ÁRBOLES BINARIOS

Algoritmos y Estructuras de Datos II

Lic. Ana María Company

QUÉ ES UN ÁRBOL?

- Es una generalización del concepto de ***lista***, donde se permite que cada registro del tipo de dato dinámico tenga más de un enlace.
- Los árboles son estructuras de datos recursivas más generales que una lista y son apropiados para aplicaciones que involucran algún tipo de jerarquía (ej: miembros de una familia, trabajadores de una organización), o de ramificación (como los árboles de juegos), o de clasificación y/o búsqueda.
- La definición recursiva de árbol es muy sencilla:
- ***Un árbol o es vacío o consiste en un nodo que contiene datos y punteros hacia otros árboles.***

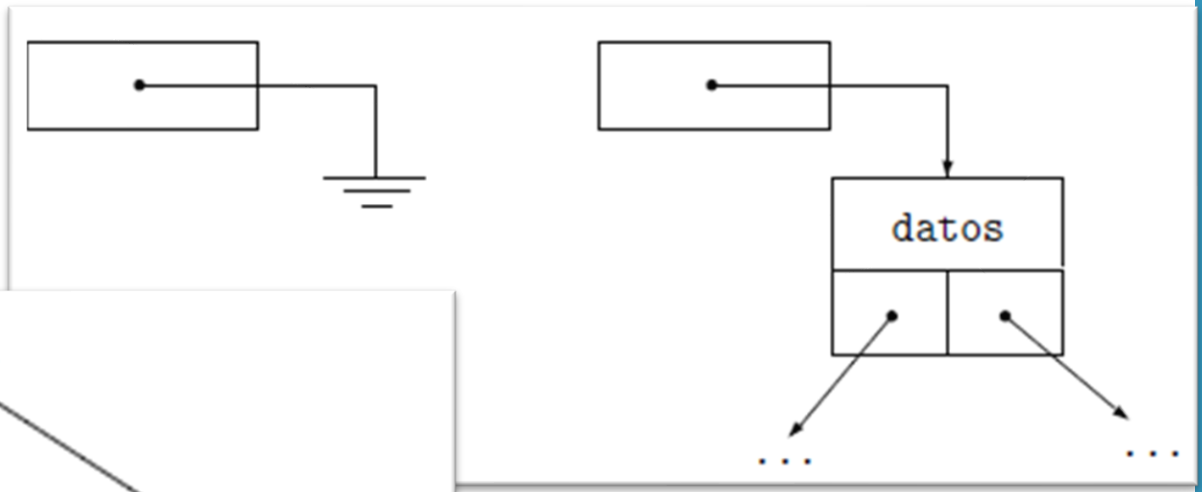
QUÉ ES UN ÁRBOL?

