SOLUCIÒN DE PROBLEMAS DE INGENIERÍA

Tema : Estructuras de datos dinámicas lineales

Asignatura : Estructura de Datos

NRC : 29904

Docente : Osorio Contreras Rosario Delia

Integrantes :

- Benites Fierro Alexander Aiken
- De la Torre Martinez Johann Brenner
- Pandal Aquino Denis Jesús
- Villena Vasquez Carlos Educardo

Huancayo, junio del 2025

Pregunta esencial:

¿Cómo se puede aplicar un árbol binario de búsqueda para representar y consultar un árbol genealógico?

Un árbol binario de búsqueda (ABB) puede aplicarse para representar un árbol genealógico si se establece una regla clara para ordenar los miembros. En este caso, se podría usar el año de nacimiento, el nombre o un identificador único (ID) como criterio principal de orden. Cada nodo del ABB representa a una persona, y sus punteros izquierdo y derecho representan posibles descendientes u otros miembros relacionados jerárquicamente, de acuerdo al criterio. El ABB facilita la búsqueda eficiente de miembros, su inserción ordenada y la visualización de la estructura familiar mediante recorridos como inorden, preorden y postorden. Esta implementación, aunque simplificada respecto a una genealogía real, es eficaz para responder consultas como encontrar ancestros, descendientes, ramas específicas o eliminar personas del árbol sin perder su organización.

¿Qué información se debe almacenar en cada nodo del árbol?

Cada nodo debe contener los datos personales de un miembro de la civilización antigua. Los campos principales recomendados son: un identificador único (por ejemplo, un número o nombre), el nombre completo, el año de nacimiento, y el género. Esta información permite distinguir y organizar a cada individuo. Además, se deben incluir dos punteros: uno al hijo izquierdo y otro al hijo derecho, cumpliendo con la estructura típica de un ABB.

Opcionalmente, se pueden incluir referencias a los padres, nivel generacional o estado civil, pero se debe evitar sobrecargar el nodo con datos innecesarios si el objetivo es mantener la eficiencia en las consultas y el mantenimiento del árbol.

¿Cómo insertar y eliminar miembros del árbol sin romper su estructura?

La inserción debe seguir la lógica del ABB: si el nuevo miembro tiene un valor (por ejemplo, ID o año de nacimiento) menor al del nodo actual, se ubica en el subárbol izquierdo; si es mayor, en el subárbol derecho. Esto asegura que el árbol mantenga su estructura ordenada.

La eliminación es más compleja. Si el nodo a eliminar no tiene hijos, simplemente se elimina. Si tiene un solo hijo, se reemplaza con su hijo. Si tiene dos hijos, se reemplaza por el nodo con el menor valor del subárbol derecho (su sucesor inorden).

¿Qué métodos permiten recorrer el árbol para visualizar la genealogía?

Para visualizar la genealogía, se utilizan métodos de recorrido del árbol. Existen tres formas principales: preorden, inorden y postorden.

En preorden, se visita primero el nodo actual, luego el izquierdo y después el derecho. Ideal para mostrar jerarquía familiar desde el miembro más antiguo.

En inorden, se visita primero el izquierdo, luego el nodo y después el derecho. Útil para mostrar la genealogía en orden creciente del criterio usado (como ID o año).

En postorden, se visita primero ambos hijos y luego el nodo. Se usa para eliminar nodos o mostrar generaciones desde los más jóvenes hacia los más antiguos.

¿Cómo determinar si un miembro pertenece a una rama específica?

Para saber si un miembro pertenece a una rama específica, se realiza una búsqueda en el árbol a partir del nodo raíz. Si durante el recorrido se encuentra el nodo con el ID buscado, se puede verificar si está en el subárbol izquierdo o derecho de un ancestro particular. Este método se puede aplicar recursivamente desde cualquier nodo que represente el punto de inicio de una rama (por ejemplo, un patriarca o matriarca).

¿Cómo balancear el árbol si se vuelve demasiado profundo?

Un árbol binario de búsqueda puede volverse desequilibrado si las inserciones no son balanceadas, causando que una rama sea mucho más profunda que otra. Esto provoca búsquedas lentas. Para evitarlo, se pueden aplicar técnicas de balanceo, como la conversión a un árbol AVL o árbol rojo-negro.

- Sin embargo, en proyectos simples como este, una alternativa razonable es:
-
- Recolectar los nodos en un vector mediante un recorrido inorden (quedando ordenados).

