|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

ARBOLES ABB

Algoritmos y Estructura de Datos

Profesor:

Roberto de Luna Caballero

Alumno:

Jiménez Velázquez José Bryan Omar

Grupo: 2CM5

**Introducción**

Los árboles de búsqueda binaria (ABB) son una estructura de datos fundamental en ciencias de la computación y matemáticas. Estos árboles se utilizan para organizar y almacenar datos de manera eficiente, permitiendo la búsqueda, inserción y eliminación rápida de elementos. Su diseño se basa en los principios de los árboles binarios, pero con la particularidad de que cumplen una propiedad adicional: el valor de cada nodo es mayor que todos los valores de sus subárboles izquierdos y menor que todos los valores de sus subárboles derechos.

La historia de los árboles de búsqueda binaria se remonta a la década de 1960, cuando el matemático y científico de la computación Adelson-Velsky y Landis desarrollaron esta estructura de datos en la antigua Unión Soviética. Su objetivo era encontrar una forma eficiente de organizar grandes volúmenes de información, de modo que las operaciones de búsqueda y manipulación fueran rápidas y eficientes. Desde entonces, los árboles de búsqueda binaria se han convertido en una herramienta ampliamente utilizada en la informática y han sido objeto de numerosos estudios e investigaciones.

La estructura de un árbol de búsqueda binaria consiste en una raíz, nodos internos y hojas. Cada nodo interno tiene dos hijos, uno a la izquierda y otro a la derecha, y contiene un valor único. Los nodos se organizan de manera jerárquica, de modo que cada nodo padre es mayor que su hijo izquierdo y menor que su hijo derecho. Esta propiedad permite una rápida búsqueda binaria: comenzando en la raíz, se compara el valor buscado con el valor del nodo actual y se continúa por el subárbol izquierdo o derecho según corresponda, hasta encontrar el elemento deseado o llegar a una hoja vacía.

La utilidad de los árboles de búsqueda binaria en la vida diaria es abundante. Una aplicación común de esta estructura es en los motores de búsqueda de internet, donde se almacenan y organizan grandes cantidades de información. Los motores de búsqueda utilizan árboles de búsqueda binaria para indexar y buscar páginas web de manera eficiente, permitiendo que los usuarios encuentren la información que buscan en cuestión de segundos.

Otra aplicación de los árboles de búsqueda binaria es en las bases de datos, donde se utilizan para indexar y buscar registros. Las bases de datos utilizan índices basados en árboles de búsqueda binaria para agilizar las consultas y las operaciones de inserción y eliminación de datos. Esto permite una gestión eficiente de grandes volúmenes de información en sistemas como bancos, empresas de comercio electrónico y redes sociales.

Los árboles de búsqueda binaria también se utilizan en la implementación de diccionarios y directorios telefónicos. Estas aplicaciones requieren una estructura de datos rápida y eficiente para buscar y almacenar información sobre palabras y números de teléfono, respectivamente. Los ABB proporcionan una solución ideal para estas aplicaciones, ya que permiten búsquedas rápidas y actualizaciones eficientes.

Además, los árboles de búsqueda binaria se utilizan en la implementación de algoritmos de ordenamiento como el ordenamiento rápido (quicksort).

Además de su papel en la búsqueda y recuperación de información, los árboles de búsqueda binaria también desempeñan un papel crucial en la implementación de algoritmos de ordenamiento, como el popular algoritmo de ordenamiento rápido (quicksort). El ordenamiento rápido se basa en la estrategia "divide y vencerás", donde la lista de elementos se divide en subconjuntos más pequeños utilizando árboles de búsqueda binaria. Esto permite una clasificación rápida y eficiente de los datos, ya que los elementos se comparan y reorganizan de acuerdo con su orden relativo en el árbol.

Además de su utilidad en la informática, los árboles de búsqueda binaria también tienen aplicaciones en otras áreas de la vida diaria. Por ejemplo, en la gestión de contactos y directorios telefónicos, los ABB proporcionan una forma eficiente de almacenar y buscar números de teléfono y otra información relacionada. Con un árbol de búsqueda binaria, es posible buscar rápidamente un número de teléfono específico, incluso en directorios telefónicos de gran tamaño, lo que facilita la comunicación y la gestión de contactos.

Otra aplicación práctica de los árboles de búsqueda binaria es en la representación de diccionarios y enciclopedias electrónicas. Estas herramientas necesitan una estructura de datos eficiente para almacenar y buscar palabras y sus definiciones. Al utilizar un árbol de búsqueda binaria, es posible acceder rápidamente a una palabra específica y obtener su significado correspondiente. Esto resulta especialmente útil en contextos educativos y de investigación, donde el acceso rápido a la información es fundamental.

