ETAPA 3 — PROTOTIPO

Curso: Estructura de Datos

Docente: Yesenia Concha Ramos

Integrantes:

- Jhon Gustavo Ccarita Velasquez
- Rodrigo Sevillanos Tinco
- Andre Sebastian Espinoza Zea

Tema: ABR - prototipado

NRC: 59008

2025-02

Cusco - Perú

CAPÍTULO 1 - FASE DE IDEACIÓN

1. Descripción general del proyecto

Este proyecto propone el diseño e implementación de un Árbol Genealógico Mitológico, cuyo propósito es modelar la compleja jerarquía de dioses, titanes, héroes y criaturas de la mitología griega.
Para representar esta estructura se emplea un Árbol Binario de Búsqueda (ABB), tomando como clave principal el año simbólico de nacimiento de cada entidad mitológica.

El ABB permite:

- organizar la genealogía en un orden temporal coherente,
- consultar relaciones de antigüedad y descendencia,
- recorrer la estructura bajo distintos enfoques jerárquicos,
- realizar operaciones de búsqueda en tiempo eficiente,
- mantener el orden interno sin necesidad de estructuras externas.

Cada nodo representa a un ser mitológico y contiene su nombre, su rol jerárquico y su año simbólico, lo que permite reconstruir relaciones como:

- linajes divinos (Titanes → Olímpicos),
- descendencias célebres (Zeus → Atenea → héroes),
- etapas mitológicas (primigenios → olímpicos → héroes).

2. Requerimientos del sistema

Requerimientos funcionales (RF)

- RF1: Insertar dioses, titanes, héroes o criaturas en el árbol.
- RF2: Buscar miembros por año simbólico.
- RF3: Eliminar nodos conservando el orden del ABB.
- RF4: Mostrar recorridos genealógicos:
    - Inorden (línea temporal)
    - Preorden (jerarquía divina)
    - Postorden (generaciones)
- RF5: Representar líneas familiares dentro del árbol.

Requerimientos no funcionales (RNF)

- RNF1: Interfaz sencilla y compatible con evaluación académica.
- RNF2: Operaciones con eficiencia promedio O(log n).

- RNF3: Implementación en C++ bajo entorno Dev-C++.
- RNF4: Manejo robusto de entradas incorrectas.

3. Respuestas a preguntas guía

1. Información almacenada en cada nodo

Cada nodo del ABB representa un personaje mitológico.
Debe incluir:

- nombre (ej. Zeus, Hades, Cronos)
- rol mitológico (Titán, Olímpico, Semidiós, Criatura)
- año simbólico de nacimiento → clave del ABB
- puntero izquierdo → entidades más antiguas
- puntero derecho → entidades más recientes

  Esto permite mantener un árbol ordenado temporalmente y apto para consultas eficientes.

2. Inserción y eliminación sin romper el ABB

Inserción

Se compara el año nuevo con el nodo actual:

- menor → va al subárbol izquierdo
- mayor → va al subárbol derecho
- igual → se rechaza para evitar duplicados

El algoritmo es recursivo y garantiza la validez del ABB.

Eliminación

Se usa el método estándar del ABB:

1. Sin hijos: se elimina directamente
2. Un hijo: el hijo reemplaza al nodo
3. Dos hijos: se reemplaza con el sucesor inorden

(nodo más pequeño del subárbol derecho)

3. Métodos de recorrido

Tres recorridos clásicos permiten visualizar diferentes aspectos:

- Inorden (izq – raíz – der):
  - Lista cronológica de los seres mitológicos
- Preorden (raíz – izq – der):

    - Muestra jerarquía divina
- Postorden (izq – der – raíz):
    - Permite analizar generaciones completas

Cada uno responde a un tipo de consulta distinta.

4. Determinar si pertenece a una rama

Para saber si alguien pertenece a una línea ancestral:

1. buscar al ancestro (ej. Cronos, Zeus),
2. desde su nodo, recorrer preorden/inorden,
3. comprobar si aparece el miembro buscado.

También se pueden establecer ramas por rol:

- Titanes → primigenios
- Olímpicos → generación central
- Semidioses → descendientes
- Criaturas → derivados alternos

Si el miembro está dentro del subárbol de esa categoría, pertenece a la rama.

5. Balancear el árbol

Si se insertan años muy dispersos, el ABB puede volverse una lista.
Para evitarlo:

Métodos aplicables:
Reconstrucción balanceada (RECOMENDADO)

1. obtener nodos en inorden,
2. reconstruir tomando elemento medio como raíz,
3. subárboles con elementos anteriores/siguientes.

Complejidad: O(n).

