Integrantes: Jean Carlo Londoño y Alejandro Garcés Ramírez

1. **Explique detalladamente el algoritmo utilizado para convertir el nodo especificado en la nueva raíz del árbol binario, considerando el manejo de nodos con claves iguales.**

1. **Buscar el nodo especificado:** Se inicia buscando el nodo específico que se desea convertir en la nueva raíz del árbol. Esto se hace utilizando el método **buscarNodo**, comenzando desde la raíz actual y buscando el nodo con la clave (en este caso, la cédula) que coincide con la clave especificada.

2. **Verificar la existencia del nodo**: Una vez encontrado el nodo especificado, se verifica si realmente existe en el árbol. Si no se encuentra el nodo con la clave especificada, se muestra un mensaje de error indicando que el nodo no se encontró y se interrumpe el proceso. Esto se hace mediante la línea:

```
if (nodo == nullptr)
{
    std::cerr << "No se encontró el nodo especificado." << std::endl;
    return nullptr;
}
```

3. **Verificar si el nodo es la raíz actual:** Se comprueba si el nodo especificado ya es la raíz actual del árbol. Si es así, se muestra un mensaje indicando que el nodo ya es la raíz, y no es necesario realizar ningún cambio en la estructura del árbol.

4. **Eliminar el nodo especificado del árbol:** Si el nodo especificado no es la raíz actual, se procede a eliminarlo del árbol. Esto se hace mediante un proceso de eliminación recursiva (**EliminarRecursivo**), que maneja correctamente los casos en los que el nodo a eliminar tiene cero, uno o dos hijos.

```
// Verificar si el nodo ya es la raíz
if (nodo == raiz)
{
    std::cout << "El nodo seleccionado ya es la raíz." << std::endl;
    return nodo;
}
```

5. **Establecer el nuevo nodo raíz:** Una vez eliminado el nodo especificado, se lo convierte en la nueva raíz del árbol. Esto implica cambiar el puntero **raiz** para que apunte al nuevo nodo y restablecer adecuadamente los punteros de los hijos izquierdo y derecho de la nueva raíz.

6. **Conectar los subárboles originales al nuevo nodo raíz:** Utilizamos una función recursiva llamada **conectarSubarboles** para reconectar los subárboles originales al nuevo nodo raíz. Esta función recorre los subárboles originales y los inserta en el nuevo árbol, asegurándose de no insertar nodos duplicados.

```
raiz = nodo;
// Conectar los subárboles izquierdo y derecho originales al nuevo nodo raíz
conectarSubarboles(raizOriginal, raiz);
```

7. **Mostrar información sobre la nueva raíz y los subárboles originales:** Después de completar la actualización de referencias, se muestra información relevante sobre la nueva raíz.
   Se imprime un mensaje que indica el nombre, la cédula y el programa académico del nuevo nodo raíz.

8. **Devolver el puntero al nuevo nodo raíz:** Finalmente, se devuelve el puntero al nuevo nodo raíz del árbol. Esto permite que el programa principal pueda utilizar la nueva raíz para futuras operaciones en el árbol.

Paso adicional: Manejar raíces iguales

Antes de establecer el nuevo nodo como la raíz, se verifica si la cédula del nodo especificado coincide con la cédula de la raíz actual.

Si las cédulas son iguales, significa que ya existe un nodo con esa cédula como raíz del árbol, lo cual no es válido ya que cada persona debe tener una cédula única.

En este caso, se lanza un error indicando que no se puede elegir el nodo especificado como nueva raíz porque ya existe otro nodo con la misma cédula.

Esto evita que se produzcan conflictos en el árbol debido a la duplicación de cédulas, lo cual es importante para mantener la integridad de los datos.

Aquí está el código que implementa este paso adicional:

```
// Verificar si la cédula ya existe en el árbol
if (buscarCedula(raiz, cedula))
{
    std::cerr << "La cédula ingresada ya existe. Por favor, ingrese una cédula válida." << std::endl;
    return;
}
```

Esto es importante para nosotros como futuros ingenieros de sistemas, ya que siempre debemos de buscar la manera más optima y segura de manejar datos.

