

Grupo 4 - ABR

# ÁRBOL GENEALÓGICO

**Docente: Cohaila Bravo Ninoska Nataly**

## **Integrantes:**

- Lazo Gómez Sofía Adara
- Quispe Yauri Henry Bryan
- Velasquez Carhuancho Angel

# Objetivos

- Representar la genealogía de una familia simulada mediante un Árbol Binario de Búsqueda.
- Ordenar los miembros según su año de nacimiento.
- Aplicar operaciones básicas del ABB: insertar, buscar, eliminar y recorrer.





# ¿Qué es un Árbol Binario de Búsqueda?

- Estructura donde cada nodo tiene máximo dos hijos.
- Los menores van a la izquierda.
- Los mayores van a la derecha.
- Permite búsqueda e inserción eficiente.

# Menu

- 1. Insertar familiar** → valida año y nombre, evita duplicados, convierte a mayúsculas.
- 2. Buscar familiar** → busca por año y muestra el resultado.
- 3. Eliminar familiar** → elimina un nodo por año.
- 4. Mostrar Inorden** → imprime árbol ordenado por año.
- 5. Mostrar Preorden** → imprime árbol en preorden.
- 6. Mostrar Postorden** → imprime árbol en postorden.
- 7. Salir** → termina el programa.

# 1. Insertar familiar

```
case 1: {
    int anio;
    string nombre;
    cout << "\nIngrese el año de nacimiento: ";
    cin >> anio;
    if (cin.fail() || anio <= 0) {
        cout << "Error: año inválido.\n";
        cin.clear();
        cin.ignore(1000, '\n');
        break;
    }
    cin.ignore();
    cout << "Ingrese el nombre completo: ";
    getline(cin, nombre);
    if (nombre == "") {
        cout << "Error: el nombre no puede estar vacío.\n";
        break;
    }
    convertirMayusculas(nombre);
    if (buscar(arbol, anio) != NULL) {
        cout << "Error: ya existe un familiar con ese año.\n";
        break;
    }
    insertado = false;
    arbol = insertar(arbol, anio, nombre, insertado);
    if (insertado)
        cout << "Familiar insertado correctamente.\n";
    else
        cout << "Error al insertar.\n";
    break;
}
```

validación del año: que sea número y mayor a 0

validación del nombre: que no esté vacío

validación del año: no se puede repetir el año

llamado a insertar()

# insertar

```
// Insertar en ABB ordenado por año (RECHAZAR DUPLICADOS)
Nodo* insertar(Nodo* arbol, int anio, string nombre, bool &insertado) {
    if (arbol == NULL) {
        insertado = true;
        return crearNodo(anio, nombre);
    }
    if (anio < arbol->anio) {
        arbol->izq = insertar(arbol->izq, anio, nombre, insertado);
    }
    else if (anio > arbol->anio) {
        arbol->der = insertar(arbol->der, anio, nombre, insertado);
    }
    else {
        insertado = false; // año duplicado
    }
    return arbol;
}
```