Para balancear un árbol binario de búsqueda (ABB) que se vuelve demasiado profundo, se puede implementar un árbol auto-balanceado como el árbol AVL, que ajusta su estructura automáticamente mediante rotaciones (simples o dobles) cada vez que una inserción o eliminación provoca un desequilibrio en la altura de sus subárboles; por ejemplo, si al insertar los nodos 30, 20, 10 el árbol se vuelve lineal hacia la izquierda, una rotación simple a la derecha en el nodo 30 restablece el equilibrio, dejando a 20 como raíz con 10 y 30 como hijos izquierdo y derecho respectivamente.

Descripción del problema

Un equipo de arqueólogos ha descubierto información referente a una antigua civilización cuyas relaciones familiares y linajes pueden modelarse como un árbol genealógico. Se necesita una aplicación que permita almacenar, organizar y consultar dicha información de manera eficiente, atendiendo a operaciones típicas sobre genealogías: agregar nuevos individuos, eliminar registros existentes, buscar ancestros o descendientes, determinar pertenencia a ramas familiares, y mostrar la estructura de la genealogía. Dado el volumen potencial de datos y el acceso frecuente a consultas de ancestros y descendientes, se opta por representar la colección de individuos mediante una estructura de Árbol Binario de Búsqueda (ABB), utilizando un criterio de orden (por ejemplo, un identificador único o año de nacimiento combinado con otro dato) que permita búsquedas, inserciones y eliminaciones en tiempo proporcional a la altura del árbol.

Aunque un árbol genealógico natural puede tener más de dos hijos por individuo, la representación con ABB requiere un criterio de orden que “apile” los miembros en una estructura con punteros izquierdo y derecho. De este modo, cada operación de consulta o modificación aprovecha la propiedad de orden del ABB para recorrer de forma eficiente. Adicionalmente, al no usar contenedores de la biblioteca estándar (como vector) ni bibliotecas externas, el manejo de memoria y estructuras será manual, con punteros y arreglos básicos, ajustado al entorno de Dev C++. Asimismo, se debe prever un mecanismo de balanceo o reequilibrado puntual cuando la estructura se degrade (por ejemplo, tras muchas inserciones sesgadas), para evitar que las operaciones se tornen lineales en tiempo.

La aplicación deberá operar principalmente en consola, permitiendo al usuario (arquéologo o asistente técnico) interactuar: registrar individuos, consultar relaciones, mostrar listados, y ejecutar procesos de mantenimiento del árbol. También se espera disponer de un método de almacenamiento persistente en archivos (por ejemplo, texto o binario) para recargar la genealogía entre sesiones, aunque este aspecto dependerá de los requerimientos específicos de proyecto.

Requerimientos del sistema

1. Requerimientos funcionales

1. **Registro de un nuevo individuo**

    - Permitir al usuario ingresar un nuevo miembro en la genealogía, solicitando los datos mínimos: identificador único (numérico o combinación que garantice unicidad), nombre o etiqueta textual, año de nacimiento u otro dato relevante para orden, género u otros metadatos deseados.

    - Al insertar, la aplicación debe ubicar el nuevo nodo en la posición correcta del ABB, manteniendo la propiedad de orden.

2. **Eliminación de un individuo existente**

    - Facilitar la eliminación de un miembro dado su identificador.

    - Implementar la lógica de eliminación en ABB: casos de nodo sin hijos, con un hijo y con dos hijos, ajustando los enlaces de punteros para mantener la estructura.

    - Si el individuo a eliminar no existe, notificar al usuario apropiadamente.

3. **Búsqueda de un individuo**

    - Dado un identificador (o criterio de búsqueda definido), buscar recursiva o iterativamente en el ABB y presentar al usuario los detalles del individuo si se encuentra.

    - Informar claramente en caso de no hallazgo.

4. **Consulta de ancestros**

    - Dada la representación en ABB, y partiendo de un individuo, permitir obtener la “ruta” hacia ancestros según la lógica de datos disponibles (esto puede requerir punteros a padre o bien buscar iterativamente en el árbol completo para reconstruir la cadena ascendente).

    - Mostrar al usuario la lista de ancestros en orden, desde el progenitor directo hasta los antepasados más remotos registrados.

5. **Consulta de descendientes**

    - A partir de un individuo considerado como “ancestro” en la genealogía, mostrar todos los descendientes registrados en el subárbol correspondiente.