2. **¿Qué casos especiales deben considerarse al implementar la operación de cambio de raíz en un árbol binario?**

   Árbol vacío: No se podrá poder imprimir el árbol, ni eliminar un nodo, ni elegir una nueva raíz si el árbol se encuentra vacío. Este es el caso especial más básico del programa.

Raíz nula: Verificar si la raíz actual es nula. En este caso, no se puede cambiar la raíz porque el árbol está vacío.

Nodo seleccionado ya es raíz: Verificar si el nodo seleccionado como nueva raíz ya es la raíz actual del árbol. En este caso, no es necesario realizar ningún cambio en la estructura del árbol.

Nodo seleccionado como nueva raíz no existe: Verificar si el nodo especificado como la nueva raíz existe en el árbol. Si el nodo no se encuentra en el árbol, no se puede establecer como nueva raíz.

Conflictos de cédula: En el caso específico de árboles basados en la cédula de identidad, como en el ejemplo provisto, es importante verificar que no haya conflictos de

cédula al seleccionar un nuevo nodo como raíz. Cada persona debe tener una cédula única, por lo que no se pueden tener dos nodos con la misma cédula como raíz.

Actualización de referencias: Después de establecer el nuevo nodo como raíz, se deben actualizar las referencias en el árbol para garantizar que todos los nodos mantengan sus conexiones correctamente. Esto implica reconectar los subárboles originales al nuevo nodo raíz y actualizar cualquier referencia que apunte a la raíz anterior

3. **Dibuje diagramas de flujo o pseudocódigo que representen el algoritmo de cambio de raíz y los métodos de la clase árbol. Se recomienda representar los pasos del algoritmo.**

// Definir la clase Árbol

Clase Árbol

// Método para insertar un nuevo nodo en el árbol

Función insertar(cedula, nombre, programa_academico)

Si buscarCedula(raiz, cedula) devuelve verdadero

Mostrar "La cédula ingresada ya existe. Por favor, ingrese una cédula válida."

Devolver

Fin Si

Crear un nuevo objeto Dato con los datos proporcionados

Crear un puntero compartido item apuntando al nuevo Dato

Insertar el nuevo dato en el árbol recursivamente utilizando insertarRecursivo

Fin Función

// Método para buscar si una cédula específica existe en el árbol

Función buscarCedula(nodo, cedula)

Si nodo es nulo

Devolver falso

Fin Si

Si nodo->dato->cedula es igual a cedula

Devolver verdadero

Fin Si

Devolver buscarCedula(nodo->izq, cedula) o buscarCedula(nodo->der, cedula)

Fin Función

// Método para eliminar un nodo del árbol dado su número de cédula

Función eliminar(cedula)

Si la raíz es nula

Devolver falso // Si el árbol está vacío, no se puede eliminar

Fin Si

Llamar a eliminarRecursivo con la raíz y la cédula especificada

Asignar el resultado a la raíz

Devolver verdadero

Fin Función

// Método para elegir una nueva raíz para el árbol basado en una cédula dada

Función elegirRaiz(cedula)

Guardar una copia del árbol original

Buscar el nodo que el usuario desea elegir como la nueva raíz

Si el nodo no se encuentra

Mostrar "No se encontró el nodo especificado."

Devolver nulo

Fin Si

Si el nodo ya es la raíz

Mostrar "El nodo seleccionado ya es la raíz."

Devolver el nodo

Fin Si

Eliminar recursivamente ese nodo del árbol

Reemplazar la raíz actual con el nodo elegido como nueva raíz

Conectar los subárboles izquierdo y derecho originales al nuevo nodo raíz utilizando conectarSubarboles

Mostrar información sobre la nueva raíz y los subárboles originales

Devolver el nodo

Fin Función

// Método para conectar los subárboles originales al nuevo nodo raíz

Función conectarSubarboles(original, nuevaRaiz)