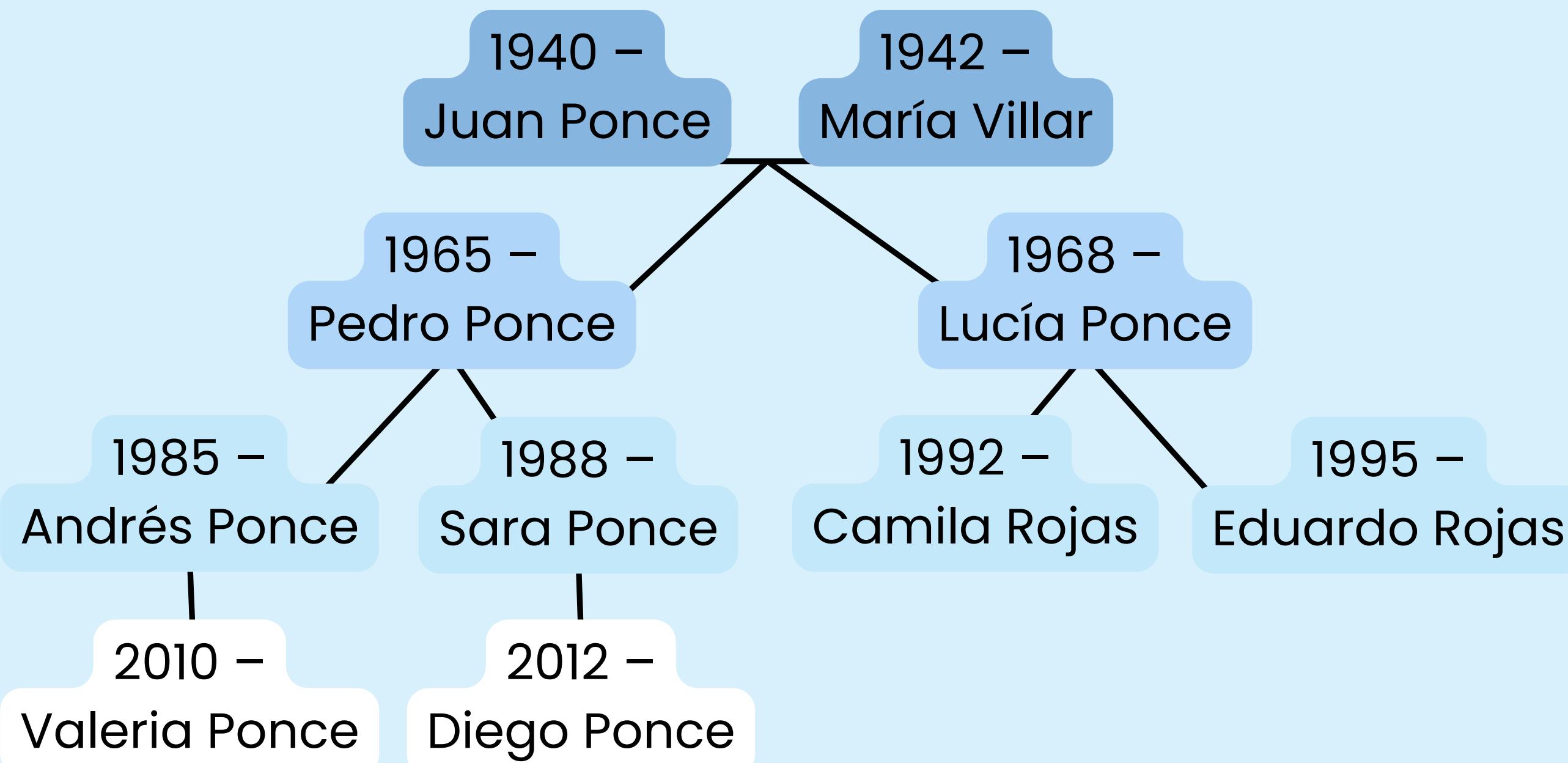
si ya hay nodo se compara los años

si no hay nodo llama a crearNodo

si el año a insertar es menor al del nodo el puntero arbol(del nodo) pasa al espacio izquierdo e insertar se vuelve a iniciar

si el año a insertar es mayor al del nodo el puntero arbol(del nodo) pasa al espacio derecho e insertar se vuelve a iniciar

# Arbol Familiar



# ABB - por orden de inserción

1968 – Lucía Ponce

1942 – María Villar

1995 – Eduardo Rojas

1940 – Juan Ponce

1965 – Pedro Ponce

1985 – Andrés Ponce

1988 – Sara Ponce

1992 – Camila Rojas

2010 – Valeria Ponce

2012 – Diego Ponce

1968 –  
Lucía Ponce

1942 –  
María Villar

1940 –  
Juan Ponce

1965 –  
Pedro Ponce

1995 –  
Eduardo Rojas

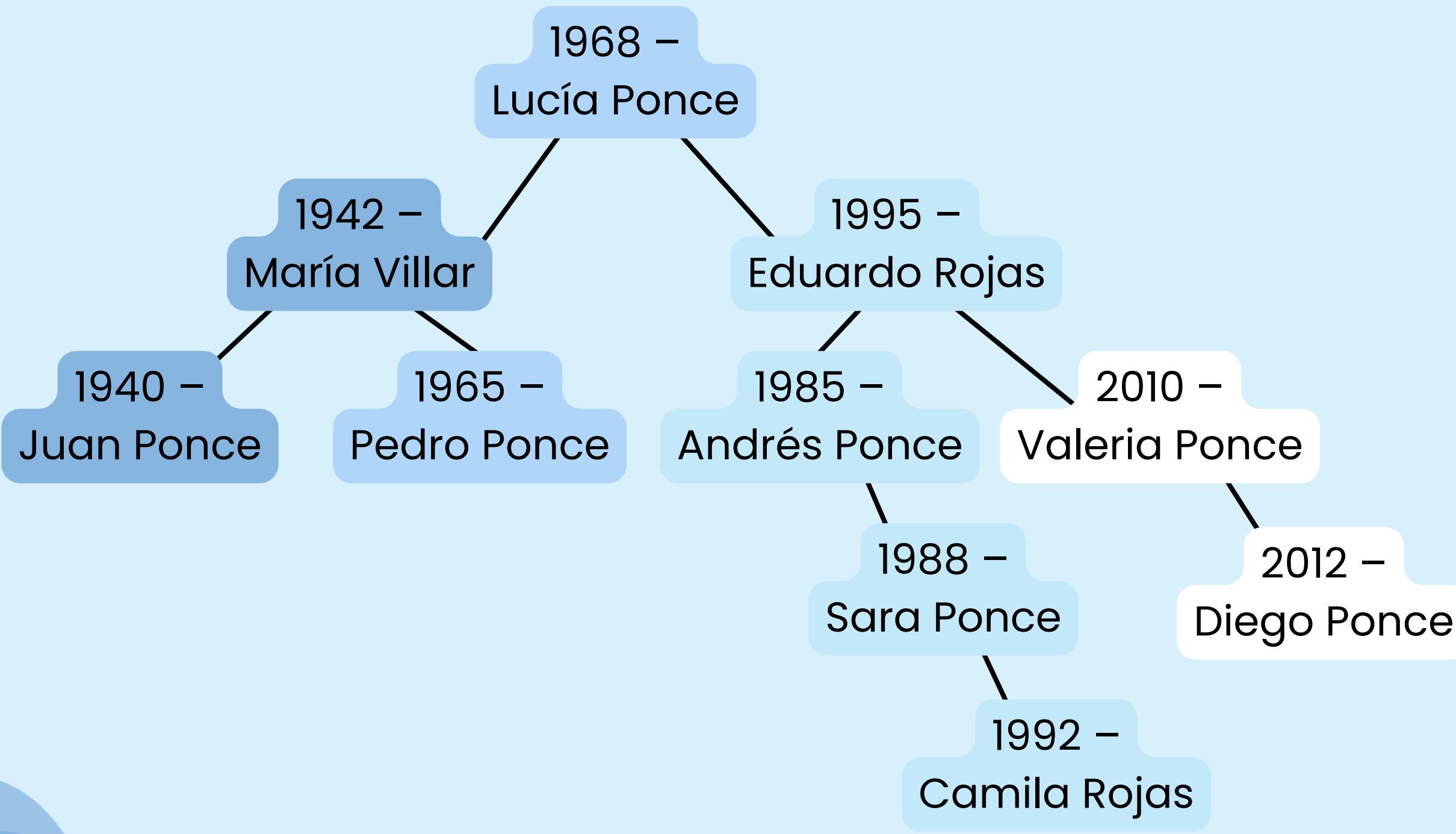
1985 –  
Andrés Ponce

1988 –  
Sara Ponce

2010 –  
Valeria Ponce

2012 –  
Diego Ponce

1992 –  
Camila Rojas



## 2. Buscar familiar

```
case 2: {
    if (arbol == NULL) {
        cout << "\nÁrbol vacío.\n";
        break;
    }

    int anio;
    cout << "\nIngrese el año a buscar: ";
    cin >> anio;

    if (anio <= 0) {
        cout << "Error: año inválido.\n";
        break;
    }

    Nodo* encontrado = buscar(arbol, anio);
    if (encontrado != NULL)
        cout << "Encontrado: " << encontrado->anio << " - " << encontrado->nombre << "\n";
    else
        cout << "No existe un familiar con ese año.\n";
    break;
}
```

validación del año: que sea mayor a 0

llamado a buscar()