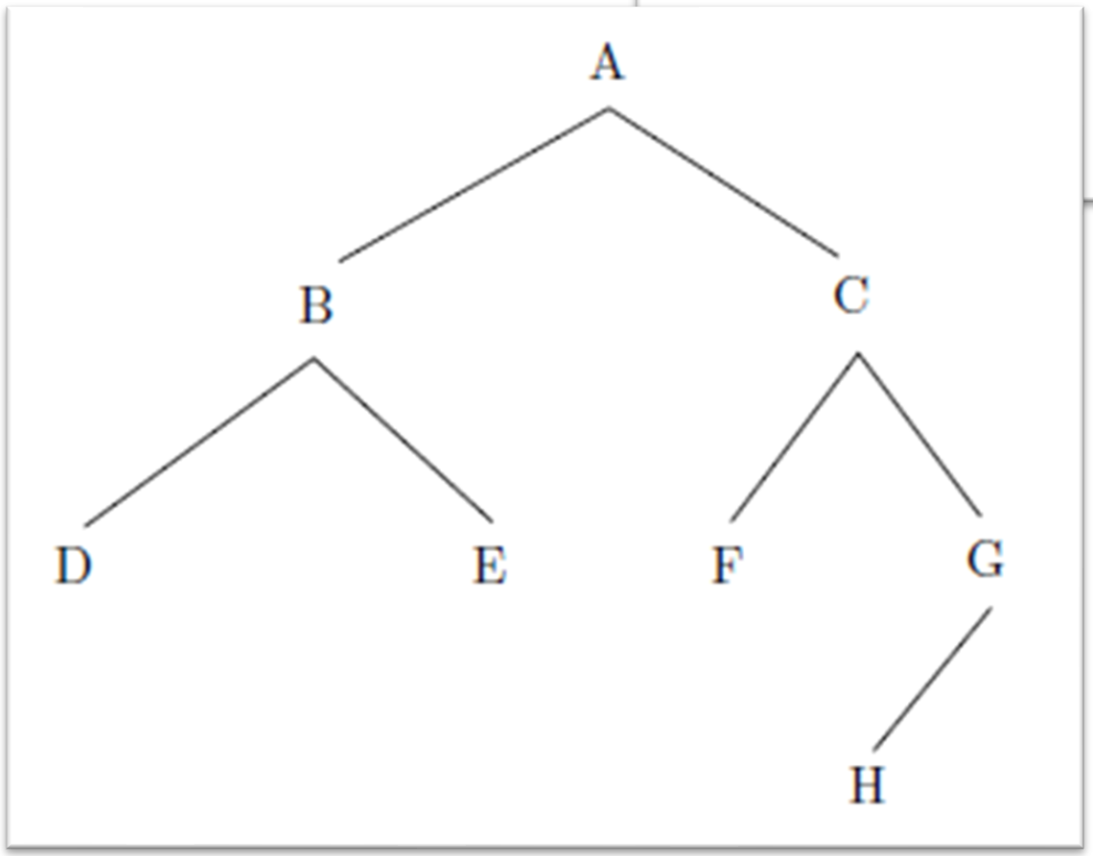
- Un árbol A es un conjunto finito de 1 o más nodos:
  - Existe un nodo especial denominado **RAIZ**(v_1) del árbol
  - Los nodos restantes (v_2, $v_{3, \ldots,} v_n$) se dividen en m >= o conjuntos disjuntos denominado A_1, $A_{2, \ldots,} A_m$, cada uno de los cuales, es a su vez un árbol, que se llaman ***subárboles del RAIZ***
- Un árbol con ningún nodo es un ***árbol nulo***: no tiene raíz.

ÁRBOL BINARIO

- Un árbol binario es un conjunto finito de cero o más nodos:
  - Existe un nodo denominado ***raíz*** del árbol.
  - Cada nodo puede tener 0, 1 ó 2 subárboles, conocidos como ***subárbol izquierdo*** y ***subárbol derecho***.
  - Es decir, cada nodo tiene a lo sumo dos descendientes.

