Análisis y Algoritmos

Luis Alberto Pineda Chavez
Universidad de Artes Digitales

Guadalajara, Jalisco

Email: idv17c.lpineda@uartesdigitales.edu.mx

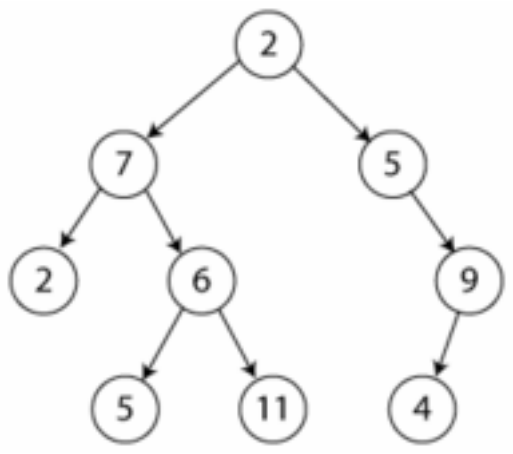
Profesor: Efraín Padilla

Mayo 23, 2019

1) Arbol binario de busqueda

Los arboles binarios son una estructura de datos formada de ”nodos”, cuentan con un puntero a su nodo ”padre” y dos a sus ”hijos” izquierdo y derecho.
La búsqueda en árboles binarios es un método de búsqueda simple, dinámico y eficiente, ya que acelera de forma considerable la velocidad de busqueda comparado con una estructura lineal.

Una de las mayores ventajas de esta estructura es que los datos ya ingresan al árbol de forma ordenada.



Una vez almacenados los datos, se pueden recorrer todos los nodos de tres distintas formas, ”pre order”, ”in order” y ”post order”, así como consultar o eliminar cualquiera de sus nodos.

Bajo estas lineas, se encuentra una implementación sencilla de un arbol en el cuál se pueden agregar y eliminar nodos, así como consultar sus valores y recorrer toda la estructura de las tres formas antes mencionadas.

```
void BinaryTree::Insert(Node*& node, Node* parent, float value) {
        if (!node) {
                node = new Node(parent, value);
        }
        else {
                if (value > node->Value) {
                        Insert(node->Right, node, value);
                }
                else {
                        Insert(node->Left, node, value);
                }
        }
}

bool BinaryTree::Search(Node* node, float value) {
        if (!node) {
                return false;
        }
        else if (node->Value == value) {
                return true;
        }
        else if (node->Value > value) {
                Search(node->Left, value);
        }
        else {
                Search(node->Right, value);
        }
}

bool BinaryTree::Erase(Node* node, float value) {
        if (!node) {
                return false;
        }
        else if (node->Value > value) {
                Erase(node->Left, value);
        }
        else if (node->Value < value) {
                Erase(node->Right, value);
        }
        else {
                EraseNode(node);
                return true;
        }
}

void BinaryTree::ReplaceNode(Node* node, Node* newNode) {
        if (node->Parent) {
                if (node->Parent->Left && node->Value == node->Parent->Left->Value) {
                        node->Parent->Left = newNode;
                }
                else if (node->Parent->Right && node->Value == node->Parent->Right->Value) {
                        node->Parent->Right = newNode;
                }
        }
        if (newNode) {
                newNode->Parent = node->Parent;
        }
}

void BinaryTree::EraseNode(Node* node) {
        if (node->Left && node->Right) {
                Node* min = Min(node->Right);
                node = min;
                EraseNode(min);
        }
        else if (node->Left) {
                ReplaceNode(node, node->Left);
                node->Left = nullptr;
                node->Right = nullptr;
                delete node;
        }
        else if (node->Right) {
                ReplaceNode(node, node->Right);
                node->Left = nullptr;
```

```
                node->Right = nullptr;
                delete node;
        }
        else {
                ReplaceNode(node, nullptr);
                node->Left = nullptr;
                node->Right = nullptr;
                delete node;
        }
}

Node* BinaryTree::Min(Node* node)
{
        if (!node) {
                return nullptr;
        }
        else if (node->Left) {
                return Min(node->Left);
        }
        else {
                return node;
        }
}

void BinaryTree::PrintPreOrder(Node* node) {
        if (!node) {
                return;
        }
        else {
                std::cout << node->Value << " - ";
                PrintPreOrder(node->Left);
                PrintPreOrder(node->Right);
        }
}

void BinaryTree::PrintInOrder(Node* node) {
        if (!node) {
                return;
        }
        else {
                PrintInOrder(node->Left);
                std::cout << node->Value << " - ";
                PrintInOrder(node->Right);
        }
}

void BinaryTree::PrintPostOrder(Node* node) {
        if (!node) {
                return;
        }
        else {
                PrintPostOrder(node->Left);
                PrintPostOrder(node->Right);
                std::cout << node->Value << " - ";
        }
}
```

REFERENCIAS

[1] Cormen, T., Leiserson, C., Rivest, R. and Stein, C. (2009). Introduction to algorithms. Cambridge (England): Mit Press.