    - Realizar un recorrido recursivo o iterativo del subárbol que parte del nodo de ese individuo, listando todos sus descendientes en orden (preorden, inorden o postorden según criterio de presentación).

6. **Determinación de pertenencia a rama específica**

    - Dado un ancestro y otro individuo, determinar si éste último pertenece a la rama descendiente del primero.

    - Usar búsqueda en el subárbol del ancestro: si se encuentra el identificador del individuo en dicho subárbol, confirmar pertenencia; de lo contrario, indicar ausencia.

7. **Recorridos para visualización de genealogía**

    - Ofrecer al usuario la opción de recorrer y visualizar la estructura completa o parcial de la genealogía mediante preorden, inorden o postorden, explicando el significado de cada uno.

    - Incluir en la presentación información relevante en cada nodo: nombre, año de nacimiento, nivel generacional (que puede calcularse durante el recorrido llevando un contador de profundidad), y otros metadatos.

8. **Balanceo o reequilibrado del árbol**

    - Detectar, de manera manual o por umbral definido, cuando la profundidad del ABB se ha vuelto excesiva en relación con el número de nodos.

    - Permitir al usuario ejecutar una acción de reequilibrio: extraer referencias a todos los nodos en un arreglo de tamaño suficiente (realizado mediante recorrido inorden) y reconstruir el ABB seleccionando el elemento medio como raíz y repitiendo recursivamente.

    - Notificar al usuario antes y después del proceso, mostrando la mejora estimada en altura si se desea (por ejemplo, imprimiendo la altura antigua y la nueva).

9. **Persistencia de datos**

    - Proporcionar mecanismos de guardado y carga desde archivo: guardar la genealogía en un formato sencillo (texto o binario) que permita recargarla en ejecuciones posteriores.

    - Al cargar, reconstruir el ABB con la información guardada, respetando el mismo criterio de orden.

    - Manejar errores de lectura/escritura y notificar fallos o inconsistencias en los datos.

10. **Interfaz de usuario en consola**

    - Presentar un menú claro con opciones para cada operación: insertar, eliminar, buscar, consultar ancestros/descendientes, determinar pertenencia, mostrar recorridos, reequilibrar, guardar, cargar, salir.

    - Validar entradas del usuario (por ejemplo, verificar que un identificador sea numérico, que el año tenga un formato esperado, que no se inserten duplicados sin confirmación, etc.).

    - Mostrar mensajes de estado y resultados de forma comprensible, con retroalimentación clara ante errores o confirmaciones de éxito.

11. **Manejo de errores y validaciones**

    - Verificar condiciones antes de operar: p. ej., al intentar eliminar o buscar, comprobar si la raíz es nula o si el árbol está vacío.

    - Control de duplicados: si el usuario intenta registrar un individuo con un identificador ya existente, permitir decisión (rechazar, actualizar datos o asignar subclave secundaria) según lógica definida.

    - Validar límites de arreglos usados para balanceo (asegurar que el número de nodos no exceda la capacidad prevista o gestionar ampliación de forma manual controlada).

12. **Informe o reporte básico**

    - Generar un informe simple en consola (o archivo de texto) que resuma estadísticas: número total de individuos, altura actual del árbol, número de hojas, número de nodos por nivel, etc.

    - Esto apoya al arqueólogo a entender la densidad de datos y posibles secciones menos desarrolladas de la genealogía.

13. **Documentación de uso**

    - Incluir dentro del programa, o como un documento separado, instrucciones claras para compilar en Dev C++ y ejecutar la aplicación, así como descripción de cada opción de menú, formato de archivos de persistencia y recomendaciones de mantenimiento.

2. Requerimientos no funcionales

1. **Portabilidad y dependencia mínima**

    - El programa debe compilar y ejecutarse en entornos Dev C++ estándar, sin usar bibliotecas externas más allá de las básicas de C++ (p. ej., `<iostream>`, `<fstream>`, `<string>`). No se debe requerir `<vector>`, `<list>` u otros contenedores de la STL, para ajustarse a la restricción de manejo manual de memoria y estructuras básicas.

    - El código debe ser compatible con compiladores de C++ comunes en Windows (Dev C++) y, en la medida de lo posible, fácilmente adaptable a otros entornos de C++ sin dependencias específicas de plataforma.

