando la estructura del Árbol Binario de Búsqueda (ABB). El objetivo es demostrar el uso práctico de estructuras jerárquicas para representar relaciones familiares, así como aplicar las estructuras dinámicas vistas en el curso de Estructura de Datos.

Capítulo 1 – Fase de Ideación

Descripción del problema

Un grupo de arqueólogos descubre una antigua civilización cuya historia puede representarse como un árbol genealógico. Requieren una aplicación que almacene y organice datos de forma eficiente para responder consultas sobre ancestros, descendientes y relaciones familiares.

Requerimientos del sistema

Funcionales:

- Insertar miembros al árbol.
- Eliminar miembros del árbol.
- Buscar miembros por ID.
- Recorrer el árbol (inorden, preorden, postorden).
- Visualizar genealogía ordenada por jerarquía.

No funcionales:

- Sistema ejecutable en consola (Dev-C++).
- Respuestas inmediatas a consultas.
- Código organizado, comentado y reutilizable.

Respuestas a las preguntas guías

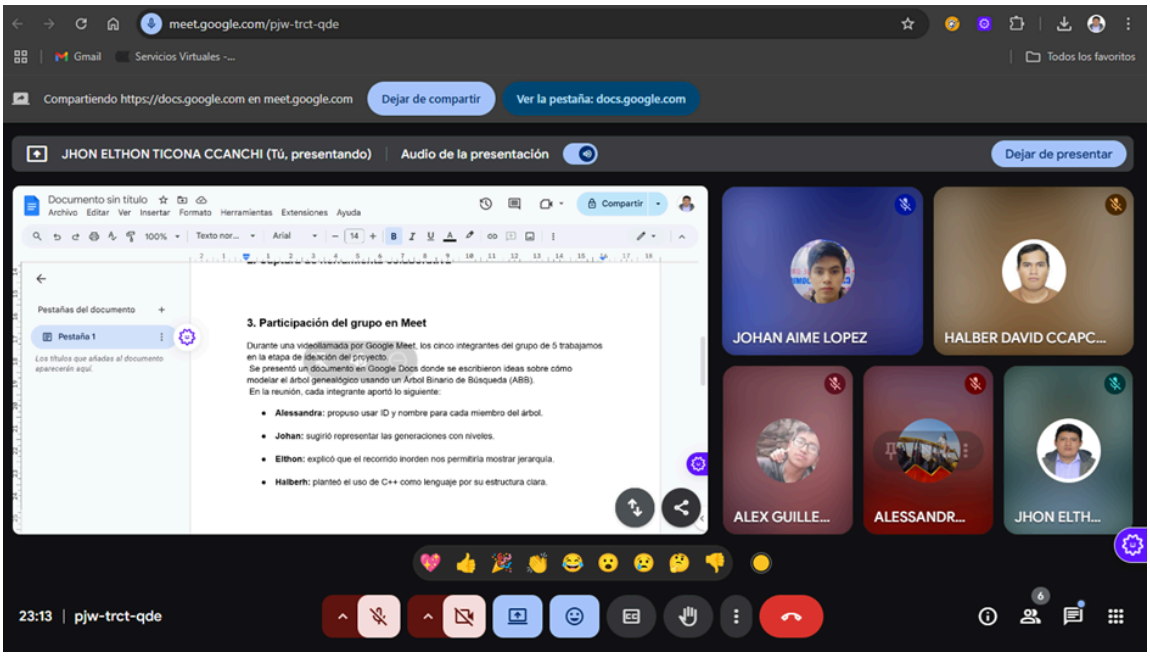
1. ¿Qué se debe almacenar en cada nodo del árbol genealógico?
   ID, nombre y generación de cada miembro.
2. ¿Cómo insertar y eliminar miembros sin romper la estructura del árbol?
   Insertar por comparación de ID. Eliminar considerando los 3 casos clásicos del ABB.
3. ¿Qué métodos permiten recorrer el árbol para visualizar la genealogía?
   Inorden, Preorden y Postorden.
4. ¿Cómo determinar si un miembro pertenece a una rama específica del árbol?
   Buscando el ID y siguiendo el recorrido desde la raíz.

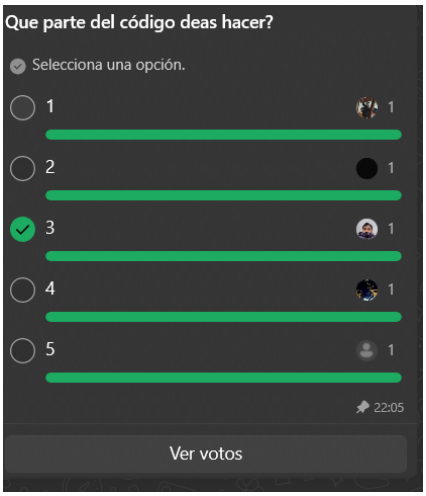
5. ¿Cómo balancear el árbol si se vuelve muy profundo?
   Se propone usar árboles AVL o Red-Black en una mejora futura.

