ALGORITMOS Y ESTRUCTURAS DE DATOS 2023/1

PROFESOR: Manuel Alejandro Moscoso Domínguez
manuel.moscoso.d@gmail.com

Laboratorio Semana 10

En esta oportunidad realizaremos actividades de programación en C + + y el trabajo con algoritmos de búsqueda.

Objetivos

- Resolver ejercicios que involucren la implementación de funciones o clases en C + +.
- Desarrollar algoritmos asociados a árboles en C++.
- Desarrollar algoritmos que permitan entregar una solución a los problemas entregados.

Ejercicios

Nomenclatura para nombre de archivos fuentes

El nombre del archivo en el cual se almacena el código fuente debe considerar el siguiente formato: labsemanaX.EXT donde; X corresponde a la semana y EXT a la extensión del lenguaje de programación utilizado.

Ir a la ayuda memoría

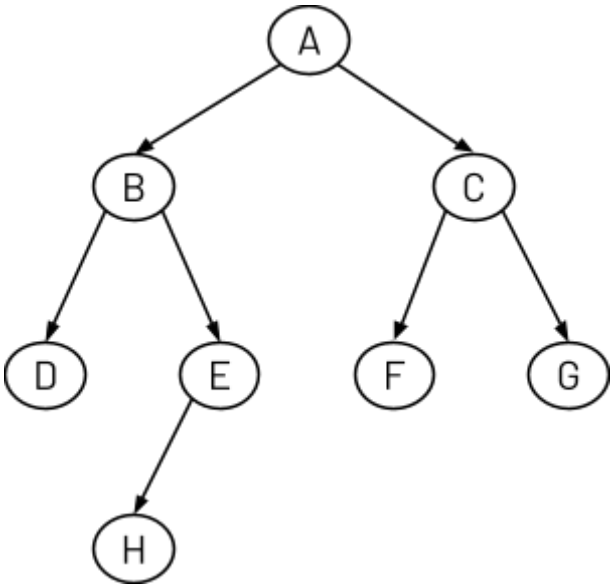
Recursive Traversals / Recorridos recursivos

Crear una estructura que permita trabajar con este tipo de árbol. El recorrido en árboles binarios involucra visitar cada nodo sólo una vez y podemos encontrar tres métodos comúnmente utilizados:

pre-order: Vista la raíz, recorre el subárbol de la izquierda en pre-order y luego recorre el subárbol de la derecha en pre-order.

in-order: Recorre el subárbol de la izquierda en in-order, luego la raíz y luego recorre el subárbol de la derecha en in-order.

post-order: Recorre el subárbol de la izquierda en post-order, recorre el subárbol de la derecha en in-order y luego la raíz.