# buscar

```
// Buscar por año
Nodo* buscar(Nodo* arbol, int anio) {
    if (arbol == NULL) return NULL;
    if (anio == arbol->anio) return arbol;
    if (anio < arbol->anio) return buscar(arbol->izq, anio);
    return buscar(arbol->der, anio);
}
```

si no hay nodo (vacío) retorna nulo

si el año ingresado es igual al del nodo, retorna la ubicación del nodo

los años no coinciden, se comparan y se vuelve a buscar

# 3. Eliminar familiar

```
case 3: {
    if (arbol == NULL) {
        cout << "\nÁrbol vacío.\n";
        break;
    }

    int anio;
    cout << "\n";
    inorden(arbol);
    cout << endl;
    cout << "Ingrese el año del integrante a eliminar: ";
    cin >> anio;
    if (anio <= 0) {
        cout << "Error: año inválido.\n";
        break;
    }

    bool eliminado = false;
    arbol = eliminarNodo(arbol, anio, eliminado); ← llamado a eliminarNodo()

    if (eliminado)
        cout << "Familiar eliminado correctamente.\n";
    else
        cout << "No se encontró ese año.\n";

    break;
}
```

validación del arbol: que no esté vacío

validación del año: que sea mayor a 0

llamado a eliminarNodo()

# eliminarNodo

```
// Eliminar por año
Nodo* eliminarNodo(Nodo* arbol, int anio, bool &eliminado) {
    if (arbol == NULL) {
        eliminado = false;
        return NULL;
    }
    if (anio < arbol->anio) {
        arbol->izq = eliminarNodo(arbol->izq, anio, eliminado);
    }
    else if (anio > arbol->anio) {
        arbol->der = eliminarNodo(arbol->der, anio, eliminado);
    }
    else {
        eliminado = true;
        // Caso 1: sin hijos
        if (arbol->izq == NULL && arbol->der == NULL) {
            delete arbol;
            return NULL;
        }
        // Caso 2: un hijo
        if (arbol->izq == NULL) {
            Nodo* temp = arbol->der;
            delete arbol;
            return temp;
        }
        if (arbol->der == NULL) {
            Nodo* temp = arbol->izq;
            delete arbol;
            return temp;
        }
        // Caso 3: dos hijos
        Nodo* suc = minimo(arbol->der);
        arbol->anio = suc->anio;
        arbol->nombre = suc->nombre;
        arbol->der = eliminarNodo(arbol->der, suc->anio, eliminado);
    }
    return arbol;
}
```