Rotaciones (AVL)

- izquierda
- derecha
- doble

CAPÍTULO 2 – PROTOTIPO

1. DESCRIPCIÓN DE LA ESTRUCTURA DE DATOS Y OPERACIONES

El prototipo usa un Árbol Binario de Búsqueda (ABB) para representar la genealogía de la mitología griega.
Cada nodo contiene información de un dios, titán, héroe u otro personaje, y queda ubicado en el árbol según una clave numérica única que deriva del año simbólico de nacimiento y el mes. Esta clave asegura que no haya duplicados exactos y mantiene el orden entre los nodos.

Los árboles de búsqueda binarios se construyen de forma que, para cada nodo:

- los valores menores se ubican en el subárbol izquierdo,
- los mayores en el subárbol derecho.

Esto permite operaciones eficientes —inserción, búsqueda y eliminación— siempre que el árbol se mantenga relativamente balanceado.

1.1 Estructura de cada nodo

Cada nodo almacena:

- int anio y int mes — datos cronológicos del personaje.
- int clave — combinación anio*100 + mes, que funciona como clave única para ordenar.
- string nombre — nombre mitológico.
- string rol — categoría o función, por ejemplo Titán, Olímpico, Héroe.
- Nodo* izq — puntero al subárbol con claves menores.
- Nodo* der — puntero al subárbol con claves mayores.

Este diseño permite que el árbol refleje un orden temporal y jerárquico, y que cada consulta o modificación sea guiada por la comparación de claves.

1.2 Operaciones implementadas

La siguiente tabla resume las operaciones fundamentales que implementa el prototipo del Árbol Binario de Búsqueda (ABB), explicando su propósito, comportamiento interno y relevancia dentro del sistema de genealogía mitológica:

Insertar
*Inserta un nuevo nodo en el ABB utilizando como clave compuesta anio*100 + mes. La función compara la clave del nuevo elemento con los nodos existentes y lo ubica recursivamente en el subárbol izquierdo (si es menor) o en el derecho (si es mayor). Si la clave ya existe, el algoritmo detiene la operación para preservar la propiedad fundamental del ABB (no duplicados).*

Buscar

Realiza una búsqueda exacta basada en la clave año+mes. Se emplea un recorrido recursivo que desciende por izquierda o derecha según la comparación con el nodo actual. La búsqueda concluye en tiempo proporcional a la altura del árbol, retornando el nodo encontrado o NULL.

Eliminar

Elimina un nodo del ABB manteniendo su estructura correcta. La función identifica tres situaciones: nodo hoja (se elimina directamente), nodo con un solo hijo (el hijo reemplaza al nodo), y nodo con dos hijos (se reemplaza por el sucesor inorden obtenido con la función mínima). Este procedimiento garantiza que la propiedad de ordenamiento del ABB se conserve tras la eliminación.

Recorridos

Implementa los recorridos clásicos del ABB: Inorden (Izq–Raíz–Der), que produce una secuencia cronológica ascendente; Preorden (Raíz–Izq–Der), que destaca jerarquía y estructura; y Postorden (Izq–Der–Raíz), empleado para visualizar descendencia completa. Cada recorrido responde a un patrón específico de análisis de la genealogía mitológica.
Visualización ASCII del árbol

Genera una representación estructural del ABB mediante caracteres ASCII. Esta visualización ubica la raíz en la parte superior y despliega ramas, nodos internos y hojas con indentación gradual. Permite inspeccionar la forma del árbol después de cada inserción, eliminación o búsqueda, facilitando su interpretación sin herramientas gráficas.
Validación de entradas

2. ALGORITMOS PRINCIPALES (PSEUDOCÓDIGO)

2.1 Pseudocódigo para crear un Árbol Binario de Búsqueda (Inserción)

```
PROCEDIMIENTO Insertar(raiz, anio, mes, nombre, rol)
    clave ← anio * 100 + mes        // Se genera clave compuesta única

    SI raiz ES NULL ENTONCES
        raiz ← NUEVO Nodo(anio, mes, nombre, rol)
        RETORNAR raiz
    FIN SI

    SI clave < raiz.clave ENTONCES
        raiz.izq ← Insertar(raiz.izq, anio, mes, nombre, rol)
    SINO SI clave > raiz.clave ENTONCES
        raiz.der ← Insertar(raiz.der, anio, mes, nombre, rol)
    SINO
        MOSTRAR "Clave duplicada: no se inserta"
    FIN SI

    RETORNAR raiz
FIN PROCEDIMIENTO
```

El proceso de inserción crea y posiciona un nuevo nodo dentro del Árbol Binario de Búsqueda comparando su clave con los nodos existentes. El algoritmo recorre el árbol de forma recursiva hasta encontrar la posición correcta, asegurando que los valores menores se ubiquen en el subárbol izquierdo y los mayores en el subárbol derecho. En caso de que la clave ya exista, la inserción se detiene para evitar duplicados y mantener la estructura válida del ABB.