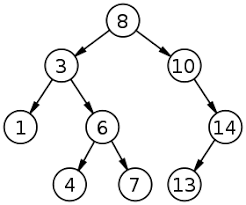
Además, los árboles de búsqueda binaria se utilizan en la implementación de sistemas de autocompletado en aplicaciones y sitios web. Al ingresar una consulta o una palabra clave, el sistema utiliza un árbol de búsqueda binaria para buscar coincidencias y proporcionar sugerencias instantáneas al usuario. Esto agiliza el proceso de búsqueda y mejora la experiencia del usuario, permitiéndole encontrar información relevante de manera más rápida y eficiente.

En el ámbito financiero, los árboles de búsqueda binaria se utilizan en el análisis y la gestión de datos de mercado. Estas estructuras de datos permiten una búsqueda rápida y eficiente de información financiera, como precios de acciones, tasas de cambio y datos de transacciones. Los árboles de búsqueda binaria se utilizan para organizar y acceder a grandes volúmenes de datos financieros, lo que ayuda a los analistas y operadores a tomar decisiones informadas y oportunas en los mercados.

Los árboles de búsqueda binaria son una herramienta esencial en la informática y tienen aplicaciones en numerosas áreas de la vida diaria. Desde la búsqueda de información en motores de búsqueda hasta la gestión de contactos en teléfonos móviles, pasando por el ordenamiento de datos y la representación de diccionarios, los ABB brindan una forma eficiente de organizar y acceder a grandes volúmenes de información. Su capacidad para agilizar las operaciones de búsqueda y manipulación de datos los convierte en una herramienta invaluable en el campo de la informática y en diversas industrias y disciplinas.

**Marco Teorico**

**Definición formal:**

Un Árbol Binario de Búsqueda (ABB) es una estructura de datos en forma de árbol en la cual cada nodo tiene a lo sumo dos hijos, y se cumple la propiedad de que para cada nodo, los valores almacenados en su subárbol izquierdo son menores o iguales que el valor almacenado en el nodo, y los valores almacenados en su subárbol derecho son mayores.

**Relación con la vida cotidiana:**

Una analogía de la vida cotidiana para entender un Árbol Binario de Búsqueda sería un directorio telefónico organizado alfabéticamente. Cada entrada en el directorio representa un nodo en el árbol, y contiene información como el nombre de una persona y su número de teléfono. Los nombres en el directorio están ordenados alfabéticamente, de modo que se cumple la propiedad de un ABB, donde los nombres en el subárbol izquierdo de un nodo son alfabéticamente menores o iguales al nombre del nodo, y los nombres en el subárbol derecho son alfabéticamente mayores.

Cuando buscamos un número de teléfono en el directorio, podemos utilizar la propiedad del ABB para realizar una búsqueda eficiente. Comenzamos en la raíz del árbol (la primera entrada en el directorio) y comparamos el nombre buscado con el nombre en la raíz. Si el nombre buscado es menor, sabemos que debe estar en el subárbol izquierdo y continuamos la búsqueda en ese subárbol. Si el nombre buscado es mayor, sabemos que debe estar en el subárbol derecho y continuamos la búsqueda allí. Repetimos este proceso hasta encontrar el nombre deseado y obtener el número de teléfono asociado.

De esta manera, los Árboles Binarios de Búsqueda nos permiten organizar y buscar eficientemente información en situaciones de la vida cotidiana, como un directorio telefónico, un diccionario o una lista de contactos.

**Inserción de un Árbol ABB**

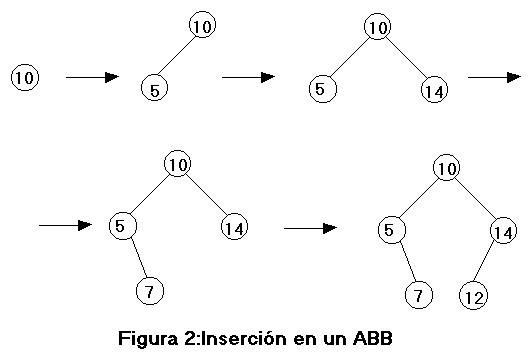
Un árbol de búsqueda binaria es una estructura de datos en la cual cada nodo tiene un valor único y se cumplen las siguientes propiedades:

* El valor en el nodo izquierdo es menor que el valor del nodo actual.
* El valor en el nodo derecho es mayor que el valor del nodo actual.
* Los subárboles izquierdo y derecho también son árboles de búsqueda binaria.
* La inserción en un ABB implica agregar un nuevo nodo con un valor determinado al árbol existente.

Aquí te explico los pasos para realizar la inserción:

* Comienza en la raíz del árbol y compara el valor del nuevo nodo con el valor del nodo actual.
* Si el valor del nuevo nodo es menor que el valor del nodo actual, se mueve al subárbol izquierdo. Si el subárbol izquierdo está vacío, el nuevo nodo se inserta como el hijo izquierdo del nodo actual. Si el subárbol izquierdo no está vacío, repite el paso 1 para ese subárbol.
* Si el valor del nuevo nodo es mayor que el valor del nodo actual, se mueve al subárbol derecho. Si el subárbol derecho está vacío, el nuevo nodo se inserta como el hijo derecho del nodo actual. Si el subárbol derecho no está vacío, repite el paso 1 para ese subárbol.
* Repite los pasos 1-3 hasta encontrar un lugar vacío adecuado para el nuevo nodo.
* Inserta el nuevo nodo en el lugar vacío encontrado en el paso anterior.
* Es importante tener en cuenta que la inserción en un ABB debe mantener las propiedades de búsqueda binaria para asegurar su correcto funcionamiento.

**EJEMPLO**



**Funcionamiento de la búsqueda en árboles ABB:**

La búsqueda en árboles ABB se realiza siguiendo un proceso de comparación y exploración recursiva, similar al algoritmo de búsqueda binaria. Aquí tienes una descripción general del proceso de búsqueda:

* Comenzamos en la raíz del árbol.
* Comparamos el valor buscado con el valor del nodo actual.
* Si son iguales, hemos encontrado el nodo deseado y finaliza la búsqueda.
* Si el valor buscado es menor, nos movemos al subárbol izquierdo y repetimos el proceso desde el paso 2.
* Si el valor buscado es mayor, nos movemos al subárbol derecho y repetimos el proceso desde el paso 2.
* Si llegamos a un nodo nulo (es decir, no hay más nodos que explorar), entonces el valor buscado no está presente en el árbol.
* Este proceso de comparación y exploración recursiva se repite hasta que se encuentre el valor deseado o hasta que se agoten los nodos disponibles para explorar.

**Reglas para mantener la propiedad del orden de búsqueda:**

Para garantizar que un árbol ABB mantenga la propiedad del orden de búsqueda, se deben seguir algunas reglas al insertar y eliminar nodos:

* Al insertar un nuevo nodo:
* Si el valor del nuevo nodo es menor o igual al valor del nodo actual, se debe insertar en el subárbol izquierdo.
* Si el valor del nuevo nodo es mayor que el valor del nodo actual, se debe insertar en el subárbol derecho.

**Al eliminar un nodo:**

* Si el nodo a eliminar es una hoja (no tiene hijos), simplemente se elimina.
* Si el nodo a eliminar tiene un hijo, ese hijo ocupa su lugar.
* Si el nodo a eliminar tiene dos hijos, se puede reemplazar por su sucesor inmediato (el valor más pequeño en su subárbol derecho) o por su predecesor inmediato (el valor más grande en su subárbol izquierdo). Este proceso garantiza que el árbol mantenga la propiedad del orden de búsqueda.

Al seguir estas reglas durante las operaciones de inserción y eliminación, el árbol ABB se mantiene balanceado y se puede realizar la búsqueda de manera eficiente.

**Eliminación de un Nodo**

La eliminación de un nodo en un árbol ABB puede ser un poco más compleja que la inserción. Aquí te explico cómo funciona el proceso de eliminación en un árbol ABB:

Encuentra el nodo a eliminar: Comienza por realizar una búsqueda en el árbol para encontrar el nodo que deseas eliminar. Si el nodo no se encuentra en el árbol, entonces no hay nada que eliminar.

* **Caso 1:** El nodo a eliminar es una hoja (no tiene hijos): En este caso, simplemente elimina el nodo del árbol. Puedes hacer esto eliminando el enlace desde su padre al nodo, es decir, establecer el enlace correspondiente (izquierdo o derecho) del padre a None o a null, dependiendo del lenguaje de programación que estés utilizando.
* **Caso 2:** El nodo a eliminar tiene un solo hijo: Si el nodo tiene solo un hijo, ya sea en el subárbol izquierdo o derecho, debes hacer que el hijo ocupe el lugar del nodo que estás eliminando. Para hacer esto, establece el enlace desde el padre del nodo a eliminar hacia el hijo, omitiendo el nodo que se va a eliminar. El hijo ahora se convierte en hijo directo del padre del nodo a eliminar.
* **Caso 3:** El nodo a eliminar tiene dos hijos: Este es el caso más complejo. Debes encontrar un sucesor o predecesor inmediato para reemplazar el nodo que se va a eliminar sin violar la propiedad del orden de búsqueda del árbol.

**Opción 1: Sucesor inmediato:** El sucesor inmediato de un nodo es el valor más pequeño en su subárbol derecho. Para encontrar el sucesor inmediato, ve al hijo derecho del nodo a eliminar y luego sigue moviéndote hacia la izquierda hasta encontrar el nodo más pequeño. El sucesor inmediato siempre será un nodo hoja o un nodo con un solo hijo en el subárbol derecho.