caso 1

caso 2

caso 3

si no hay nodo (vacío) retorna nulo

si el año ingresado es menor baja a la izquierda

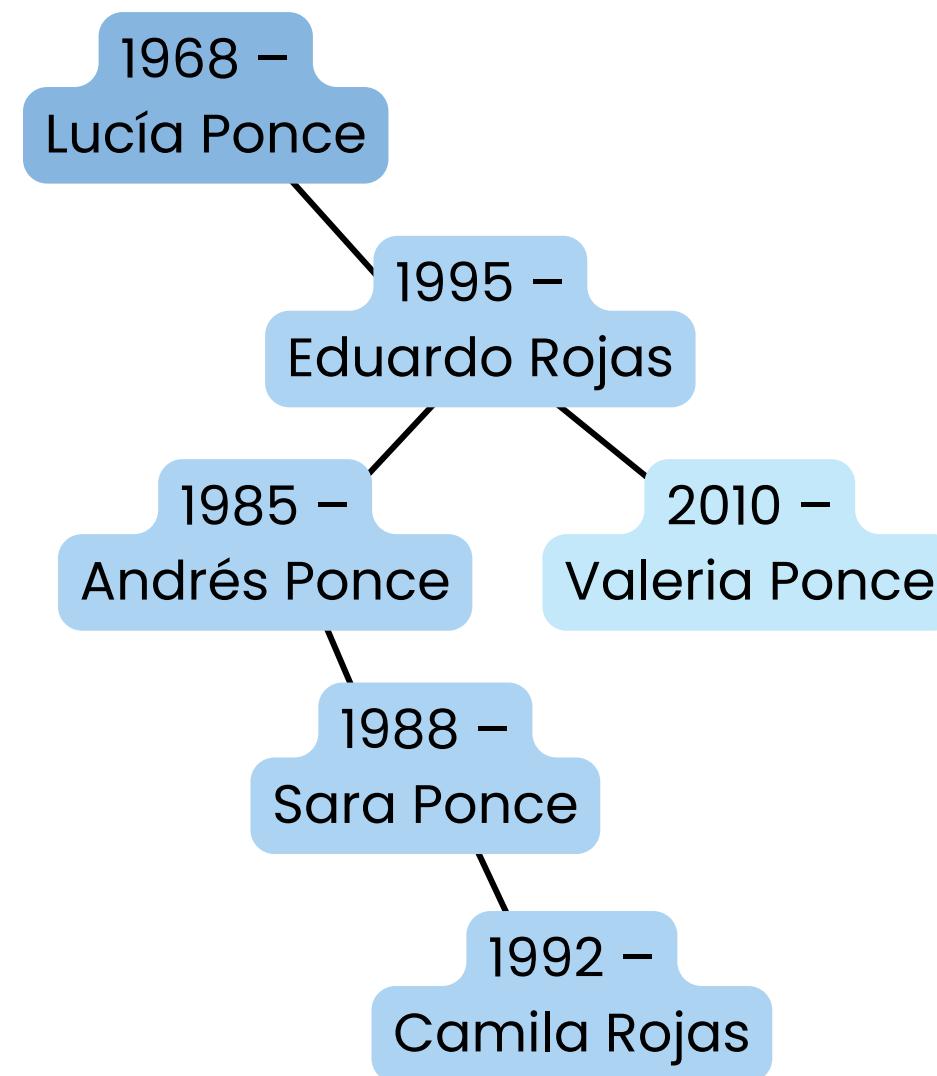
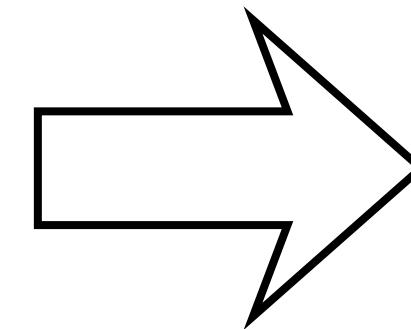
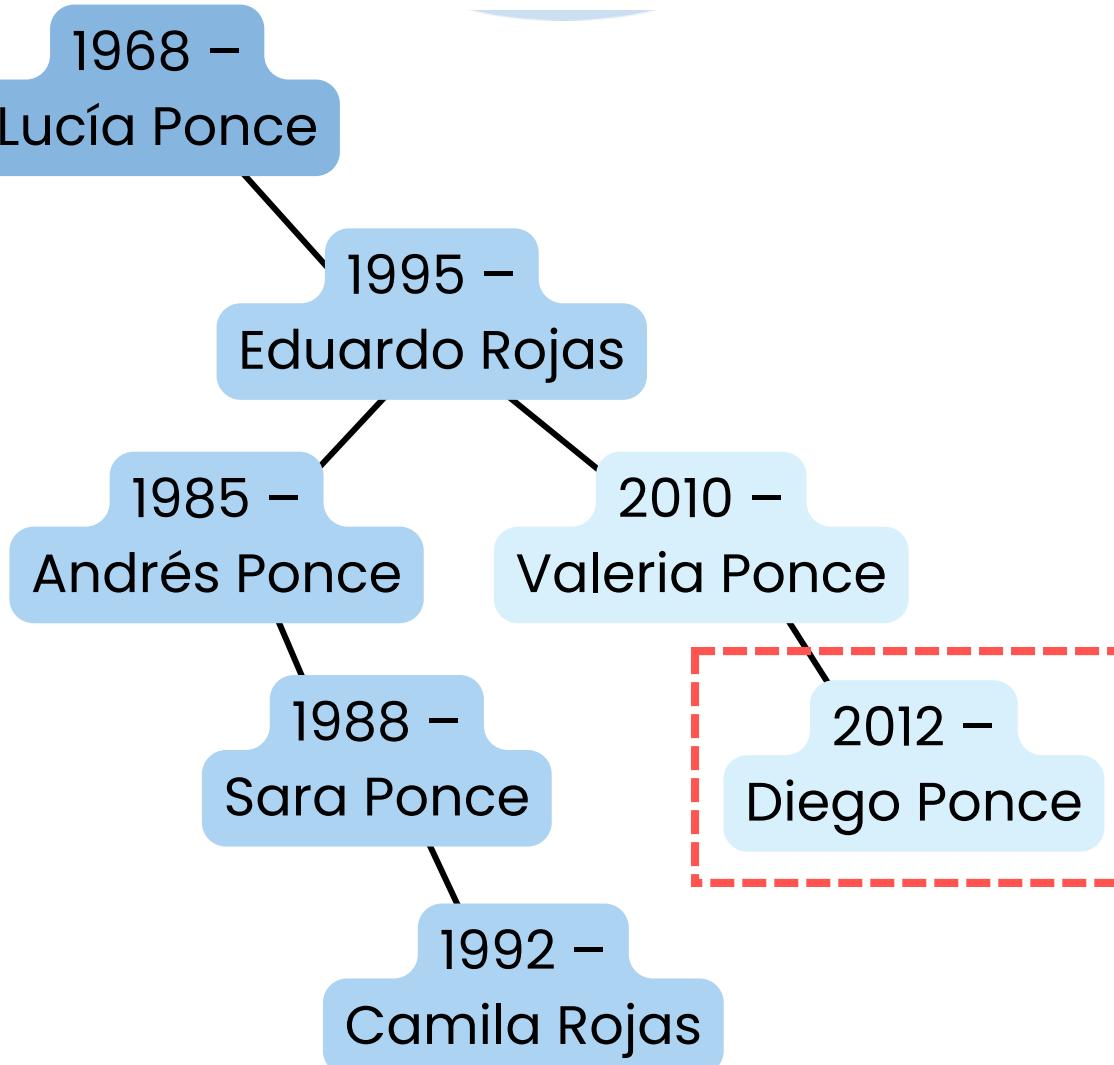
si es mayor baja a la derecha

