ESTRUCTURA DE DATOS

Integrantes:

- Caso Caysahuana Henrik Anderson (100%)

- Yupanqui Suarez Jesus Angel (100%)

- Cabrera Ortega Dhastin Ray (100%)

- Quinte Perez Erick Zahid (100%)

Capítulo 2 - Prototipo

2.1 Descripción de estructuras de datos y operaciones:

En este proyecto, se usa un Árbol Binario de Búsqueda (ABB) como la base de datos principal para el árbol familiar de una vieja cultura. Elegimos esto para ordenar bien la info familiar, haciendo que meter datos, buscar, recorrer y quitar sea fácil y rápido.

Cómo es el Árbol Binario de Búsqueda
Un Árbol Binario de Búsqueda es como un árbol con nodos, donde cada nodo tiene hasta dos "hijos": uno a la izquierda y otro a la derecha. Lo clave del ABB es que:

Si algo es menor que el nodo, va a la izquierda.
Si algo es mayor que el nodo, va a la derecha.

Esto ayuda a tener todo en orden, para que buscar y recorrer sea más fácil.
Qué guarda cada nodo:
Cada nodo es una persona en el árbol familiar. Guarda esto:

ID: Un número único para cada persona.
Nombre: El nombre completo de la persona.
Fecha de nacimiento: Para saber cuándo nació.
Puntero al hijo izquierdo: El hijo con un ID menor.
Puntero al hijo derecho: El hijo con un ID mayor.

Cosas importantes que se pueden hacer
Con el ABB, podemos hacer varias cosas importantes para el sistema del árbol familiar:

a) Meter datos

Agregar nuevas personas al árbol. Se respeta la regla del ABB, poniendo a la persona a la izquierda o derecha según su ID.

b) Buscar
Encontrar a alguien por su ID (o nombre), revisando solo una parte del árbol. Así, la búsqueda es rápida, en tiempo O(log n) en promedio.

c) Recorridos
Hay tres formas de recorrer el árbol para verlo:

- Inorden (Izquierda – Raíz – Derecha): Muestra a todos en orden de menor a mayor.
- Preorden (Raíz – Izquierda – Derecha): Muestra primero a los ancestros y luego a sus hijos, para ver cómo están organizados.
- Postorden (Izquierda – Derecha – Raíz): Para ver ramas completas o quitar nodos ordenadamente.

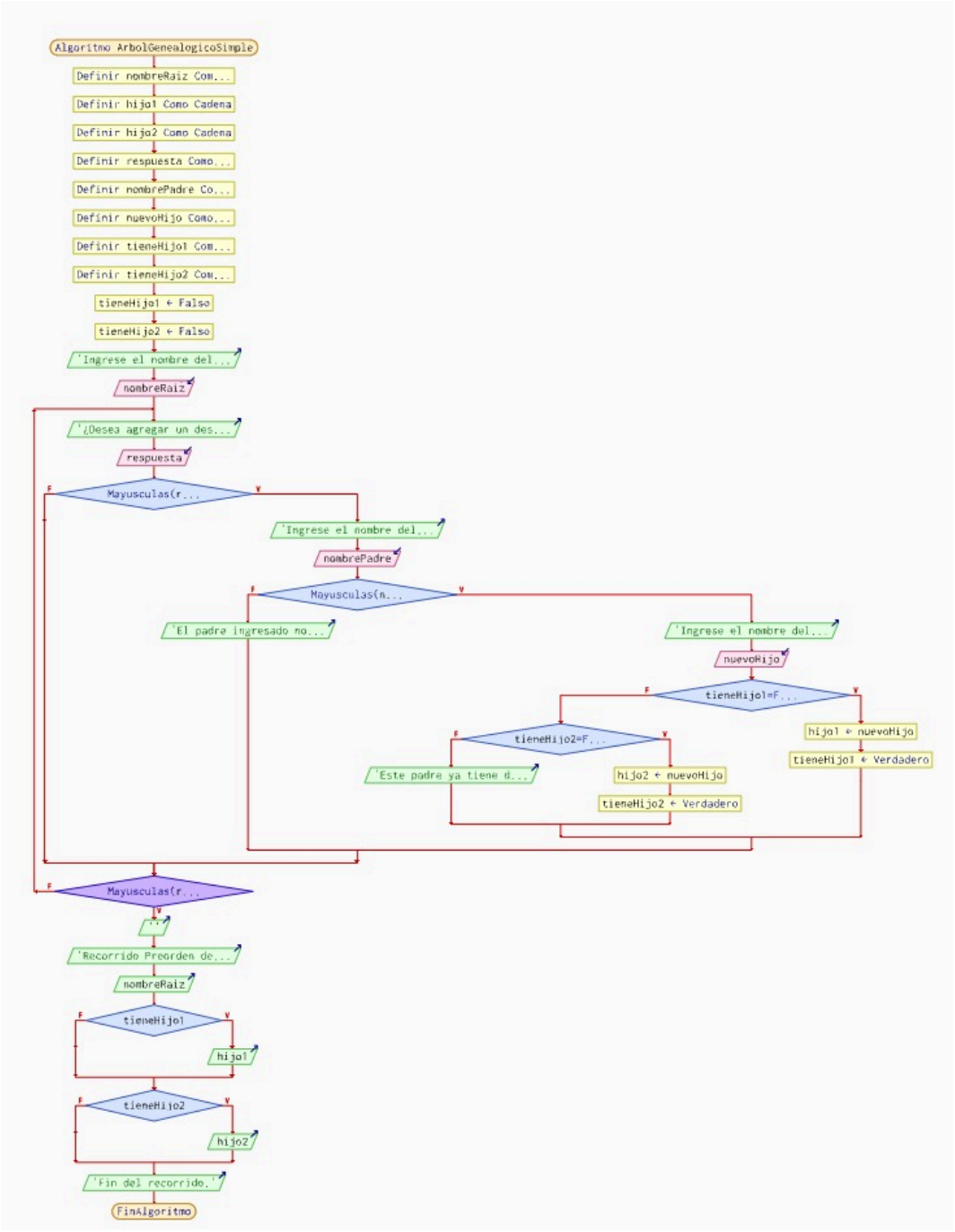
2.2 Algoritmos principales:

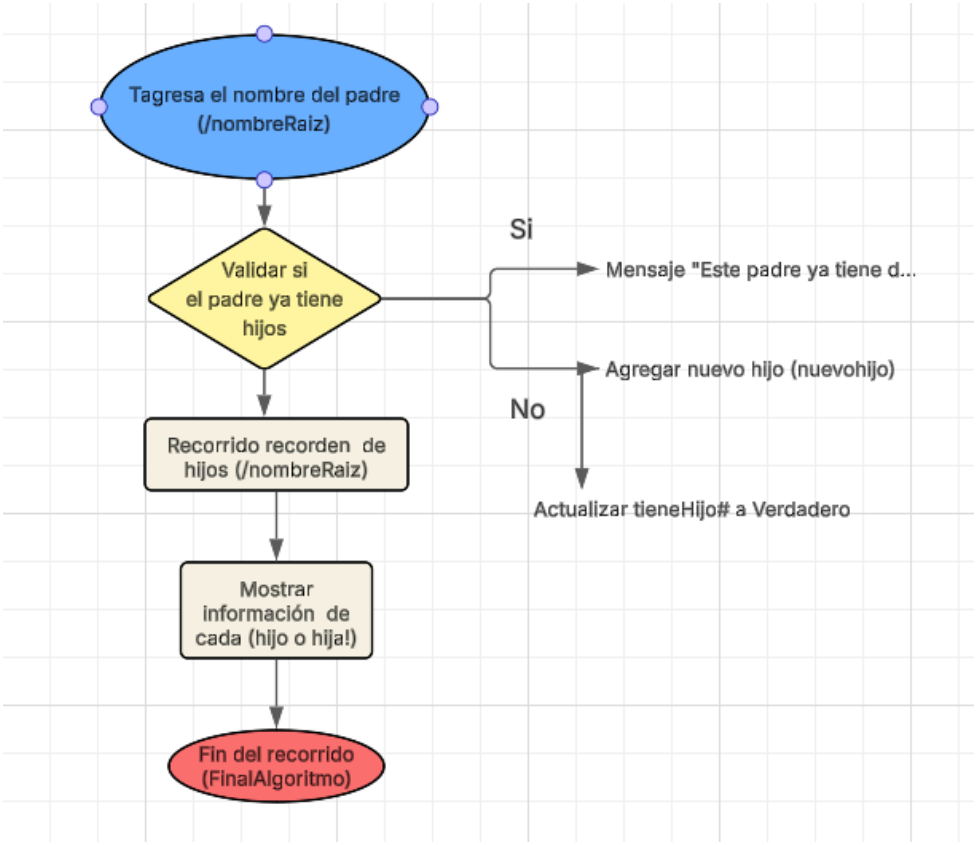
- ***Pseudocódigo para crear un árbol binario.***

```
Algoritmo ArbolGenealogicoSimple

    Definir nombreRaiz Como Cadena
    Definir hijo1 Como Cadena
    Definir hijo2 Como Cadena
    Definir respuesta Como Cadena
    Definir nombrePadre Como Cadena
    Definir nuevoHijo Como Cadena
    Definir tieneHijo1 Como Logico
    Definir tieneHijo2 Como Logico

    tieneHijo1 ← Falso
    tieneHijo2 ← Falso

    Escribir "Ingrese el nombre del ancestro raiz:"
    Leer nombreRaiz

    Repetir
        Escribir "¿Desea agregar un descendiente? (SI/NO):"
        Leer respuesta

        Si Mayusculas(respuesta) = "SI" Entonces
            Escribir "Ingrese el nombre del padre:"
            Leer nombrePadre

            Si Mayusculas(nombrePadre) = Mayusculas(nombreRaiz) Entonces
                Escribir "Ingrese el nombre del nuevo hijo:"
                Leer nuevoHijo

                Si tieneHijo1 = Falso Entonces
                    hijo1 ← nuevoHijo
                    tieneHijo1 ← Verdadero
                Sino
                    Si tieneHijo2 = Falso Entonces
                        hijo2 ← nuevoHijo
                        tieneHijo2 ← Verdadero
                    Sino
                        Escribir "Este padre ya tiene dos hijos."
                    FinSi
                FinSi
            Sino
                Escribir "El padre ingresado no es válido (solo se permite el ancestro raiz en esta versión)."
            FinSi
        FinSi
```

```
    Hasta Que Mayusculas(respuesta) ≠ "SI"

    Escribir ""
    Escribir "Recorrido Preorden del árbol:"
    Escribir nombreRaiz

    Si tieneHijo1 Entonces
        Escribir hijo1
    FinSi

    Si tieneHijo2 Entonces
        Escribir hijo2
    FinSi

    Escribir "Fin del recorrido."

FinAlgoritmo