2.2 Pseudocódigo para realizar recorridos del árbol

```
PROCEDIMIENTO Inorden(nodo)
    SI nodo ES NULL ENTONCES
        RETORNAR
    FIN SI

    Inorden(nodo.izq)                // Visitar subárbol izquierdo
    MOSTRAR nodo                     // Procesar nodo actual
    Inorden(nodo.der)                // Visitar subárbol derecho
FIN PROCEDIMIENTO

PROCEDIMIENTO Preorden(nodo)
    SI nodo ES NULL ENTONCES
        RETORNAR
    FIN SI

    MOSTRAR nodo                     // Procesar nodo primero
    Preorden(nodo.izq)               // Subárbol izquierdo
    Preorden(nodo.der)               // Subárbol derecho
FIN PROCEDIMIENTO


PROCEDIMIENTO Postorden(nodo)
    SI nodo ES NULL ENTONCES
        RETORNAR
    FIN SI

    Postorden(nodo.izq)              // Subárbol izquierdo
    Postorden(nodo.der)              // Subárbol derecho
    MOSTRAR nodo                     // Procesar al final
FIN PROCEDIMIENTO
```

Los recorridos del árbol permiten analizar la estructura desde diferentes enfoques. El recorrido Inorden lista los nodos en orden ascendente según la clave; Preorden muestra primero la raíz y luego sus descendientes, ideal para analizar jerarquías; mientras que Postorden explora primero los hijos y finaliza con el nodo actual, permitiendo interpretar generaciones completas. Los tres recorridos se implementan mediante procedimientos recursivos simples y directos.

Recorrido INORDEN (Izquierda → Raíz → Derecha)

El recorrido Inorden visita primero el subárbol izquierdo, luego procesa el nodo actual y finalmente recorre el subárbol derecho.
En un Árbol Binario de Búsqueda, este recorrido siempre produce los elementos en orden ascendente según la clave, que en este sistema corresponde al año simbólico de nacimiento más el mes.
Aplicado a la genealogía mitológica, permite generar una línea temporal perfectamente ordenada, mostrando a los seres más antiguos primero y a los más recientes al final. Por ello, es el recorrido ideal para elaborar cronologías, etapas mitológicas y secuencias históricas dentro del árbol.

Recorrido PREORDEN (Raíz → Izquierda → Derecha)

El recorrido Preorden procesa primero la raíz, luego desciende por el subárbol izquierdo y finalmente explora el derecho.

Este método permite observar la jerarquía natural del árbol, ya que se inicia siempre desde el nodo más importante o más reciente de la genealogía (raíz del ABB inicial o cargado).
En el contexto del árbol mitológico, el Preorden ofrece una vista estructurada de líneas familiares, mostrando a cada dios o titán junto a sus descendientes inmediatos.
Es útil para representar organigramas mitológicos, árboles jerárquicos y estructuras donde el rol dominante o el ancestro principal debe visualizarse primero.

Recorrido POSTORDEN (Izquierda → Derecha → Raíz)

El Postorden recorre primero los subárboles izquierdo y derecho, procesando el nodo actual únicamente al final.
Este enfoque revela la estructura desde los elementos más específicos hasta los más generales, es decir, desde las generaciones más jóvenes hacia los progenitores.
Es particularmente adecuado para analizar descendencias completas, estudiar ramas familiares específicas o preparar la eliminación segura de nodos (ya que garantiza que todos los descendientes se procesen antes que el nodo padre).
En genealogía mitológica, el Postorden permite identificar líneas de sangre completas y visualizar cómo se organizan las distintas familias dentro del árbol.

2.3 Pseudocódigo: Crear un Árbol Binario (Insertar nodo en ABB)

```
ALGORITMO Insertar(raiz, nacimiento, nombre, rol)
    SI raiz ES NULO ENTONCES
        // Crear un nuevo nodo con la información proporcionada
        nuevoNodo ← CrearNodo(nacimiento, nombre, rol)
        RETORNAR nuevoNodo
    FIN SI

    // Si la fecha de nacimiento es menor que la raíz, insertamos en el subárbol izquierdo
    SI nacimiento < raiz.nacimiento ENTONCES
        raiz.izq ← Insertar(raiz.izq, nacimiento, nombre, rol)

    // Si la fecha de nacimiento es mayor que la raíz, insertamos en el subárbol derecho
    SINO SI nacimiento > raiz.nacimiento ENTONCES
        raiz.der ← Insertar(raiz.der, nacimiento, nombre, rol)

    // Si la fecha de nacimiento es igual a la raíz, mostramos un mensaje de error
    SINO
        MOSTRAR "Clave duplicada. No se puede insertar."
    FIN SI

    // Retornamos la raíz (sin cambios si no se insertó nada)
    RETORNAR raiz
FIN ALGORITMO
```

La función Inserción coloca cada nuevo ser mitológico en el lugar correcto del ABB comparando su año simbólico de nacimiento con el de los nodos existentes.

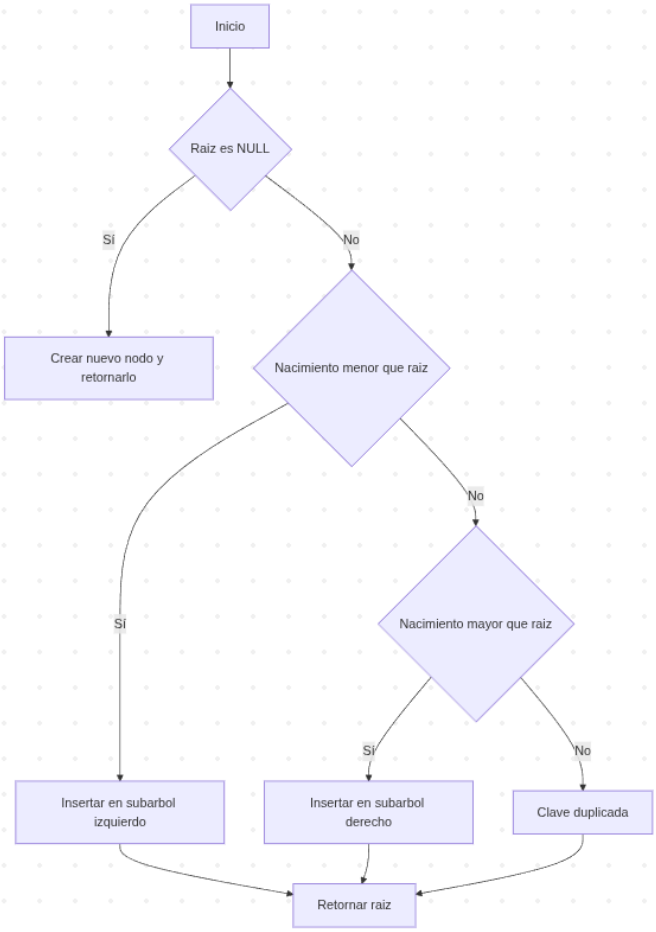
2.4 Pseudocódigo: eliminar un miembro del ABB

```
ALGORITMO Eliminar(raiz, nacimiento)
    SI raiz ES NULO ENTONCES
        // No se encontró la clave
        RETORNAR NULL
    FIN SI

    // Buscar el nodo a eliminar
    SI nacimiento < raiz.nacimiento ENTONCES
        raiz.izq ← Eliminar(raiz.izq, nacimiento)

    SINO SI nacimiento > raiz.nacimiento ENTONCES
        raiz.der ← Eliminar(raiz.der, nacimiento)

    SINO
        // Caso 1: Nodo sin hijos (hoja)
        SI raiz.izq ES NULL Y raiz.der ES NULL ENTONCES
            eliminar raiz
            RETORNAR NULL
        FIN SI

        // Caso 2: Nodo con un solo hijo (derecho)
        SI raiz.izq ES NULL ENTONCES
            temp ← raiz.der
            eliminar raiz
            RETORNAR temp
        FIN SI

        // Caso 2: Nodo con un solo hijo (izquierdo)
        SI raiz.der ES NULL ENTONCES
            temp ← raiz.izq
            eliminar raiz
            RETORNAR temp
        FIN SI

        // Caso 3: Nodo con dos hijos
        sucesor ← Minimo(raiz.der)
        copiar datos de sucesor a raiz
        raiz.der ← Eliminar(raiz.der, sucesor.nacimiento)
    FIN SI

    RETORNAR raiz
FIN ALGORITMO
```

Este algoritmo mantiene la estructura del árbol al eliminar un nodo, usando el sucesor inorden si es necesario.

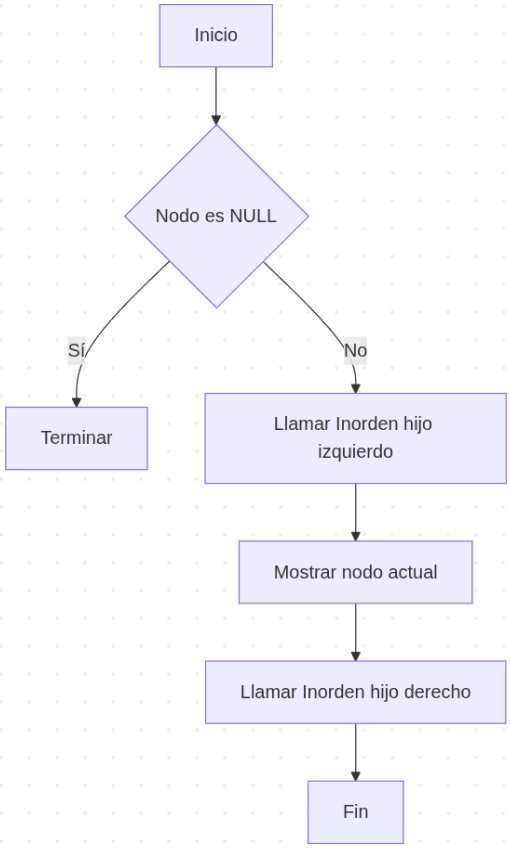
3. DIAGRAMAS DE FLUJO

DIAGRAMA 1 – Proceso de Inserción en un Árbol Binario de Búsqueda (ABB)



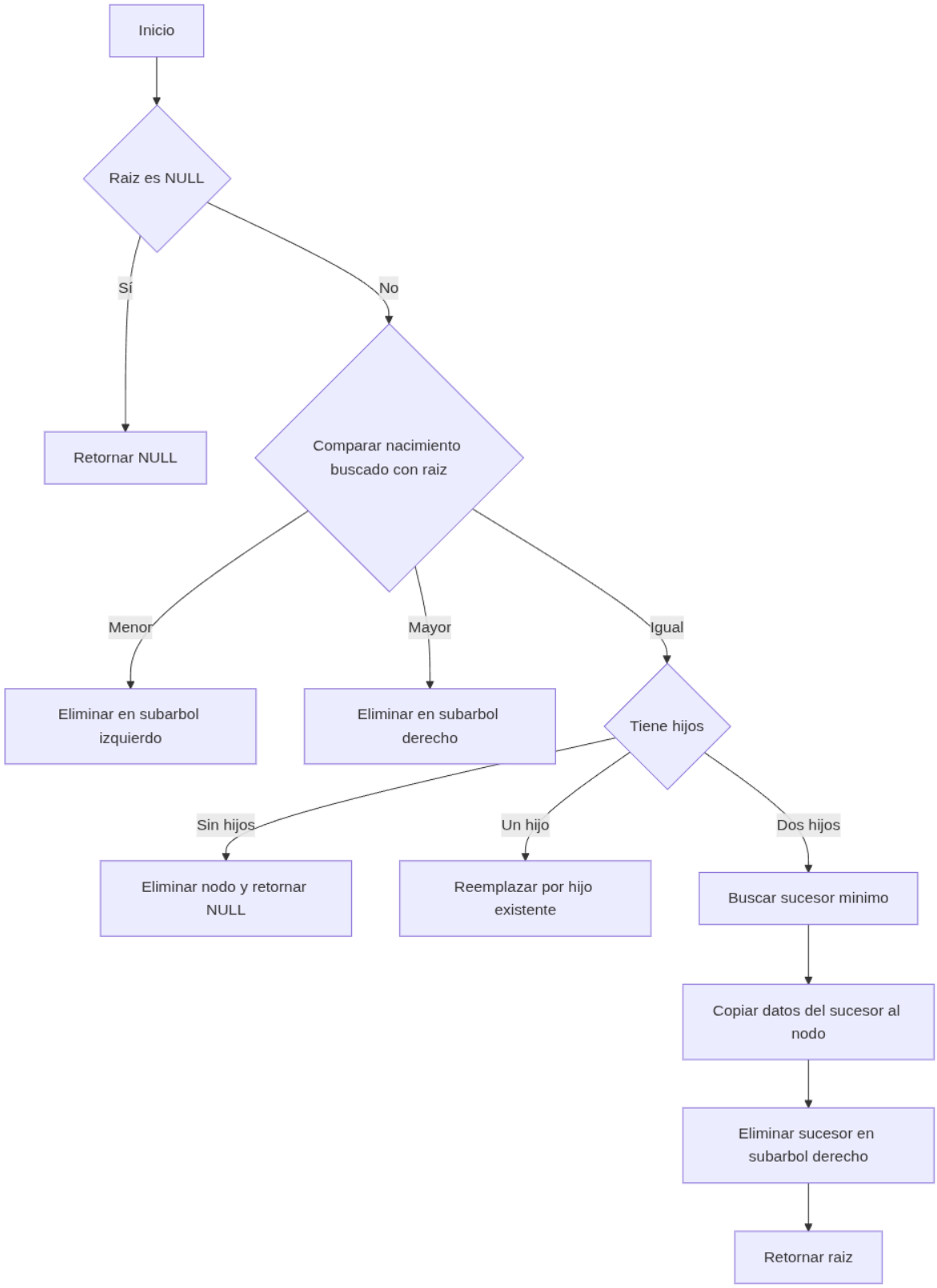
Este diagrama muestra cómo el algoritmo decide en qué parte del árbol insertar un nuevo miembro basado en su año simbólico.
Si el árbol está vacío, crea la raíz; si el año es menor avanza por el subárbol izquierdo, si es mayor avanza por el derecho; y si es igual, detecta duplicados.

DIAGRAMA 2 — Recorrido INORDEN (Izq – Raíz – Der)



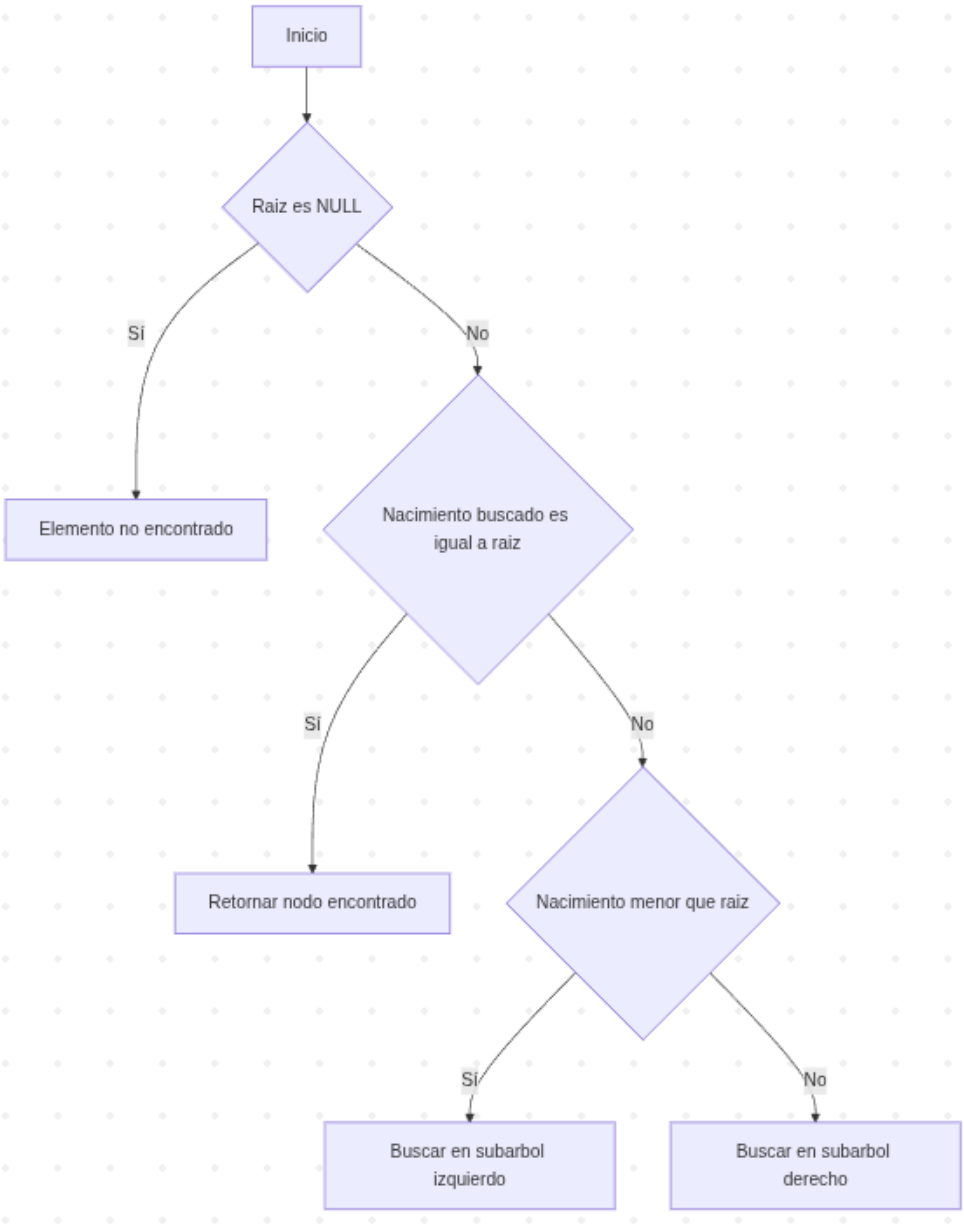
Este recorrido muestra los elementos del árbol en orden ascendente según su año simbólico.
Primero explora el subárbol izquierdo (miembros más antiguos), luego muestra el nodo actual y finalmente el subárbol derecho (miembros más recientes).

DIAGRAMA 3 – Eliminación en un Árbol Binario de Búsqueda (ABB)



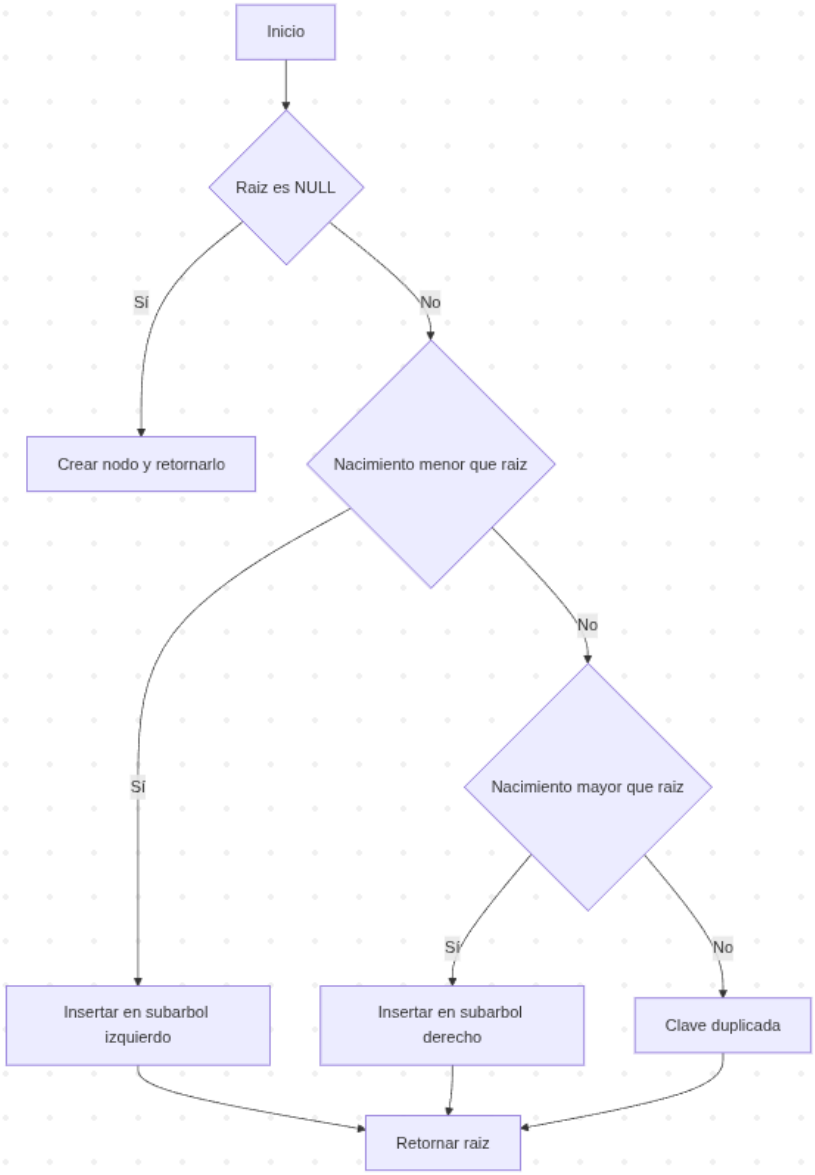
Este diagrama explica el proceso para eliminar un nodo de un ABB manteniendo el orden. El algoritmo distingue entre nodos sin hijos, con un hijo, o con dos hijos (donde es necesario reemplazar con el sucesor mínimo del subárbol derecho).

DIAGRAMA 4 – Búsqueda en ABB



El algoritmo de búsqueda compara la clave buscada con el nodo actual y decide si explorar izquierda, derecha o finalizar si encuentra el valor.

DIAGRAMA 5 – Inserción en ABB (para referencia)



El proceso inserta un nuevo nodo respetando el orden del ABB: menores a la izquierda, mayores a la derecha, evitando duplicados.

4. AVANCE DEL CÓDIGO FUENTE

1.1 Definición del nodo

```
struct Nodo {
  int anio;
  int mes;
  int clave;
  string nombre;
  string rol;
  Nodo *izq;
  Nodo *der;

  Nodo(int a, int m, const string &nom, const string &r) {
    anio = a;
    mes = m;
    clave = a * 100 + m;
    nombre = nom;
    rol = r;
    izq = NULL;
    der = NULL;
  }
};
```

Este bloque define la estructura fundamental del Árbol Binario de Búsqueda.
Cada nodo representa a un ser mitológico y almacena su año y mes (para formar una clave única), nombre, rol y los punteros a sus hijos izquierdo y derecho. Esta estructura permite organizar a los personajes siguiendo un orden temporal.

1.2 Función Insertar

```
Nodo *insertar(Nodo *raiz, int anio, int mes, const string &nombre,
               const string &rol) {

  int clave = anio * 100 + mes;

  if (raiz == NULL)
    return new Nodo(anio, mes, nombre, rol);

  if (clave < raiz->clave)
    raiz->izq = insertar(raiz->izq, anio, mes, nombre, rol);
  else if (clave > raiz->clave)
    raiz->der = insertar(raiz->der, anio, mes, nombre, rol);
  else
    cout << "Aviso: ya existe un miembro con ese anio y mes.\n";

  return raiz;
}
```

Esta función inserta nuevos personajes en el ABB respetando el orden definido por la clave año–mes. Si la clave es menor, va al subárbol izquierdo; si es mayor, al derecho. Si

coincide, se detecta un duplicado. Esto garantiza que la genealogía se mantenga ordenada cronológicamente.

1.3 Función Buscar

```c
Nodo *buscarExacto(Nodo *raiz, int anio, int mes) {
  int clave = anio * 100 + mes;

  if (raiz == NULL)
    return NULL;

  if (clave == raiz->clave)
    return raiz;

  if (clave < raiz->clave)
    return buscarExacto(raiz->izq);

  return buscarExacto(raiz->der);
}
```

Este algoritmo localiza un nodo exacto en el ABB comparando la clave buscada con la del nodo actual. Según el valor, continúa la búsqueda por la izquierda o la derecha. Su eficiencia proviene del orden del ABB, permitiendo localizar personajes rápidamente.

1.4 Función ELiminar

```
Nodo *eliminarMiembro(Nodo *raiz, int anio, int mes) {
  if (raiz == NULL)
    return raiz;

  int clave = anio * 100 + mes;

  if (clave < raiz->clave)
    raiz->izq = eliminarMiembro(raiz->izq, anio, mes);
  else if (clave > raiz->clave)
    raiz->der = eliminarMiembro(raiz->der, anio, mes);
  else {
    if (raiz->izq == NULL) {
      Nodo *temp = raiz->der;
      delete raiz;
      return temp;
    }
    if (raiz->der == NULL) {
      Nodo *temp = raiz->izq;
      delete raiz;
      return temp;
    }
    Nodo *suc = minimo(raiz->der);
    raiz->anio = suc->anio;
    raiz->mes = suc->mes;
    raiz->clave = suc->clave;
    raiz->nombre = suc->nombre;
    raiz->rol = suc->rol;
    raiz->der = eliminarMiembro(raiz->der, suc->anio, suc->mes);
  }
  return raiz;
}
```

La función eliminar gestiona correctamente los tres casos clásicos del ABB: nodos sin hijos, con un solo hijo y con dos hijos. En este último caso utiliza el sucesor inorden para mantener el árbol ordenado. Este método asegura la integridad estructural del árbol tras cada eliminación.

1.5 Implementación en ASCII del arbol

```
void imprimirArbolAscii(Nodo *raiz, string prefix, bool esUltimo) {
  if (raiz == NULL)
    return;

  cout << prefix << (esUltimo ? "└── " : "├── ")
       << raiz->nombre << " [" << raiz->anio << "-" << raiz->mes
       << "] (" << raiz->rol << ")\n";

  string nuevoPrefix = prefix + (esUltimo ? "    " : "│   ");

  if (raiz->izq != NULL && raiz->der != NULL) {
    imprimirArbolAscii(raiz->izq, nuevoPrefix, false);
    imprimirArbolAscii(raiz->der, nuevoPrefix, true);
  } else if (raiz->izq != NULL) {
    imprimirArbolAscii(raiz->izq, nuevoPrefix, true);
  } else if (raiz->der != NULL) {
    imprimirArbolAscii(raiz->der, nuevoPrefix, true);
  }
}
```

Este procedimiento genera una representación visual del ABB en formato ASCII, mostrando jerarquía y ramas del árbol. Gracias a los prefijos y caracteres especiales, el usuario puede visualizar claramente las relaciones genealógicas entre los personajes mitológicos.