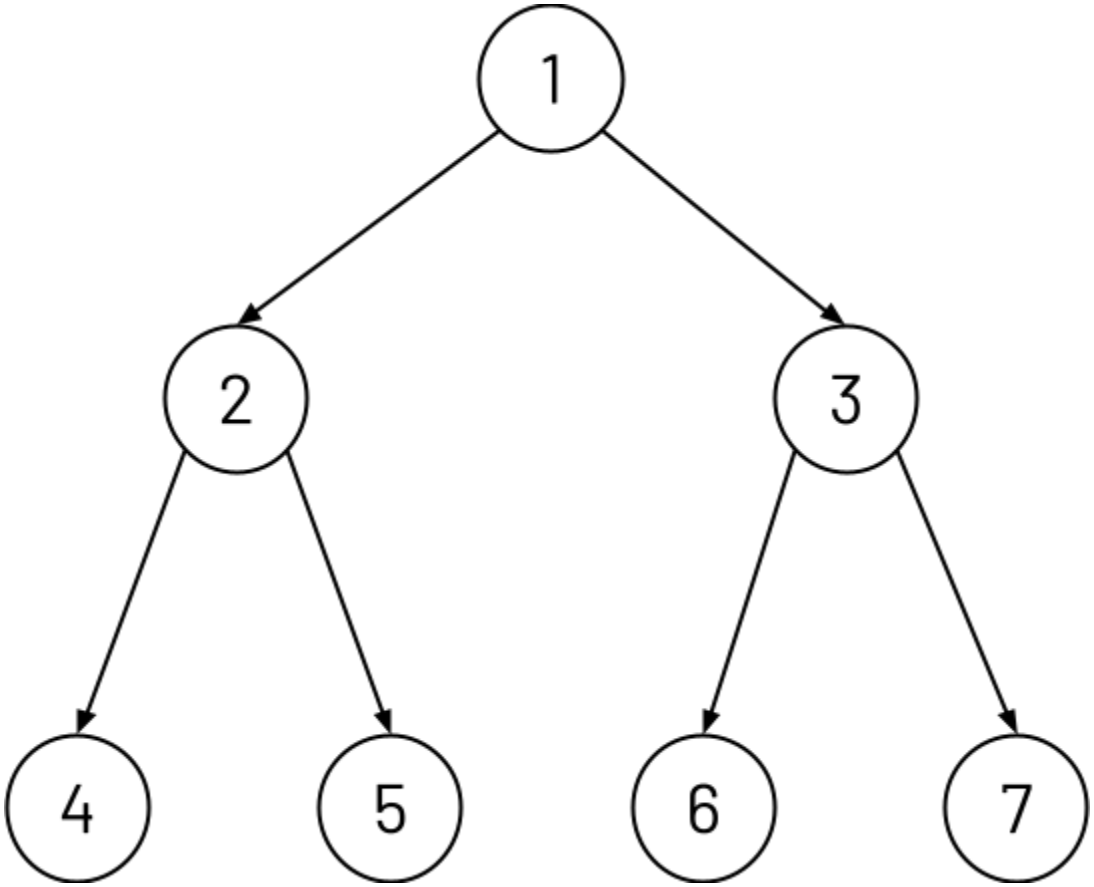


pre-order: A B D E H C F G

in-order: D B H E A F C G

post-order: D B H E F C G A

Tome de referencia el siguiente árbol binario



Ejercicio número 1

Crear una estructura que permita trabajar con este tipo de árbol y crear las instrucciones que permitan incorporar la información al árbol en el orden señalado en la imagen de referencia.

Ejercicio número 2

Crear una función que permite recorrer el árbol en pre-order. En este recorrido se debe visitar la raíz primero, luego el subárbol del lado izquierdo y posterior al subárbol del lado derecho.

Ejercicio número 3

Crear una función que permite recorrer el árbol en in-order. En este recorrido se debe visitar el subárbol del lazo izquierdo, la raíz y posterior al subárbol del lado derecho.

Ejercicio número 4

Crear una función que permite recorrer el árbol en postorden. En este recorrido se debe visitar el subárbol del lazo izquierdo, el subárbol del lado derecho y posterior la raíz.

Ejercicio número 5

Considere el arreglo entregado en este ejercicio para; implementar la funcionalidad para la creación del árbol binario y la selección del recorrido para que la impresión sea en orden.

Arreglo: 21, 18, 6, 9, 10, 7, 19, 15, 12, 5.

Ejercicio número 6

Crear una función que implemente la búsqueda por profundidad. Aquí se puede considerar la recursividad para realizar la implementación o la utilización de un stack.

Ejercicio número 7

Crear una función que implemente la búsqueda por amplitud. Aquí debe utilizar una cola o queue para realizar la implementación.

Ejercicios adicionales:

1. Cree una estructura para trabajar con un árbol general. Aquí se puede considerar el uso de vectores para gestionar los punteros a nodos “hijos”. Se sugiere considerar en la estructura más de un miembro o atributo para almacenar datos (no solo en int data).
2. Cree una función para crear el árbol.
3. Cree una función para insertar en el árbol.
4. Cree una función para recorrer el árbol.

Ayuda memoria

```cpp
#include <iostream>
using namespace std;

struct Node {
    int data;
    Node* left;
    Node* right;
};

Node* createNode(int data) {
    Node* newNode = new Node;
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
Node* deleteNode(Node* root, int key) {
    if (root == NULL) {
        return root;
    }

    if (key < root->data) {
        root->left = deleteNode(root->left, key);
    } else if (key > root->data) {
        root->right = deleteNode(root->right, key);
    } else {
        if (root->left == NULL) {
            Node* temp = root->right;
            delete root;
            return temp;
        } else if (root->right == NULL) {
            Node* temp = root->left;
            delete root;
            return temp;
        }

        Node* temp = root->right;
        while (temp->left != NULL) {
            temp = temp->left;
        }

        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }

    return root;
}
```