2. **Rendimiento y escalabilidad**

    - Las operaciones de búsqueda, inserción y eliminación deben ejecutarse en tiempo proporcional a la altura del ABB. Se espera que, con un árbol balanceado, la altura crezca aproximadamente como O(log n).

    - El balanceo puntual debe realizarse de forma eficiente en tiempo O(n) cuando se ejecute, siendo tolerable para tamaños moderados de genealogías (según capacidad de memoria y CPU de la máquina).

    - Se debe prever un límite razonable de nodos en función de la memoria disponible en ejecución; si se usan arreglos estáticos para reequilibrio, documentar el tamaño máximo o mecanismo de ampliación manual.

3. **Uso de memoria**

    - El manejo de memoria es manual: cada nodo debe asignarse dinámicamente y liberarse correctamente al eliminar. Se debe evitar fugas de memoria.

    - Si se implementa almacenamiento temporal en arreglos para balanceo o para informes, debe controlarse el tamaño y liberar o reutilizar la memoria cuando corresponda.

- Especificar (en documentación) un tamaño máximo estimado de genealogía acorde al entorno (por ejemplo, decenas de miles de nodos pueden requerir consideraciones especiales de memoria).

4. **Usabilidad y experiencia de usuario en consola**

   - El menú y las indicaciones deben ser claras y amigables, con descripciones de cada función y mensajes de confirmación o error bien detallados.

   - Validar entradas del usuario con mensajes precisos (por ejemplo, “El identificador debe ser un entero positivo; inténtelo de nuevo”).

   - Incluir opciones para cancelar o retroceder en la interacción, y para mostrar ayuda contextual si el usuario la solicita.

5. **Mantenibilidad y claridad del código**

   - El código fuente debe organizarse en funciones o módulos claros (por ejemplo, funciones para inserción, eliminación, búsqueda, recorridos, balanceo, persistencia), con comentarios que expliquen la lógica central.

   - Evitar macros complejas o técnicas poco legibles; preferir funciones con nombres descriptivos.

   - Documentar las estructuras de datos usadas y la lógica de manejo de punteros para facilitar futuras extensiones o correcciones.

6. **Robustez y manejo de errores**

   - Detectar y manejar situaciones inesperadas: árbol vacío, entrada inválida, archivo de persistencia corrupto o inaccesible, falta de memoria al asignar un nodo, etc.

   - Ante errores graves (por ejemplo, fallo de asignación con `new`), informar al usuario y terminar de forma controlada o intentar recuperación mínima si es posible.

   - Evitar que el programa quede en estado inconsistente tras errores; garantizar que, si una operación falla, el árbol permanezca en estado coherente o bien se reviertan los cambios.

7. **Documentación y legibilidad**

   - Acompañar el código con un documento o comentarios que expliquen la estructura general, cómo compilar en Dev C++, cómo ejecutar y cómo interpretar resultados.

- Definir claramente el formato de archivos de guardado (por ejemplo, texto con líneas “ID,nombre,año,género” o binario con registros fijos) para que otros desarrolladores o arqueólogos puedan crear o procesar estos archivos con herramientas externas si fuera necesario.

8. **Extensibilidad**

   - La arquitectura debe permitir, en el futuro, incorporar funcionalidades adicionales: por ejemplo, incorporación de punteros a padre, manejo de múltiples criterios de orden (caso de empates en año de nacimiento), relaciones colaterales (hermanos, primos), o incluso migración a una estructura de datos más compleja (como un grafo si se incorporan relaciones matrimoniales).

   - Diseñar la interfaz y las estructuras de manera que agregar nuevos campos de metadatos en el nodo o nuevas opciones de consulta requiera cambios localizados y no reescritura masiva.

9. **Seguridad de datos**

   - Aunque se trata de datos arqueológicos y no personales sensibles en el sentido convencional, se debe asegurar que los archivos de persistencia no se corrompan: usar métodos de escritura segura (por ejemplo, escribir primero en un archivo temporal y luego renombrar).

   - Validar los datos que se cargan, garantizando que cumplen el formato esperado, para evitar comportamiento indefinido.

10. **Portabilidad futura hacia GUI o integración**

    - Aunque inicialmente la interfaz es de consola, estructurar el código de manera que la lógica del árbol y acceso a datos esté desacoplada de la capa de presentación. Así, en el futuro se podría portar la lógica a una interfaz gráfica o exponer como biblioteca para otras aplicaciones sin reescribir la parte central de gestión de ABB.