Si el original es nulo

Devolver

Fin Si

Recorrer el subárbol original en orden

Llamar a conectarSubarboles con el hijo izquierdo de original y nuevaRaiz

Insertar el nodo actual en el nuevo subárbol si no está presente

Llamar a insertarRecursivo con nuevaRaiz y los datos del nodo actual

Llamar a conectarSubarboles con el hijo derecho de original y nuevaRaiz

Fin Función

// Otros métodos de la clase Árbol: insertarRecursivo, eliminarRecursivo, buscarNodo, minValorNodo, imprimirArbol, imprimirArbolRecursivo

// Método privado para realizar la inserción recursiva de un nodo en el árbol

Función insertarRecursivo(nodo, dato)

Si el nodo es nulo

Crear un nuevo nodo con el dato proporcionado

Devolver el nuevo nodo

Fin Si

Si dato->cedula es menor que nodo->dato->cedula

Asignar a nodo->izq el resultado de llamar a insertarRecursivo con nodo->izq y dato

Sino Si dato->cedula es mayor que nodo->dato->cedula

Asignar a nodo->der el resultado de llamar a insertarRecursivo con nodo->der y dato

Fin Si

Devolver el nodo

Fin Función

// Método privado para realizar la eliminación recursiva de un nodo del árbol

Función eliminarRecursivo(nodo, cedula, padre)

Si el nodo es nulo

Devolver nulo

Fin Si

Si cedula es menor que nodo->dato->cedula

Asignar a nodo->izq el resultado de llamar a eliminarRecursivo con nodo->izq, cedula y nodo

Sino Si cedula es mayor que nodo->dato->cedula

Asignar a nodo->der el resultado de llamar a eliminarRecursivo con nodo->der, cedula y nodo

Sino // Se encontró el nodo a eliminar

Si nodo->izq es nulo

Devolver nodo->der // Conectar el nodo derecho con el padre del nodo a eliminar

Sino Si nodo->der es nulo

Devolver nodo->izq // Conectar el nodo izquierdo con el padre del nodo a eliminar

Sino // Nodo con dos hijos: obtener el sucesor inmediato

Crear un nuevo nodo temp y asignarle el valor mínimo del subárbol derecho de nodo

Asignar los datos de temp a nodo

Asignar a nodo->der el resultado de llamar a eliminarRecursivo con nodo->der, cedula del sucesor y nodo

Fin Si

Fin Si

Devolver el nodo

Fin Función

// Método privado para buscar un nodo específico dado su número de cédula

Función buscarNodo(nodo, cedula)

Si el nodo es nulo

Devolver nulo

Fin Si

Si nodo->dato->cedula es igual a cedula

Devolver el nodo

Fin Si

Buscar recursivamente en los subárboles izquierdo y derecho

Devolver el resultado de buscarNodo con nodo->izq, cedula

Devolver el resultado de buscarNodo con nodo->der, cedula

Fin Función

// Método privado para encontrar el nodo con el valor mínimo en un subárbol

Función minValorNodo(nodo)

Asignar el nodo actual a current

Mientras current no sea nulo y current->izq no sea nulo

Asignar a current el hijo izquierdo de current // Avanzar hacia el hijo izquierdo

Devolver current

Fin Función

// Método para imprimir el árbol en preorden

Función imprimirArbol()

Si la raíz es nula

Mostrar "El árbol está vacío."

Sino

Llamar a imprimirArbolRecursivo con la raíz, cadena vacía y verdadero

Fin Si

Fin Función

// Método privado para imprimir el árbol recursivamente en preorden

Función imprimirArbolRecursivo(nodo, prefijo, esIzquierda)

Si nodo->der no es nulo

Llamar a imprimirArbolRecursivo con nodo->der, prefijo concatenado con "│ " si esIzquierda es verdadero, sino con " ", falso

Fin Si

Mostrar prefijo concatenado con " └── " si esIzquierda es verdadero, sino con " ┌── ", nodo->dato->nombre, " (Cedula: ", nodo->dato->cedula, " - Programa Academico: ", nodo->dato->programa_academico, ")"

Si nodo->izq no es nulo

Llamar a imprimirArbolRecursivo con nodo->izq, prefijo concatenado con " " si esIzquierda es verdadero, sino con "│ ", verdadero

Fin Si

Fin Función

Fin Clase