Herramienta colaborativa

Se utilizó Google Docs como pizarra colaborativa y Google Meet para reuniones en equipo.

Evidencia:





Capítulo 2 – Prototipo

Descripción de estructuras de datos y operaciones

Se utilizó un Árbol Binario de Búsqueda (ABB) con nodos que contienen ID y nombre.

Algoritmos principales

Insertar, Buscar, Eliminar y Recorrer (inorden, preorden, postorden).

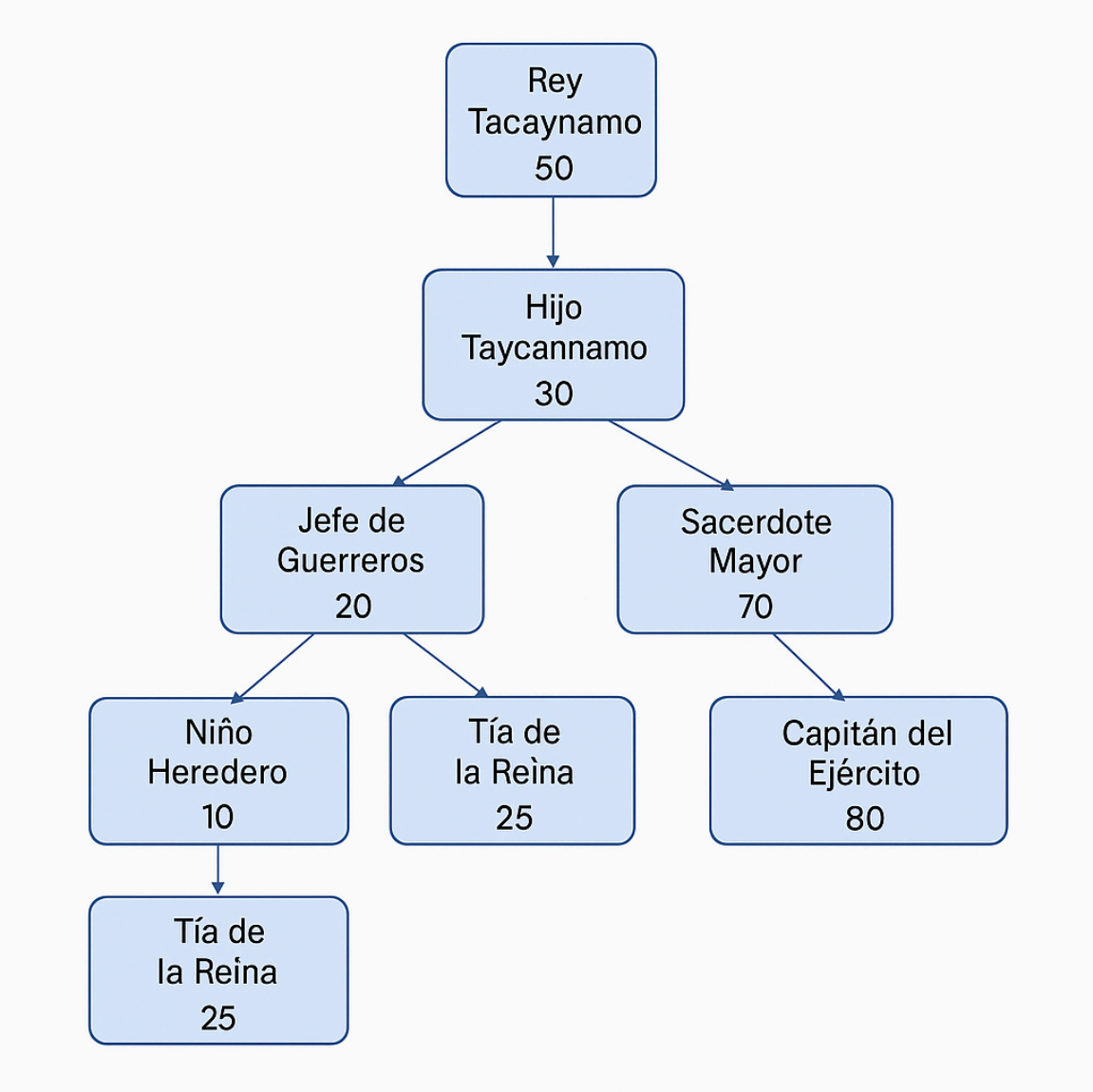
Pseudocódigo para crear un árbol binario