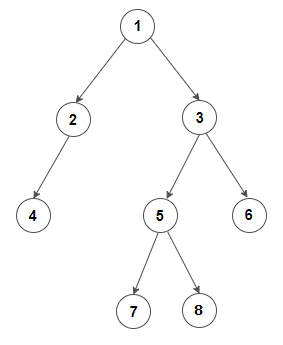
**Opción 2: Predecesor inmediato:** El predecesor inmediato de un nodo es el valor más grande en su subárbol izquierdo. Para encontrar el predecesor inmediato, ve al hijo izquierdo del nodo a eliminar y luego sigue moviéndote hacia la derecha hasta encontrar el nodo más grande. El predecesor inmediato también será un nodo hoja o un nodo con un solo hijo en el subárbol izquierdo.

Una vez que encuentres el sucesor o predecesor inmediato, reemplaza el valor del nodo a eliminar con el valor del sucesor o predecesor. Luego, repite el proceso de eliminación en el sucesor o predecesor inmediato para eliminar ese nodo. Recuerda que el sucesor o predecesor inmediato siempre será un nodo del Caso 1 o Caso 2, por lo que la eliminación en este caso se simplificará.

Es importante destacar que la eliminación de un nodo en un árbol ABB puede afectar el balance y la estructura del árbol. Para mantener un árbol ABB balanceado, existen algoritmos de balanceo, como el árbol AVL o el árbol rojo-negro, que se utilizan en implementaciones más avanzadas de árboles de búsqueda.

**Recorrido de un Árbol ABB**

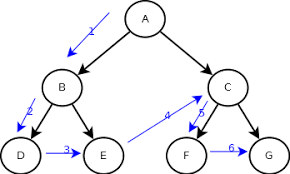
El recorrido de un árbol ABB es el proceso de visitar y procesar todos los nodos del árbol de manera sistemática. Hay tres tipos principales de recorrido en un árbol ABB: recorrido en orden (in-order), recorrido en preorden (pre-order) y recorrido en postorden (post-order). A continuación, te explico las reglas y cómo funciona cada uno de ellos:

**1. Recorrido en orden (in-order):**

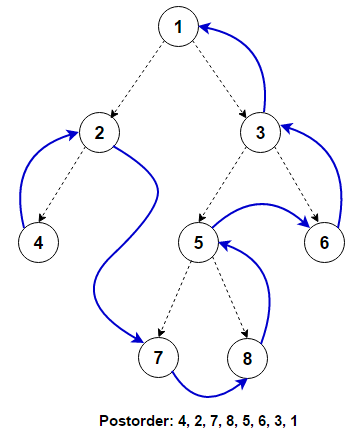
En el recorrido en orden, los nodos se visitan en el siguiente orden:

* Primero, se visita el subárbol izquierdo.
* Luego, se visita el nodo actual.
* Finalmente, se visita el subárbol derecho.
* El recorrido en orden produce una secuencia de nodos en orden ascendente si los valores almacenados en el árbol son numéricos o están ordenados.

**2. Recorrido en preorden (pre-order):**

En el recorrido en preorden, los nodos se visitan en el siguiente orden:

* Primero, se visita el nodo actual.
* Luego, se visita el subárbol izquierdo.
* Finalmente, se visita el subárbol derecho.
* El recorrido en preorden es útil para crear una copia exacta del árbol o para imprimir el árbol en forma de expresión algebraica.

**3. Recorrido en postorden (post-order):**

En el recorrido en postorden, los nodos se visitan en el siguiente orden:

* Primero, se visita el subárbol izquierdo.
* Luego, se visita el subárbol derecho.
* Finalmente, se visita el nodo actual.
* El recorrido en postorden es útil para liberar la memoria asignada a los nodos del árbol, ya que se visitan primero los nodos hoja y luego se avanza hacia arriba hasta la raíz.

**Funcionamiento del recorrido:**

El recorrido en un árbol ABB se realiza de forma recursiva para cada tipo de recorrido. Aquí te muestro cómo funciona el proceso:

**Para el recorrido en orden:**

* Realiza un recorrido en orden del subárbol izquierdo.
* Visita el nodo actual.
* Realiza un recorrido en orden del subárbol derecho.

**Para el recorrido en preorden:**

* Visita el nodo actual.
* Realiza un recorrido en preorden del subárbol izquierdo.
* Realiza un recorrido en preorden del subárbol derecho.

**Para el recorrido en postorden:**

* Realiza un recorrido en postorden del subárbol izquierdo.
* Realiza un recorrido en postorden del subárbol derecho.
* Visita el nodo actual.

En cada tipo de recorrido, la recursión se aplica primero al subárbol izquierdo, luego al subárbol derecho y finalmente al nodo actual. Esto garantiza que todos los nodos se visiten una vez en el orden correspondiente.