si es igual, es el nodo a eliminar

Los punteros arbol->izq y arbol->der se actualizan en cada retorno.

# caso 1: sin hijos

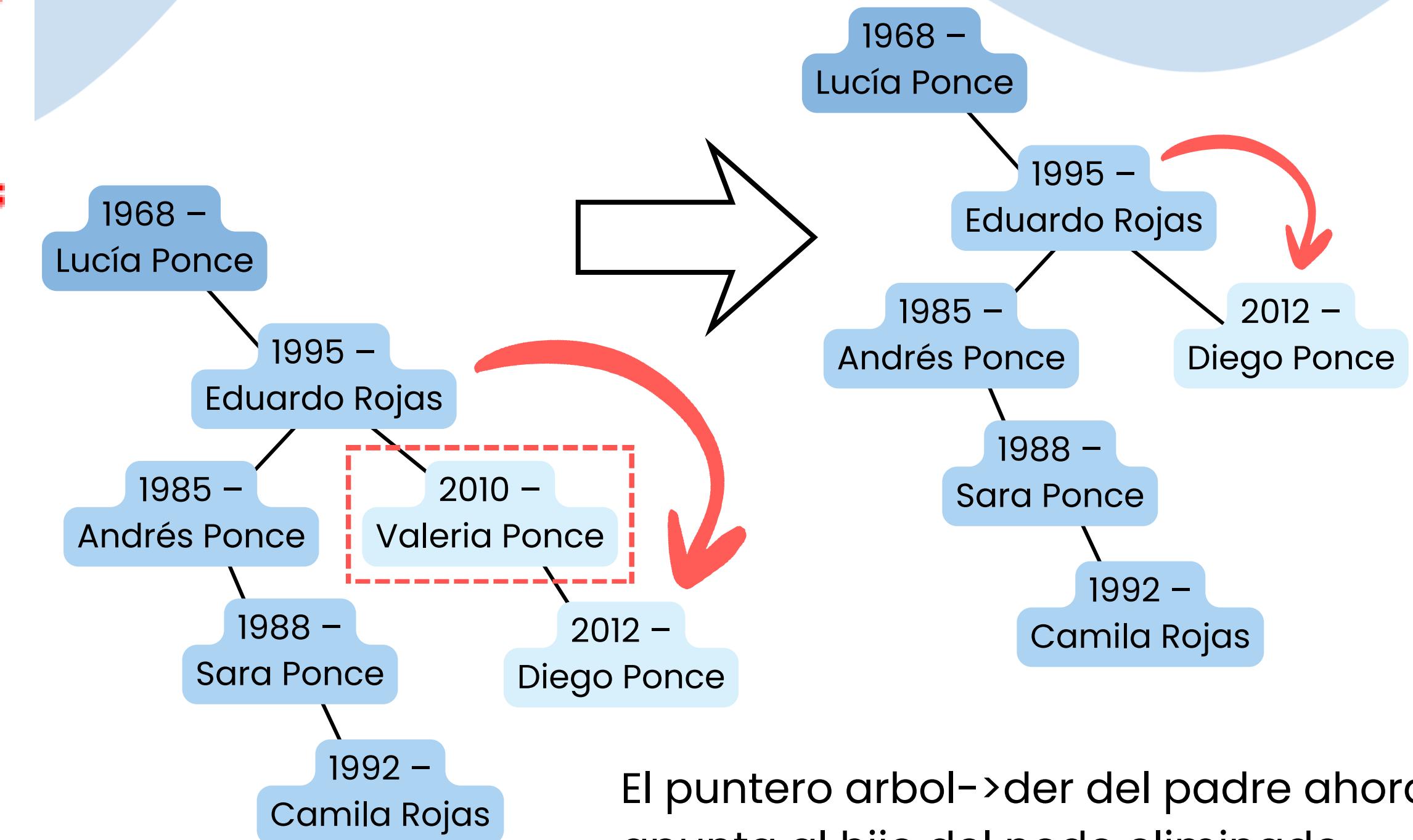
```
// Caso 1: sin hijos
if (arbol->izq == NULL && arbol->der == NULL) {
    delete arbol;
    return NULL;
}
```