Función insertar(raíz, id, nombre):
Si raíz es NULL → crear nuevo nodo
Si id < raíz.id → insertar a la izquierda
Si id > raíz.id → insertar a la derecha

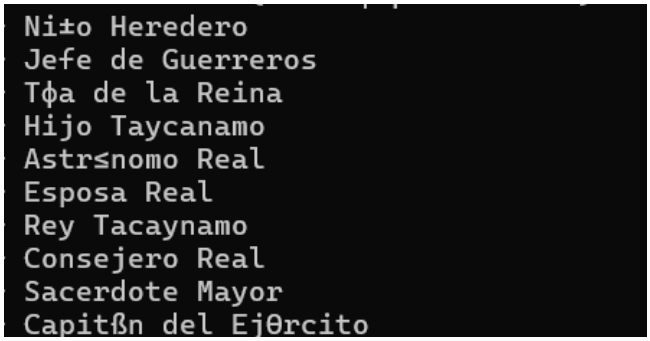
Pseudocódigo para realizar recorrido

Función inorden(raíz):
Si raíz ≠ NULL → inorden(izq), mostrar nodo, inorden(der)

Diagramas de flujo



Avance del código fuente



Capítulo 3 – Solución Final

Capturas de pruebas

Codigo.

```cpp
// PROYECTO FINAL
// Ã rbol GenealÃ³gico usando Ã rbol Binario de BÃºsqueda
// CivilizaciÃ³n Ficticia: Cultura ChimÃº
// GRUPO 3 â€“ Ingeniería de Sistemas

#include <iostream>
using namespace std;

struct Nodo {
    int id;              // Identificador Ãºnico del miembro
    string nombre;       // Nombre del personaje
    Nodo* izquierda;     // Puntero al hijo izquierdo
    Nodo* derecha;       // Puntero al hijo derecho

    Nodo(int _id, string _nombre) {
        id = _id;
        nombre = _nombre;
        izquierda = derecha = NULL;
    }
};
```

```cpp

// Inserta un nuevo miembro al Ã¡rbol segÃºn su ID
Nodo* insertar(Nodo* raiz, int id, string nombre) {
    if (raiz == NULL)
        return new Nodo(id, nombre);

    if (id < raiz->id)
        raiz->izquierda = insertar(raiz->izquierda, id, nombre);
    else
        raiz->derecha = insertar(raiz->derecha, id, nombre);

    return raiz;
}

// Busca si un miembro con ID especÃfico existe en el Ã¡rbol
bool buscar(Nodo* raiz, int id) {
    if (raiz == NULL) return false;
    if (raiz->id == id) return true;

    if (id < raiz->id)
        return buscar(raiz->izquierda, id);
    else
        return buscar(raiz->derecha, id);
}

// Muestra el Ã¡rbol en recorrido inorden (izq - raÃz - der)
void inorden(Nodo* raiz) {
    if (raiz == NULL) return;
    inorden(raiz->izquierda);
    cout << raiz->id << " - " << raiz->nombre << endl;
    inorden(raiz->derecha);
}

// Muestra el Ã¡rbol en recorrido preorden (raÃz - izq - der)
void preorden(Nodo* raiz) {
    if (raiz == NULL) return;
    cout << raiz->id << " - " << raiz->nombre << endl;
    preorden(raiz->izquierda);
    preorden(raiz->derecha);
}