**Eficiencia de los arboles ABB**

Los árboles ABB (Árboles Binarios de Búsqueda) ofrecen una eficiencia considerable en varias operaciones. Aquí tienes un resumen de su eficiencia en términos de búsqueda, inserción y eliminación:

**Búsqueda:** La búsqueda en un árbol ABB tiene una eficiencia de tiempo promedio de O(log n), donde "n" es el número de nodos en el árbol. Esto se debe a que en cada paso de búsqueda, el número de nodos a considerar se reduce aproximadamente a la mitad. Sin embargo, en el peor de los casos, cuando el árbol está desequilibrado, la búsqueda puede tener una eficiencia de O(n), donde "n" es el número de nodos en el árbol, ya que se tendría que recorrer una rama lineal del árbol.

**Inserción:** La inserción en un árbol ABB también tiene una eficiencia de tiempo promedio de O(log n). Esto se debe a que durante la inserción, se busca la posición correcta para colocar el nuevo nodo siguiendo la propiedad del orden de búsqueda. Al mantener el árbol balanceado, se garantiza que la altura del árbol sea logarítmica con respecto al número de nodos. Sin embargo, si el árbol está desequilibrado, la inserción puede tener una eficiencia de O(n) en el peor de los casos, ya que se convertiría en una inserción lineal en una rama.

**Eliminación:** La eliminación en un árbol ABB tiene una eficiencia de tiempo promedio de O(log n). Durante la eliminación, se debe buscar el nodo a eliminar y luego realizar ajustes en la estructura del árbol para mantener la propiedad del orden de búsqueda. Al igual que en las operaciones anteriores, si el árbol está desequilibrado, la eliminación puede tener una eficiencia de O(n) en el peor de los casos.

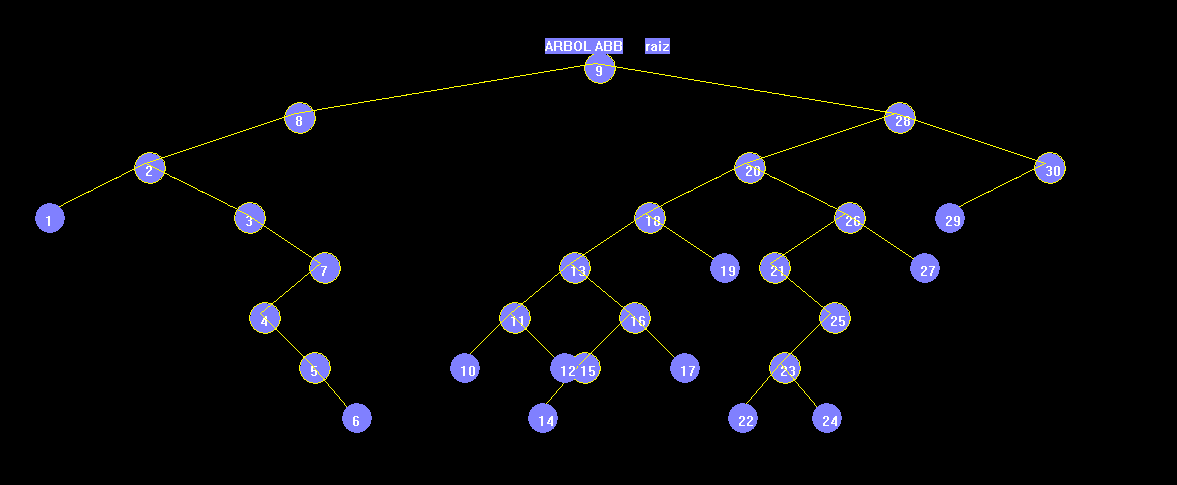
Es importante destacar que para mantener una eficiencia óptima en los árboles ABB, es necesario aplicar técnicas de balanceo, como los árboles AVL o los árboles rojo-negro, que garantizan una altura balanceada y mejoran la eficiencia de las operaciones.

**Ejemplo del árbol abb**

Nuestro Arreglos

1,2,8,3,7,4,5,6,9,10,11,12,14,15,13,16,17,18,19,20,26,21,25,23,22,24,27,28,30,29

**En este ejemplo de 30 datos, por practicidad, el llenado del árbol ABB se utilizo el llenado automático.**

****

**Desarrollo**

El fragmento de código que has mostrado es un programa en lenguaje C que incluye bibliotecas como `stdio.h`, `stdlib.h`, `time.h`, `graphics.h` y `math.h`. A continuación, te describo brevemente las funciones y estructuras de datos presentes en el código:

- `struct Nodo`: Define una estructura de datos llamada "Nodo" que tiene un campo `dato` de tipo entero y dos punteros `der` e `izq` que apuntan a nodos hijos derecho e izquierdo respectivamente.

- `CrearNodo`: Es una función que recibe un entero como parámetro y devuelve un puntero a un nodo recién creado. Esta función se utiliza para crear un nuevo nodo con el valor pasado como argumento.

- `insertar`: Es una función que inserta un nodo en un árbol binario de búsqueda. Recibe un puntero a la raíz del árbol, el valor a insertar y otros parámetros relacionados con la posición y tamaño de la ventana de gráficos. La función realiza la inserción del nodo en la posición correcta dentro del árbol de acuerdo con la propiedad del orden de búsqueda.