# caso 2: con 1 hijo

```
// Caso 2: un hijo
if (arbol->izq == NULL) {
    Nodo* temp = arbol->der;
    delete arbol;
    return temp;
}

if (arbol->der == NULL) {
    Nodo* temp = arbol->izq;
    delete arbol;
    return temp;
}
```



El puntero `arbol->der` del padre ahora apunta al hijo del nodo eliminado.

# caso 3: con 2 hijos

// Caso 3: dos hijos

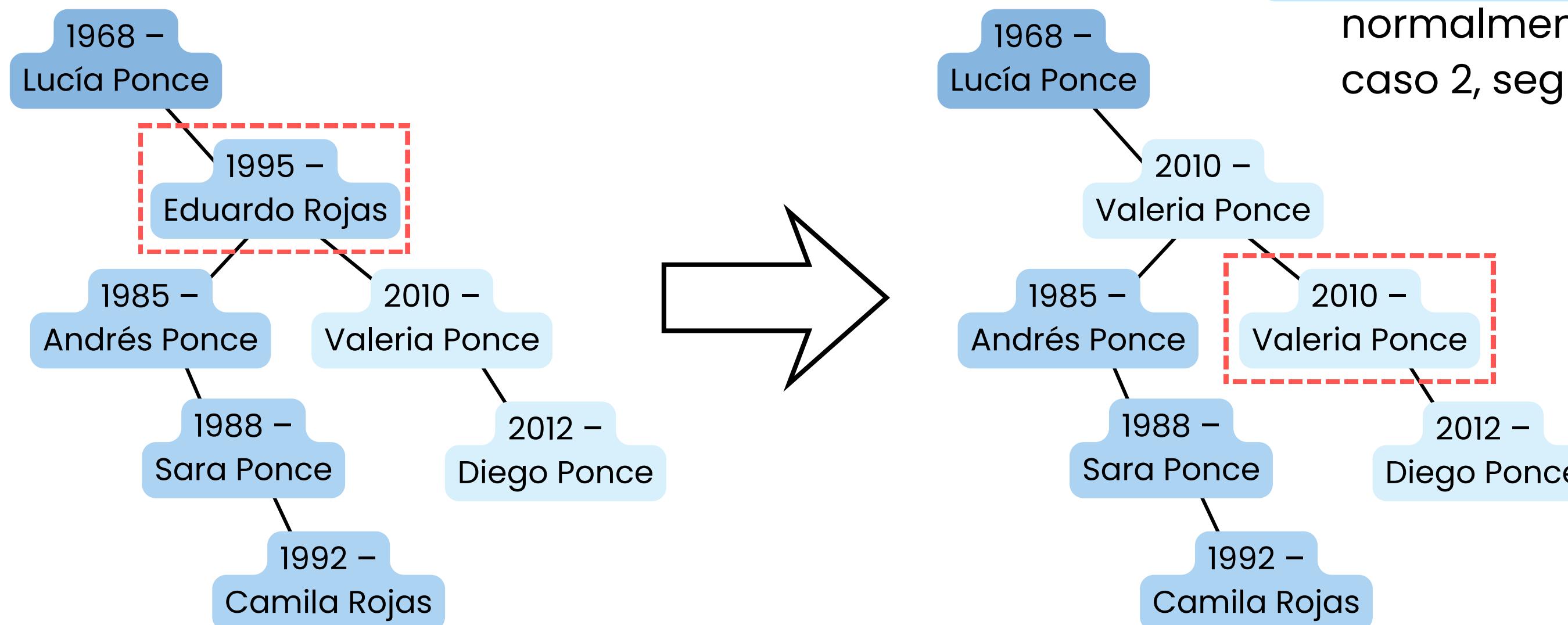
```
Nodo* suc = minimo(arbol->der);
arbol->anio = suc->anio;
arbol->nombre = suc->nombre;
arbol->der = eliminarNodo(arbol->der, suc->anio, eliminado);
```