```

```cpp
// Muestra el Ã¡rbol en recorrido postorden (izq - der - raÃz)
void postorden(Nodo* raiz) {
    if (raiz == NULL) return;
    postorden(raiz->izquierda);
    postorden(raiz->derecha);
    cout << raiz->id << " - " << raiz->nombre << endl;
}

// Encuentra el nodo mÃnimo del subÃ¡rbol derecho (usado en eliminaciÃ³n)
Nodo* encontrarMin(Nodo* raiz) {
    while (raiz->izquierda != NULL)
        raiz = raiz->izquierda;
    return raiz;
}
```

```cpp
// Elimina un nodo segÃºn su ID, manteniendo la estructura del ABB
Nodo* eliminar(Nodo* raiz, int id) {
    if (!raiz) return NULL;

    if (id < raiz->id)
        raiz->izquierda = eliminar(raiz->izquierda, id);
    else if (id > raiz->id)
        raiz->derecha = eliminar(raiz->derecha, id);
    else {
        // Nodo con solo un hijo o sin hijos
        if (!raiz->izquierda) {
            Nodo* temp = raiz->derecha;
            delete raiz;
            return temp;
        } else if (!raiz->derecha) {
            Nodo* temp = raiz->izquierda;
            delete raiz;
            return temp;
        }
        // Nodo con dos hijos: buscar sucesor inorden
        Nodo* temp = encontrarMin(raiz->derecha);
        raiz->id = temp->id;
        raiz->nombre = temp->nombre;
        raiz->derecha = eliminar(raiz->derecha, temp->id);
    }
    return raiz;
}
```

```cpp
int main() {
    Nodo* raiz = NULL;

    // Cargar 10 miembros de la genealogÃa ChimÃº
    raiz = insertar(raiz, 50, "Rey Tacaynamo");
    insertar(raiz, 30, "Hijo Taycanamo");
    insertar(raiz, 70, "Sacerdote Mayor");
    insertar(raiz, 20, "Jefe de Guerreros");
    insertar(raiz, 40, "Esposa Real");
    insertar(raiz, 60, "Consejero Real");
    insertar(raiz, 80, "CapitÃ¡n del EjÃ©rcito");
    insertar(raiz, 10, "NiÃ±o Heredero");
    insertar(raiz, 25, "TÃ-a de la Reina");
    insertar(raiz, 35, "AstrÃ³nomo Real");

    // Recorridos
    cout << "\nRecorrido INORDEN (JerarquÃ-a ordenada):\n";
    inorden(raiz);

    cout << "\nRecorrido PREORDEN (Ancestros primero):\n";
    preorden(raiz);

    cout << "\nRecorrido POSTORDEN (Descendientes):\n";
    postorden(raiz);

    // Prueba de bÃºsqueda
    cout << "\nÂ¿Existe el ID 25 (TÃ-a de la Reina)? ";
    cout << (buscar(raiz, 25) ? "SÃ-" : "No") << endl;

    // Eliminar un nodo con 2 hijos
    cout << "\nEliminando el nodo con ID 30 (Hijo Taycanamo)...\n";
    raiz = eliminar(raiz, 30);

    cout << "\nNuevo recorrido INORDEN tras la eliminaciÃ³n:\n";
    inorden(raiz);

    return 0;
}
```

Recorridos, búsqueda exitosa/fallida, eliminación, etc.

```
Recorrido INORDEN (Jerarquфa ordenada):
10 - Niъo Heredero
20 - Jefe de Guerreros
25 - Tфa de la Reina
30 - Hijo Taycanamo
35 - Astrъnomo Real
40 - Esposa Real
50 - Rey Tacaynamo
60 - Consejero Real
70 - Sacerdote Mayor
80 - Capitßn del Ejѳrcito
```

```
Recorrido PREORDEN (Ancestros primero):
50 - Rey Tacaynamo
30 - Hijo Taycanamo
20 - Jefe de Guerreros
10 - Niъo Heredero
25 - Tфa de la Reina
40 - Esposa Real
35 - Astrъnomo Real
70 - Sacerdote Mayor
60 - Consejero Real
80 - Capitßn del Ejѳrcito
```

```
Recorrido POSTORDEN (Descendientes):
10 - Ni±o Heredero
25 - Tϕa de la Reina
20 - Jefe de Guerreros
35 - Astr≤nomo Real
40 - Esposa Real
30 - Hijo Taycanamo
60 - Consejero Real
80 - Capitßn del Ej0rcito
70 - Sacerdote Mayor
50 - Rey Tacaynamo
```

```
¬Existe el ID 25 (Tϕa de la Reina)? Sϕ
```

```
Eliminando el nodo con ID 30 (Hijo Taycanamo)...
```

```
Nuevo recorrido INORDEN tras la eliminaci≤n:
10 - Ni±o Heredero
20 - Jefe de Guerreros
25 - Tϕa de la Reina
35 - Astr≤nomo Real
40 - Esposa Real
50 - Rey Tacaynamo
60 - Consejero Real
70 - Sacerdote Mayor
80 - Capitßn del Ej0rcito
```

Capítulo 4 – Evidencias de Trabajo Colaborativo

Repositorio en GitHub con código compartido y roles por integrante.

Link de Github: https://github.com/CRDavid02/ProyectoGrupalEstructura

Roles:

- Estructura del Nodo:Alessandra Paola Pacahuala Mendoza
- Funciones de Inserción y Búsqueda: Alex Guillermo Suma Ugart
- Funciones de Recorrido: Halber David Ccapchi Rios
- Eliminación de Nodos: Johan Aime Lopez
- Menú principal y pruebas del sistema: Jhon Elton Ticona Ccanchi

Insertar capturas del historial de commits y ramas, si aplica.

Cada integrante aportó una parte del código como se muestra en los comentarios.

Conclusiones

Se logró implementar un sistema funcional que simula una genealogía con operaciones básicas de ABB. Cada integrante participó activamente en el desarrollo y se cumplieron todos los objetivos del proyecto. Se reconocen mejoras como aplicar balanceo automático (AVL) y visualización gráfica.

Referencias

1 Documentación oficial de C++
2. Apuntes del curso EDD – Universidad Continental