- `mostrar`: Es una función que muestra el contenido del árbol binario de búsqueda en una ventana gráfica. Recibe un puntero a la raíz del árbol y parámetros relacionados con la posición y tamaño de la ventana de gráficos.

- `busqueda`: Es una función que realiza la búsqueda de un valor en el árbol binario de búsqueda. Recibe un puntero a la raíz del árbol y el valor a buscar. Devuelve un valor booleano que indica si se encontró o no el valor en el árbol.

- `menu`: Es una función que muestra un menú interactivo para interactuar con el árbol binario de búsqueda. No se muestra la implementación completa de esta función en el fragmento de código proporcionado.

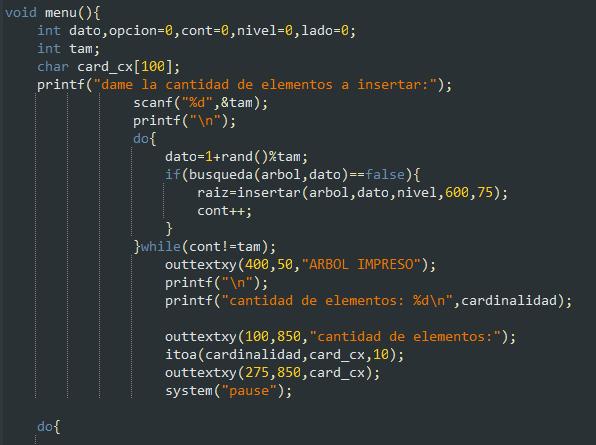
- `cardinalidad`: Variable entera que representa la cantidad de nodos en el árbol.

- `arbol`: Puntero a la raíz del árbol binario de búsqueda.

- `raiz`: Variable entera que representa el valor de la raíz del árbol binario de búsqueda.

Texto

Descripción generada automáticamente



La función menu() implementa un menú interactivo que permite al usuario realizar diferentes operaciones en el árbol binario de búsqueda. A continuación, se describen las acciones que se pueden realizar según las opciones del menú:

Rellenar: Esta opción está incompleta en el código proporcionado y no se muestra la implementación. Presumiblemente, se usaría para insertar elementos en el árbol binario de búsqueda. La cantidad de elementos a insertar se solicita al usuario utilizando la variable tam. Luego, se genera un número aleatorio (dato) y se verifica si ya existe en el árbol utilizando la función busqueda(). Si el número no está en el árbol, se inserta utilizando la función insertar(). El bucle continúa hasta que se hayan insertado tam elementos únicos en el árbol.

Mostrar: Esta opción muestra el contenido del árbol binario de búsqueda. Primero, se muestra un mensaje en la ventana gráfica indicando "ARBOL IMPRESO". Luego, se llama a la función mostrar() pasando el árbol, el nivel inicial, la posición x y la posición y iniciales. Esta función mostrará el contenido del árbol en la ventana gráfica. Después de mostrar el árbol, se imprime la cantidad de elementos en el árbol utilizando la variable cardinalidad. Además, se muestra la cantidad de elementos en la ventana gráfica utilizando las funciones outtextxy() y itoa().

Busqueda: Esta opción permite al usuario buscar un elemento en el árbol binario de búsqueda. Se solicita al usuario el elemento a buscar, que se almacena en la variable dato. Luego, se llama a la función busqueda() para verificar si el elemento está presente en el árbol. Dependiendo del resultado de la búsqueda, se imprime un mensaje indicando si el elemento ha sido encontrado o no.

Salir: Esta opción termina el bucle y sale del menú.

El menú se ejecuta en un bucle do-while que continúa hasta que el usuario selecciona la opción de salir (opción 4). Después de cada operación seleccionada, se limpia la pantalla con la función system("cls") para mostrar el menú nuevamente.

En general, la función menu() proporciona una interfaz interactiva para interactuar con el árbol binario de búsqueda, permitiendo al usuario insertar elementos, mostrar el contenido del árbol y buscar elementos específicos.

Texto

Descripción generada automáticamente

El fragmento de código proporcionado corresponde a la implementación de la función `CrearNodo()` en C. Esta función crea y devuelve un nuevo nodo para un árbol binario de búsqueda con el valor `n` especificado. A continuación, se explica paso a paso lo que hace el código:

1. Se declara una variable `nuevo\_nodo` de tipo `Nodo\*`, que es un puntero a la estructura de datos `Nodo`. Este puntero se utilizará para crear y almacenar el nuevo nodo.

