ARBOLES

Índice

Introducción a los árboles en C++

Un árbol es una estructura de datos jerárquica que consiste en nodos conectados por aristas. Cada nodo contiene un valor y puede tener uno o más hijos. En particular, un árbol binario es un tipo de árbol en el que cada nodo tiene como máximo dos hijos, denominados hijo izquierdo e hijo derecho.

Conceptos básicos:

Nodo: Elemento que contiene un valor y referencias a sus nodos hijos.
Raíz: El nodo superior del árbol.
Hojas: Nodos que no tienen hijos.
Subárbol: Árbol formado por un nodo y todos sus descendientes.
Árboles binarios

Creación de un árbol binario

Para implementar un árbol binario en C++, primero debes definir la estructura de un nodo. Un nodo típico tiene un valor y punteros a sus hijos izquierdo y derecho

```
#include <iostream>

// Definición de la estructura del nodo
struct Nodo {
        int valor;
        Nodo* izquierdo;
        Nodo* derecho;

        // Constructor para inicializar el nodo
        Nodo(int val) : valor(val), izquierdo(nullptr), derecho(nullptr) {}
};

// Función para crear un nuevo nodo
Nodo* crearNodo(int valor) {
        return new Nodo(valor);
}
```

Insertar elementos en un árbol binario

La inserción de elementos en un árbol binario depende de cómo deseas organizar los datos. En un árbol binario de búsqueda **(BST)**, se insertan los valores siguiendo reglas específicas: *los valores menores se insertan en el subárbol izquierdo y los mayores en el subárbol derecho.*

Aquí te muestro cómo insertar elementos en un árbol binario simple:

```
void insertar(Nodo*& raiz, int valor) {
        if (raiz == nullptr) {
                raiz = crearNodo(valor);
        } else {
                if (valor < raiz->valor) {
                        insertar(raiz->izquierdo, valor);
                } else {
                        insertar(raiz->derecho, valor);
                }
        }
```

```
}
```

Recorrer un árbol binario

Hay tres formas comunes de recorrer un árbol binario:

Recorrido en preorden: Visita el nodo actual antes de sus hijos.
Recorrido en inorden: Visita el hijo izquierdo, luego el nodo actual, y después el hijo derecho.
Recorrido en postorden: Visita los hijos antes de visitar el nodo actual.

Aquí te muestro cómo implementar cada uno:

```
void recorrerPreorden(Nodo* nodo) {
        if (nodo != nullptr) {
                std::cout << nodo->valor << " ";
                recorrerPreorden(nodo->izquierdo);
                recorrerPreorden(nodo->derecho);
        }
}

void recorrerInorden(Nodo* nodo) {
        if (nodo != nullptr) {
                recorrerInorden(nodo->izquierdo);
                std::cout << nodo->valor << " ";
                recorrerInorden(nodo->derecho);
        }
}

void recorrerPostorden(Nodo* nodo) {
        if (nodo != nullptr) {
                recorrerPostorden(nodo->izquierdo);
                recorrerPostorden(nodo->derecho);
                std::cout << nodo->valor << " ";
        }
}
```

Buscar elementos en un árbol binario

La búsqueda en un árbol binario se puede realizar de manera recursiva, siguiendo las ramas del árbol hasta encontrar el valor deseado.

```
bool buscar(Nodo* raiz, int valor) {
        if (raiz == nullptr) {
                return false;
        }
        if (raiz->valor == valor) {
                return true;
        } else if (valor < raiz->valor) {
                return buscar(raiz->izquierdo, valor);
        } else {
                return buscar(raiz->derecho, valor);
        }
}
```

Eliminar elementos de un árbol binario

La eliminación en un árbol binario es más compleja, ya que involucra tres casos:

Eliminar un nodo sin hijos (nodo hoja): Simplemente se elimina el nodo.
Eliminar un nodo con un solo hijo: Se elimina el nodo y su hijo lo reemplaza.
Eliminar un nodo con dos hijos: Se reemplaza el nodo con el sucesor inorden (el menor nodo en el subárbol derecho).

Aquí tienes un ejemplo de cómo implementar la eliminación:

```
Nodo* encontrarMinimo(Nodo* nodo) {
        while (nodo->izquierdo != nullptr) {
                nodo = nodo->izquierdo;
        }
        return nodo;
}

Nodo* eliminar(Nodo* raiz, int valor) {
        if (raiz == nullptr) {
                return raiz;
        }
        if (valor < raiz->valor) {
                raiz->izquierdo = eliminar(raiz->izquierdo, valor);
        } else if (valor > raiz->valor) {
                raiz->derecho = eliminar(raiz->derecho, valor);
        } else {
                if (raiz->izquierdo == nullptr) {
                        Nodo* temp = raiz->derecho;
                        delete raiz;
                        return temp;
                } else if (raiz->derecho == nullptr) {
                        Nodo* temp = raiz->izquierdo;
                        delete raiz;
                        return temp;
                }

                Nodo* temp = encontrarMinimo(raiz->derecho);
                raiz->valor = temp->valor;
                raiz->derecho = eliminar(raiz->derecho, temp->valor);
        }
        return raiz;
}
```

Ejemplo completo de árbol binario

```
#include <iostream>

struct Nodo {
        int valor;
        Nodo* izquierdo;
        Nodo* derecho;

        Nodo(int val) : valor(val), izquierdo(nullptr), derecho(nullptr) {}
};

Nodo* crearNodo(int valor) {
        return new Nodo(valor);
}

void insertar(Nodo*& raiz, int valor) {
        if (raiz == nullptr) {
                raiz = crearNodo(valor);
        } else {
                if (valor < raiz->valor) {
                        insertar(raiz->izquierdo, valor);
                } else {
                        insertar(raiz->derecho, valor);
                }
        }
}

void recorrerInorden(Nodo* nodo) {
        if (nodo != nullptr) {
                recorrerInorden(nodo->izquierdo);
                std::cout << nodo->valor << " ";
```

```
                recorrerInorden(nodo->derecho);
        }
}

int main() {
        Nodo* raiz = nullptr;
        insertar(raiz, 50);
        insertar(raiz, 30);
        insertar(raiz, 70);
        insertar(raiz, 20);
        insertar(raiz, 40);
        insertar(raiz, 60);
        insertar(raiz, 80);

        std::cout << "Recorrido inorden del árbol: ";
        recorrerInorden(raiz);
        std::cout << std::endl;

        return 0;
}
```

Salida esperada:

Recorrido inorden del árbol: 20 30 40 50 60 70 80

Árboles binarios de búsqueda (BST)

Un árbol binario de búsqueda (BST) es un tipo especial de árbol binario donde:

El valor de cada nodo es mayor que todos los valores en su subárbol izquierdo.
El valor de cada nodo es menor que todos los valores en su subárbol derecho.

Los ejemplos anteriores ya muestran cómo crear, insertar, buscar y eliminar en un árbol binario de búsqueda. La principal ventaja de los BST es que permiten realizar estas operaciones en tiempo logarítmico en promedio, lo que los hace muy eficientes.

Consideraciones adicionales

Equilibrio: Los árboles binarios de búsqueda pueden desbalancearse, lo que degrada su rendimiento. Para evitar esto, se pueden utilizar versiones equilibradas como árboles AVL o árboles Rojo-Negro.

Complejidad: Las operaciones básicas en un árbol binario de búsqueda tienen una complejidad de O(log n) en promedio, pero en el peor de los casos (cuando el árbol está desbalance