encuentra el sucesor inorden

copiar los datos del sucesor  
al nodo a eliminar

eliminar el sucesor en el  
subárbol derecho

normalmente es caso 1 o  
caso 2, según su estructura.



# 4. Mostrar Inorden

izquierda - raiz - derecha

```
// Recorridos
void inorden(Nodo* arbol) {
    if (arbol == NULL) return;
    inorden(arbol->izq);
    cout << arbol->anio << " - " << arbol->nombre << "\n";
    inorden(arbol->der);
}
```

primero visita todo el subárbol izquierdo

luego imprime el nodo actual

luego visita todo el subárbol derecho

INORDEN:

```
1940 - JUAN PONCE
1942 - MARIA VILLAR
1965 - PEDRO PONCE
1968 - LUCIA PONCE
1985 - ANDRES PONCE
1988 - SARA PONCE
1992 - CAMILA ROJAS
1995 - EDUARDO ROJAS
2010 - VALERIA PONCE
2012 - DIEGO PONCE
```

En un ABB, el recorrido inorden SIEMPRE imprime los datos ordenados de menor a mayor.  
Imprime los familiares ordenados por AÑO.

# 5. Mostrar Preorden

raiz - izquierda - derecha

```
void preorden(Nodo* arbol) {  
    if (arbol == NULL) return;  
    cout << arbol->anio << " - " << arbol->nombre << "\n";  
    preorden(arbol->izq);  
    preorden(arbol->der);  
}
```

imprime primero el nodo actual

luego recorre su subárbol izquierdo

finalmente recorre su subárbol derecho

PREORDEN:  
1968 - LUCIA PONCE  
1942 - MARIA VILLAR  
1940 - JUAN PONCE  
1965 - PEDRO PONCE  
1995 - EDUARDO ROJAS  
1985 - ANDRES PONCE  
1988 - SARA PONCE  
1992 - CAMILA ROJAS  
2010 - VALERIA PONCE  
2012 - DIEGO PONCE

Preorden es el recorrido que “cuenta la historia del árbol desde la raíz”. Es útil cuando queremos mostrar la estructura del árbol comenzando desde la raíz

# 6. Mostrar Postorden

izquierda - derecha - raiz

```
void postorden(Nodo* arbol) {  
    if (arbol == NULL) return;  
    postorden(arbol->izq);  
    postorden(arbol->der);  
    cout << arbol->anio << " - " << arbol->nombre << "\n";  
}
```

primero visita todo el subárbol izquierdo

luego recorre su subárbol izquierdo

finalmente imprime el nodo actual

POSTORDEN:

1940	-	JUAN PONCE
1965	-	PEDRO PONCE
1942	-	MARIA VILLAR
1992	-	CAMILA ROJAS
1988	-	SARA PONCE
1985	-	ANDRES PONCE
2012	-	DIEGO PONCE
2010	-	VALERIA PONCE
1995	-	EDUARDO ROJAS
1968	-	LUCIA PONCE

Este recorrido asegura que los hijos se procesen antes que sus padres. Es útil, por ejemplo, para liberar memoria o borrar nodos.

**Muchas  
Gracias**