```

2.3 Diagramas de Flujo:





Componentes del Diagrama

1. Inicio del Proceso
   - "Ingresa el nombre del padre (/nombreRaiz)"
2. Validación Principal
   - "Validar si el padre ya tiene hijos"
3. Rama Afirmativa (Padre con hijos)
   - “Mensaje 'Este padre ya tiene d...”
4. Rama Negativa (Padre sin hijos)
   - "Agregar nuevo hijo (nuevohijo)"
   - "Actualizar tieneHijo# a Verdadero"
5. Recorrido de Hijos
   - "Recorrido recorden de hijos (/nombreRaiz)"
   - "Mostrar información de cada (¡hijo o hija!)"
6. Finalización
   - "Fin del recorrido (FinalAlgoritmo)"

2.2 Avance del código fuente:

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

struct Nodo {
    string nombre;
    Nodo* izquierda;
    Nodo* derecha;
};

// Crear nuevo nodo
Nodo* crearNodo(string nombre) {
    Nodo* nuevo = new Nodo();
    nuevo->nombre = nombre;
    nuevo->izquierda = nuevo->derecha = NULL;
    return nuevo;
}

// Insertar nodo en ABB
Nodo* insertar(Nodo* raiz, string nombre) {
    if (raiz == NULL) return crearNodo(nombre);
    if (nombre < raiz->nombre)
        raiz->izquierda = insertar(raiz->izquierda, nombre);
    else
        raiz->derecha = insertar(raiz->derecha, nombre);
    return raiz;
}

// Recorrido inorden
void inorden(Nodo* raiz) {
    if (raiz != NULL) {
        inorden(raiz->izquierda);
        cout << raiz->nombre << " ";
        inorden(raiz->derecha);
    }
}

// Buscar miembro
bool pertenece(Nodo* raiz, string nombre) {
    if (raiz == NULL) return false;
    if (raiz->nombre == nombre) return true;
    if (nombre < raiz->nombre)
        return pertenece(raiz->izquierda, nombre);
    else
        return pertenece(raiz->derecha, nombre);
}

int main() {
    Nodo* raiz = NULL;
    raiz = insertar(raiz, "Atahualpa");
    raiz = insertar(raiz, "Huayna Capac");
    raiz = insertar(raiz, "Tupac Yupanqui");

    cout << "Recorrido Inorden: ";
    inorden(raiz);
```

```cpp
    inorden(raiz);
    cout << endl;

    string buscar = "Huayna Capac";
    if (pertenece(raiz, buscar))
        cout << buscar << " pertenece al árbol." << endl;
    else
        cout << buscar << " NO pertenece al árbol." << endl

    return 0;
}
```

Link del Github:

https://github.com/zahid09190/Arbol_Genealogico.git