Análisis y Algoritmos

Sergio Juan Díaz Carmona
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.sdiaz@uartesdigitales.edu.mx

Profesor: Efraín Padilla

Julio 10, 2019

1) Arbol Binario

un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No pueden tener más de dos hijos (de ahí el nombre ”binario”). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo. En el caso contrario el hijo es llamado un nodo interno. Usos comunes de los árboles binarios son los árboles binarios de búsqueda, los montículos binarios y Codificación de Huffman.

En teoría de grafos, se usa la siguiente definición: Un árbol binario es un grafo conexo, acíclico y no dirigido tal que el grado de cada vértice no es mayor a 3. De esta forma solo existe un camino entre un par de nodos.

Un árbol binario con enraizado es como un grafo que tiene uno de sus vértices, llamado raíz, de grado no mayor a 2. Con la raíz escogida, cada vértice tendrá un único padre, y nunca más de dos hijos. Si rehusamos el requerimiento de la conectividad, permitiendo múltiples componentes conectados en el grafo, llamaremos a esta última estructura un bosque'. Recorrido en inorden: En este caso se trata primero el subárbol izquierdo, después el nodo actual y por último el subárbol derecho. En un ABB este recorrido daría los valores de clave ordenados de menor a mayor. Otra forma para entender el recorrido con este método seria seguir el orden: nodo izquierda, nodo raíz, nodo derecha. En el árbol de la figura el recorrido en inorden sería: 2, 7, 5, 6, 11, 2, 5, 4, 9.

Codigo:
Insert:

```
void CBinaryTree::insert(node *tree, node *newnode)
{
        if (m_root == nullptr)
        {
                m_root = new node;
                m_root->info = newnode->info;
                m_root->left = nullptr;
                m_root->right = nullptr;
                cout << "Se agregado el padre" << endl;
                return;
        }
        if (tree->info == newnode->info)
        {
                cout << "Elemento ya existente" << endl;
                return;
        }
        if (tree->info > newnode->info)
        {
                if (tree->left != nullptr)
                {
                        insert(tree->left, newnode);
                }
                else
                {
```

```
                        tree->left = newnode;
                        (tree->left)->left = nullptr;
                        (tree->left)->right = nullptr;
                        cout << "Se a adio el nodo a la izquierda" << endl;
                        return;
                }
        }
        else
        {
                if (tree->right != nullptr)
                {
                        insert(tree->right, newnode);
                }
                else
                {
                        tree->right = newnode;
                        (tree->right)->left = nullptr;
                        (tree->right)->right = nullptr;
                        cout << "Se a adio el nodo a la derecha" << endl;
                        return;
                }
        }
}
```

Delete:

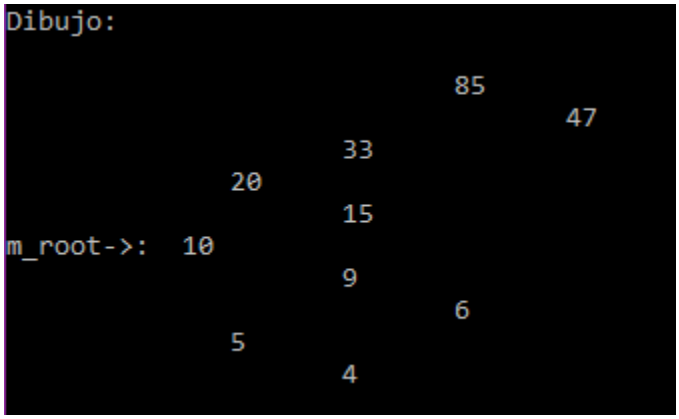
```
void CBinaryTree::Delete(int item)
{
        node *parent, *location;
        if (m_root == nullptr)
        {
                cout << "Tree vacio" << endl;
                return;
        }
        find(item, &parent, &location);
        if (location == nullptr)
        {
                cout << "No existe ese elemento" << endl;
                return;
        }
        if (location->left == nullptr && location->right == nullptr)
                case1(parent, location);
        if (location->left != nullptr && location->right == nullptr)
                case2(parent, location);
        if (location->left == nullptr && location->right != nullptr)
                case2(parent, location);
        if (location->left != nullptr && location->right != nullptr)
                case3(parent, location);
        free(location);
        cout << "El elemeneto : " <<item<<" se a borrado con exito"<< endl;
}

void CBinaryTree::case1(node *parent, node *location)
{
        if (parent == nullptr)
        {
                m_root = nullptr;
        }
```

```
        else
        {
                if (location == parent->left)
                        parent->left = nullptr;
                else
                        parent->right = nullptr;
        }
}

void CBinaryTree::case2(node *parent, node *location)
{

        node *child;
        if (location->left != nullptr)
                child = location->left;
        else
                child = location->right;
        if (location == nullptr)
        {
                m_root = child;
        }
        else
        {
                if (location == parent->left)
                        parent->left = child;
                else
                        parent->right = child;
        }
}

void CBinaryTree::case3(node *parent, node *location)
{
        node *ptr, *ptrsave, *suc, *parsuc;
        ptrsave = location;
        ptr = location->right;
        while (ptr->left != nullptr)
        {
                ptrsave = ptr;
                ptr = ptr->left;
        }
        suc = ptr;
        parsuc = ptrsave;
        if (suc->left == nullptr && suc->right == nullptr)
                case1(parsuc, suc);
        else
                case2(parsuc, suc);
        if (parent == nullptr)
        {
                m_root = suc;
        }
        else
        {
                if (location == parent->left)
                        parent->left = suc;
                else
                        parent->right = suc;
        }
        suc->left = location->left;
```

```
        suc->right = location->right;
}
```

InOrden:

```
void CBinaryTree::inorder(node *ptr)
{
        if (m_root == NULL)
        {
                cout << "Tree vacio" << endl;
                return;
        }
        if (ptr != NULL)
        {
                inorder(ptr->left);
                cout << ptr->info << "  ";
                inorder(ptr->right);
        }
}
```



Muestra de un arbol: