Ciencias computacionales

Propedéutico: Programación

Dra. Hayde Peregrina

Contents

1 Árboles **2**
1.1 Introducción 2
1.2 Características y Propiedades 2
1.3 Longitud de camino interno y externo 3
1.4 Árboles binarios 4
1.5 Recorrido de un árbol 6

1 Árboles

Árboles y su implementación en C++

1.1 Introducción

Los árboles son estructuras *no lineales* al contrario a estructuras *lineales* como lo son los arreglos y las listas. Además, los árboles reducen la complejidad (a un comportamiento logarítmico) en cuanto a operaciones como inserción y eliminación.

Los árboles muy utilizados debido a que son un método eficiente para búsquedas grandes y complejas por ejemplo en algunas áreas como lo son inteligencia artificial, algoritmos de cifrado y también son utilizados por los sistemas operativos para almacenar archivos.

1.2 Características y Propiedades

De manera general un árbol es una estructura que organiza los datos de modo que sus elementos estén relacionados entre sí por medio de ramas. El ejemplo más claro es el del árbol genealógico. Se les conoce como árboles ya que estos de manera gráfica pueden verse como un árbol invertido.

Un árbol consta de un número finito de elementos, a estos se les llama *nodos* estos nodos están unidos por líneas llamadas *ramas*.

Definición 1: Un árbol consta de un conjunto infinito de elementos llamados nodos, y un conjunto de líneas dirigidas llamadas ramas que conectan los nodos [1].

Definición 2: Un árbol es un conjunto de nodos tales que:

1. Existe un nodo raíz.
2. Los nodos restantes se dividen en $n \geq 0$ conjuntos disjuntos, $T_1 \dots T_n$ en donde cada uno de estos conjuntos es un árbol. A $T_1 \dots T_n$ se les denomina subárboles de la raíz.

La Fig. 1, muestra un árbol general.

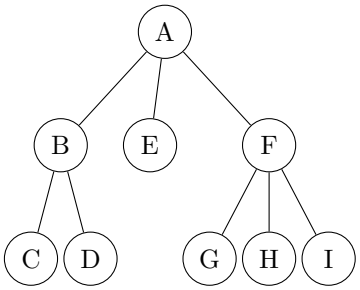

Figure 1: Árbol general.

Un nodo es padre si tiene sucesores a los cuales se les llama hijos. Tomemos como ejemplo el árbol que se muestra en la Fig. 2, el nodo **B** es padre de los hijos *E* y *F*. Los nodos *E, F, I,* y *J* son descendientes de *B*. Además, los nodos con el mismo padre se llaman hermanos y los nodos sin hijos se llaman nodos hoja.

La **altura** o **profundidad** de los árboles esta definida como el nivel de la hoja del camino más largo desde la raíz más uno. Por ejemplo la Fig. 3 muestra tres niveles: 0, 1, 2. Por lo tanto, la altura de este árbol es de 3.

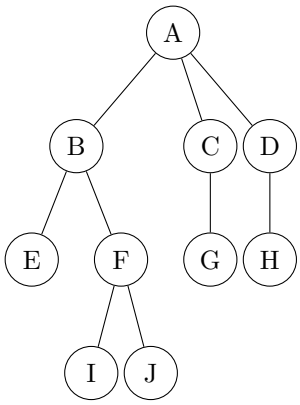

Figure 2: Ejemplo de árbol general.

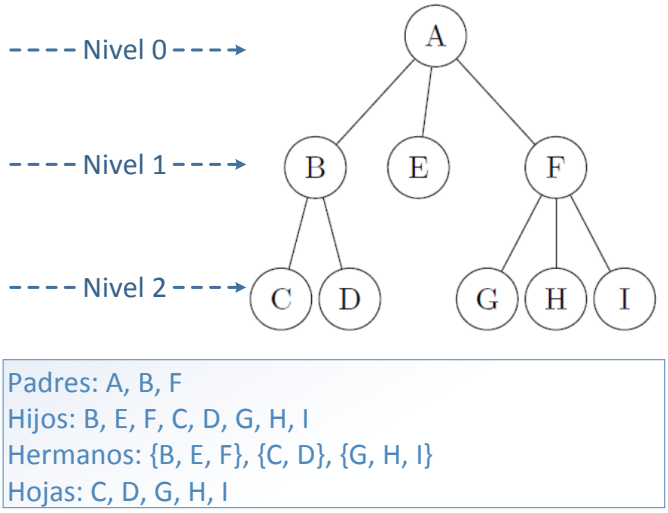

Figure 3: Terminología Árboles.

1.3 Longitud de camino interno y externo

Para poder definir los conceptos de camino interno y externo es necesario primero definir la longitud de camino. Se define como el número de arcos que deben ser recorridos para llegar desde la raíz al nodo destino. Por definición la raíz tiene longitud 1, sus descendientes directos longitud 2 y así sucesivamente.

La longitud de camino interno es la suma de las longitudes de camino de todos los nodos del árbol. Para el calculo de dicho se utiliza la siguiente formula: $LCI = \sum_{i=1}^{h} n_i * i$, donde i es el nivel del árbol, h es la altura y n_i representa el número de nodos del nivel i. Su media se calcula dividiendo entre el número de nodos del árbol. $LCIM = \frac{LCI}{n}$. Esto da el número promedio para llegar, desde la raíz a un nodo del árbol.

La longitud de camino externo Para poder definir el conceptos de camino externo es necesario primero definir lo que es un árbol extendido. Este es aquel en donde el número de hijos de cada nodo es igual al grado del árbol. De forma contraria se debe incorporar un nodo *especial* para cumplir la condición anterior.

Es decir:

- Los nodos *especiales* reemplazarían ramas vacías.

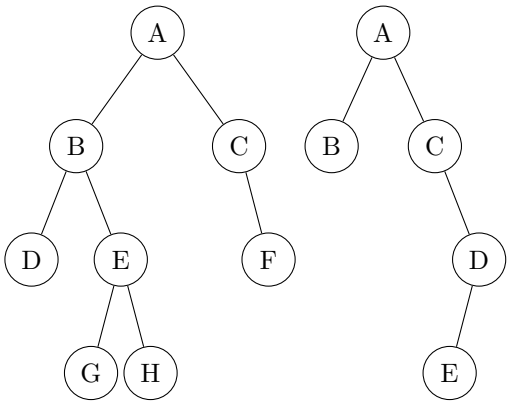

Figure 4: Árboles binarios.

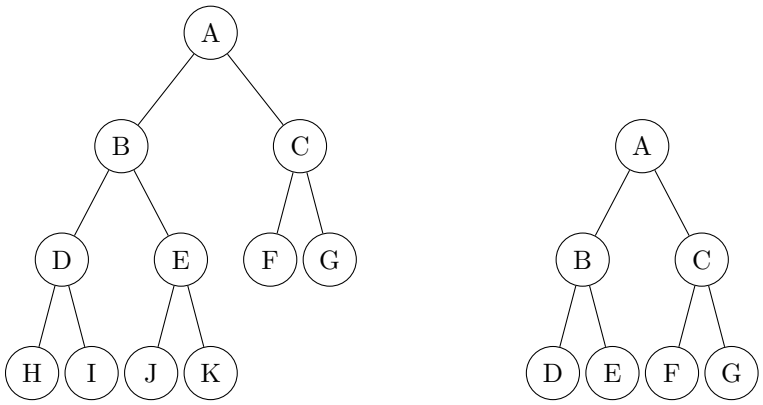

Figure 5: Árbol completo y Árbol lleno.

- De esta forma la **longitud de camino externo (LCE)** es la suma de las longitudes del camino de todos los nodos especiales del árbol.

Con la siguiente formula se calcula el camino externo, $LCE = \sum_{i=2}^{h} ne_i * i$, donde i representa el nivel del árbol, h la altura y ne_i el número de nodos especiales en el nivel i. Su media se calcula dividiendo entre el número de nodos especiales del árbol. $LCEM = \frac{LCE}{ne}$, lo cual da el número de arcos que deben ser recorridos en promedio, desde la raíz hasta un nodo especial cualquiera del árbol.

1.4 Árboles binarios

Se dice que un árbol binario es aquel en donde ningún nodo puede tener más de dos subárboles. Es decir, un árbol binario puede tener cero, uno o dos hijos y estos nodos se conocen como *hijo izquierdo* y como *hijo derecho*.

La Fig. 4, muestra ejemplos de árboles binarios. Se debe recordar que un árbol binario no puede tener más de dos subárboles. Esto se puede representar de manera recursiva, donde cada nodo es el nodo raíz de su propio subárbol y tiene hijos los subárboles derecho e izquierdo.

Árboles binarios completos

Un árbol **completo**, es un árbol que para cada nivel, desde el nivel 0 hasta el nivel $n-1$ tiene llenos los nodos izquierdo y derecho. Por otro lado, un árbol **lleno** tiene 2^n nodos a nivel n, es decir tiene el máximo número de entradas para su altura.

Representación en C de un nodo

Los árboles se pueden expresar mediante una estructura y de esta manera representar fácilmente a sus campos como lo son *dato*, rama izquierda y rama derecha. Además se puede generalizar el tipo de dato. A continuación se muestra un ejemplo:

```
#include <iostream>
#include <stdio.h>
using namespace std;

typedef int TypeofElement;    //Any type of element

typedef struct nodo{
        TypeofElement data;
        struct nodo *left, *right;
}Node;

typedef Node *BianryTree;
```

Creación de un árbol binario en C

Los árboles binarios se pueden construir recursivamente hasta llegar a las hojas de dicho. Por ejemplo, a partir de un nodo base se puede acceder a los demás nodos del árbol haciendo referencia a la raíz. Debido a que las ramas izquierda y derecha son árboles binarios que a su vez tienen raíz.

```
BinaryTree createNode (TipeofElement x){
        BinaryTree a;
        a = (BinaryTree) malloc(sizeof(Node)) ;
        a -> dato = x ;
        a -> right = a -> left = NULL;
        return a;
}
```

Para crear un nuevo árbol se puede utilizar la siguiente función:

```
void newTree (BinaryTree *root, BinaryTree leftbranch, TypeofElement x,
   BinaryTree *rightbranch){
        *root = createNode (x) ;
        (*root) -> left = leftbranch ;
        (*root) -> right = rightbranch ;
}
```

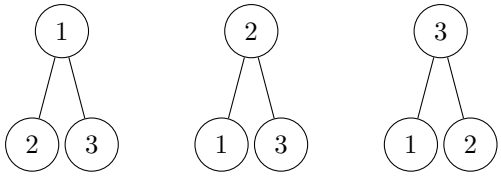

Figure 6: Preorden, Enorden y Postorden.

1.5 Recorrido de un árbol

Recorrido de árboles binarios

Recorrer el árbol no es otra cosa más que visitar o consultar los datos almacenados dentro del mismo, para esto es necesario visitar los nodos [2]. En los árboles sólo el primer nodo (raíz) es conocido, pero no se conoce quien viene a continuación.

Para recorrer un árbol existen dos secuencias: recorrido en *profundidad* y *recorrido* en anchura. En el recorrido en *profundidad* todos los descendientes de un hijo se procesan o visitan antes del siguiente hijo. En el recorrido en *anchura* cada nivel se procesa horizontalmente totalmente antes de comenzar el siguiente nivel.

Los tres tipos de secuencia de recorrido en profundidad se conocen como **en orden** (*inorder*), **preorden** (*preorder*) y **postorden** (*postorder*). La Fig. 6, muestra los recorridos y en que secuencia se deben recorrer según sea el caso.

A continuación se resumen los recorridos antes mencionados y se muestra un ejemplo en C.

Recorrido *en orden*:

1. Recorrer el subárbol izquierdo *en orden (inorder)*.
2. Visitar raíz.
3. Recprrer el subárbol derecho *en orden*.

```
1  #include <iostream>
2  #include <stdio.h>
3  using namespace std;
4
5  void inorder(NodoArbol *nodo) {
6          if (nodo != NULL) {
7                  inorder(nodo->izq);
8                  visitar(nodo);
9                  inorder(nodo->der);
10         }
11 }
```

Recorrido en postorden:

1. Recorrer el subárbol izquierdo en *postorden (postorder)*.
2. Recorrer el subárbol derecho en *postorder*.
3. Visitar raíz.

```
1  #include <iostream>
2  #include <stdio.h>
3  using namespace std;
4
5  void postorder(NodoArbol *nodo) {
6  if (nodo != NULL) {
```

```
7  postorder(nodo->izq);
8  postorder(nodo->der);
9  visitar(nodo);
10 }
11 }
```

References

[1] Alfred V Aho, John E Hopcroft, Jeffrey D Ullman, Américo Vargas Villazón, and Jorge Lozano Moreno. *Estructuras de datos y algoritmos*, volume 1. Addison-Wesley Iberoamericana, 1988.

[2] Noel Kalicharan. *Advanced Topics in Java: Core Concepts in Data Structures*. Apress, 2014.