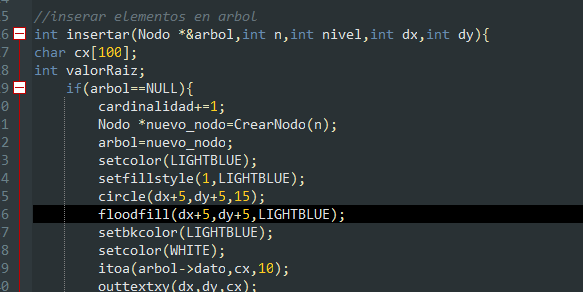
2. Se utiliza la función `malloc()` para asignar dinámicamente memoria para el nuevo nodo. La función `sizeof(struct Nodo)` se utiliza para determinar el tamaño necesario para almacenar un nodo.

3. Se asigna el valor `n` al campo `dato` del nuevo nodo utilizando el operador de acceso a miembros `->`. Esto establece el valor del nodo con el dato proporcionado.

4. Los campos `der` e `izq` del nuevo nodo se inicializan como `NULL`, lo que indica que no tienen hijos en este momento.

5. Finalmente, se devuelve el puntero `nuevo\_nodo` que apunta al nodo recién creado.

En resumen, esta función crea un nuevo nodo para un árbol binario de búsqueda y lo inicializa con el valor especificado. El nodo creado se asigna dinámicamente en memoria y se devuelve como resultado de la función.



La función `insertar()` implementa la inserción de un nuevo nodo con el valor `n` en un árbol binario de búsqueda. A continuación, se proporciona un resumen de lo que hace esta función:

1. Se declara una variable `cx` de tipo `char` que se utilizará para convertir el valor del nodo a una cadena de caracteres.

2. Se declara una variable `valorRaiz` de tipo `int` que almacenará el valor de la raíz del árbol. Esta variable se utiliza más adelante para realizar la comparación y determinar si el nuevo nodo debe insertarse en el subárbol izquierdo o derecho.

3. Si el árbol está vacío (es decir, `arbol` es igual a `NULL`), se crea un nuevo nodo con el valor `n` utilizando la función `CrearNodo()`. Luego, se asigna este nuevo nodo a `arbol`, lo que lo convierte en la raíz del árbol. Se incrementa la variable `cardinalidad` en 1 para llevar un conteo de la cantidad de nodos en el árbol.

4. Se establece el color y el relleno de la figura gráfica que representa el nodo en la ventana gráfica. Se dibuja un círculo con el centro en las coordenadas `(dx+5, dy+5)` y un radio de 15. El círculo se rellena con el color `LIGHTBLUE`. Luego, se establece el color de fondo como `LIGHTBLUE` y el color del texto como `WHITE`. Se convierte el valor del nodo a una cadena de caracteres utilizando la función `itoa()`. Finalmente, se muestra el valor del nodo en la ventana gráfica utilizando la función `outtextxy()`.

5. Si el árbol no está vacío, se compara el valor `n` con el valor de la raíz del árbol almacenado en `valorRaiz`. Si `n` es menor que `valorRaiz`, se establece el color como `YELLOW` y se dibuja una línea que conecta el nodo actual (`dx, dy`) con el nodo hijo izquierdo (`dx - (300 / (nivel + 1)), dy + 50`). Luego, se llama recursivamente a la función `insertar()` pasando el subárbol izquierdo (`arbol->izq`) como argumento, junto con `n`, `nivel + 1`, y las coordenadas actualizadas para dibujar el siguiente nodo en la ventana gráfica.

6. Si `n` es mayor o igual que `valorRaiz`, se establece el color como `YELLOW` y se dibuja una línea que conecta el nodo actual (`dx, dy`) con el nodo hijo derecho (`dx + (300 / (nivel + 1)), dy + 50`). Luego, se llama recursivamente a la función `insertar()` pasando el subárbol derecho (`arbol->der`) como argumento, junto con `n`, `nivel + 1`, y las coordenadas actualizadas para dibujar el siguiente nodo en la ventana gráfica.

7. Por último, se devuelve el valor de `valorRaiz`.

En resumen, la función `insertar()` realiza la inserción de un nuevo nodo en un árbol binario de búsqueda, actualizando la estructura del árbol y dibujando los nodos correspondientes en una ventana gráfica.

Texto

Descripción generada automáticamente

La función `mostrar()` se encarga de imprimir en la consola y en una ventana gráfica la representación gráfica de un árbol binario de búsqueda y sus nodos. A continuación, se proporciona un resumen de lo que hace esta función:

1. Se declara una variable `cx` de tipo `char` que se utilizará para convertir el valor del nodo a una cadena de caracteres.

2. Si el árbol está vacío (es decir, `arbol` es igual a `NULL`), la función retorna sin hacer nada.

3. Si el árbol no está vacío, se imprime en la consola el nivel y el valor del nodo actual utilizando las funciones `printf()`. Esto muestra la estructura jerárquica del árbol en la consola.

4. Se establece el color de las líneas que conectarán el nodo actual con sus hijos en la ventana gráfica como `YELLOW`. Si el nodo tiene un hijo izquierdo (`arbol->izq != NULL`), se dibuja una línea desde las coordenadas `(dx, dy)` hasta `(dx - (300 / (nivel + 1)), dy + 50)`. Si el nodo tiene un hijo derecho (`arbol->der != NULL`), se dibuja una línea desde `(dx, dy)` hasta `(dx + (300 / (nivel + 1)), dy + 50)`.

5. Se establece el color y el relleno de la figura gráfica que representa el nodo en la ventana gráfica. Se dibuja un círculo con el centro en las coordenadas `(dx + 5, dy + 5)` y un radio de 15. El círculo se rellena con el color `LIGHTBLUE`. Luego, se establece el color del texto como `WHITE`. Se convierte el valor del nodo a una cadena de caracteres utilizando la función `itoa()`. Finalmente, se muestra el valor del nodo en la ventana gráfica utilizando la función `outtextxy()`.

6. Se imprime en la consola las coordenadas `(dx, dy)` del nodo actual utilizando la función `printf()`.

7. Se realiza una llamada recursiva a la función `mostrar()` pasando el subárbol izquierdo (`arbol->izq`) como argumento, junto con el nivel incrementado en 1 y las coordenadas actualizadas `(dx - (300 / (nivel + 1)), dy + 50)`. Esto permite mostrar los nodos del subárbol izquierdo en la ventana gráfica.

8. Se realiza una llamada recursiva a la función `mostrar()` pasando el subárbol derecho (`arbol->der`) como argumento, junto con el nivel incrementado en 1 y las coordenadas actualizadas `(dx + (300 / (nivel + 1)), dy + 50)`. Esto permite mostrar los nodos del subárbol derecho en la ventana gráfica.

En resumen, la función `mostrar()` muestra la representación gráfica de un árbol binario de búsqueda en una ventana gráfica y proporciona información sobre los nodos y sus relaciones en la consola. Esto ayuda a visualizar y comprender la estructura del árbol.

**Conclusión**

Los árboles de búsqueda binaria (ABB) son una estructura de datos fundamental en ciencias de la computación y son ampliamente utilizados en una variedad de aplicaciones. A lo largo de este artículo, hemos explorado en detalle qué son los árboles ABB, cómo funcionan, las reglas que se deben seguir al trabajar con ellos, los recorridos que se pueden realizar en estos árboles, así como su eficiencia y algunas operaciones básicas como la inserción y eliminación de nodos.

En resumen, los árboles ABB son una estructura jerárquica en la que cada nodo tiene un valor asociado y dos subárboles, uno a la izquierda y otro a la derecha. Las reglas fundamentales de un árbol ABB son que el valor de cada nodo en el subárbol izquierdo debe ser menor que el valor del nodo padre, y el valor de cada nodo en el subárbol derecho debe ser mayor que el valor del nodo padre. Esto permite una búsqueda rápida y eficiente en el árbol, ya que se puede descartar la mitad del árbol en cada comparación.

El recorrido de un árbol ABB puede realizarse de varias formas, como el recorrido en orden (inorder), preorden (preorder) y postorden (postorder), que definen el orden en el que se visitan los nodos. Cada uno de estos recorridos tiene sus propias aplicaciones y puede ser útil según el contexto.

En cuanto a la eficiencia, los árboles ABB son una estructura de datos muy eficiente para la búsqueda, inserción y eliminación de elementos, siempre y cuando estén balanceados. Sin embargo, si el árbol no está balanceado, puede degenerar en una estructura lineal, lo que afectaría negativamente el rendimiento de las operaciones. Para evitar esto, existen técnicas de balanceo, como el árbol AVL y el árbol rojo-negro, que garantizan un equilibrio adecuado del árbol.

En términos de complejidad temporal, la búsqueda, inserción y eliminación en un árbol ABB balanceado tienen una complejidad promedio de O(log n), donde n es el número de nodos en el árbol. Esto se debe a que en cada paso se puede descartar aproximadamente la mitad de los nodos del árbol. Sin embargo, en el peor caso, si el árbol está desbalanceado, la complejidad puede llegar a ser O(n), convirtiéndose en una lista enlazada.

En conclusión, los árboles de búsqueda binaria (ABB) son una estructura de datos poderosa y versátil que se utiliza ampliamente en ciencias de la computación. Proporcionan una forma eficiente de almacenar, organizar y recuperar datos, especialmente cuando se busca un equilibrio entre el rendimiento y la complejidad. Sin embargo, es importante tener en cuenta el balance del árbol para garantizar su eficiencia óptima. Además, existen variantes de los árboles ABB, como el árbol AVL y el árbol rojo-negro, que garantizan un balance adecuado y mejoran aún más la eficiencia de las operaciones. Los árboles ABB son una herramienta poderosa

en el kit de herramientas de cualquier programador y comprender su funcionamiento y aplicaciones puede ser beneficioso para el desarrollo de software eficiente y optimizado.

**Bibliografia**

* Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.
* Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley Professional.
* Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in Python. Wiley.