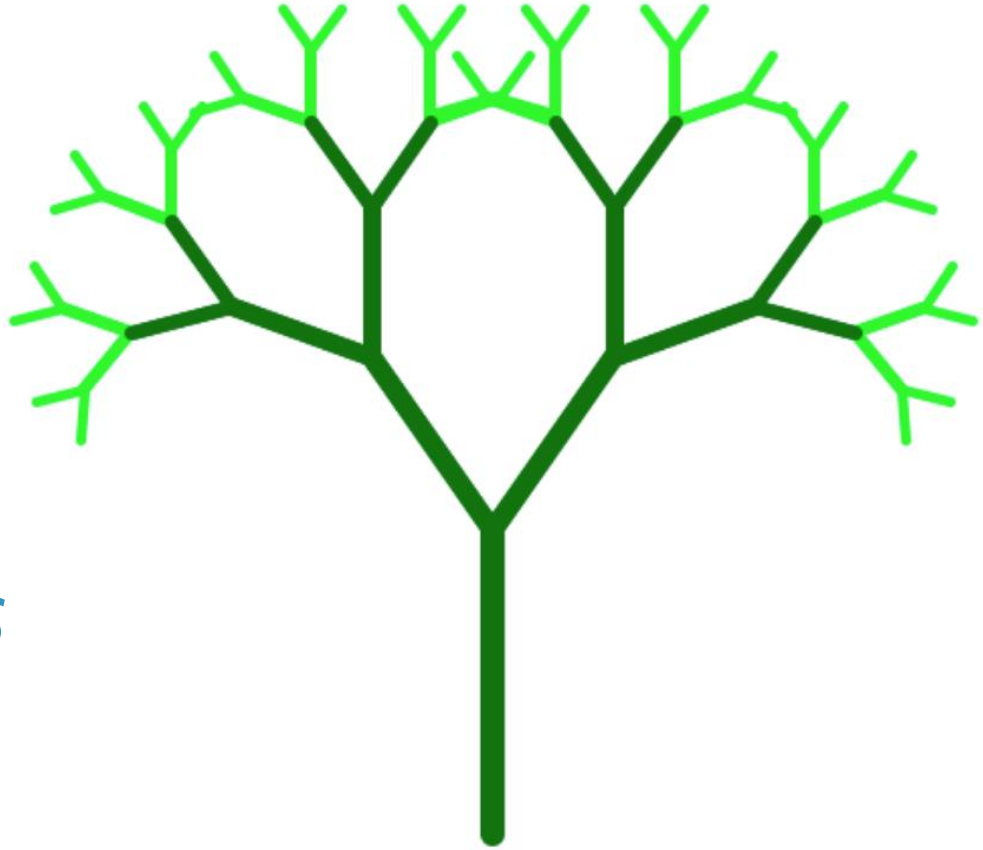
Árbol Binario - Gráficamente





Terminología

- *Raíz*
- *Hijos*
- *Hojas*
- *Ascendientes*
- *Descendientes*
- *Hermanos*
- *Padres*



Definición del tipo Árbol Binario

- Cada nodo del árbol posee dos punteros en lugar de uno:

```
typedef struct nodoArbol {
        int contenido;
        struct nodoArbol *hijoIzdo;
        struct nodoArbol *hijoDcho;
}tArbol;
```

RECORRIDO DE UN ÁRBOL BINARIO I

- Definir un algoritmo de recorrido de un árbol binario no es una tarea directa debido a que no es una estructura lineal, existen distintas formas de recorrerlo.
- A partir de un nodo podemos realizar alguna de las siguientes operaciones:
  - Leer el valor del nodo
  - Continuar por el hijo izquierdo
  - Continuar por el hijo derecho

- El orden en el que se efectúen las tres operaciones anteriores determinará el orden en el que los valores de los nodos del árbol son leídos.

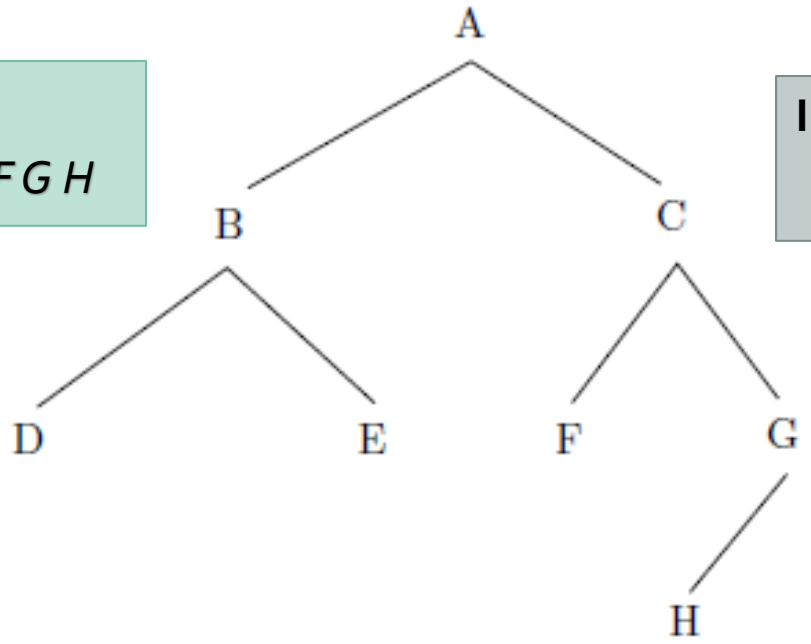
RECORRIDO DE UN ÁRBOL BINARIO II

- Si se acuerda que siempre se leerá primero el hijo izquierdo y después el derecho, existen tres maneras diferentes de recorrer un árbol:
- ***Preorden***: Primero se lee el valor del nodo y después se recorren los sub-árboles.
  - Esta forma de recorrer el árbol también recibe el nombre de recorrido ***Primero en Profundidad***.
- ***Inorden***: En este tipo de recorrido, primero se recorre el sub-árbol izquierdo, luego se lee el valor del nodo y, finalmente, se recorre el sub-árbol derecho.
- ***Postorden***: En este caso, se visitan primero los sub-árboles izquierdo y derecho y después se lee el valor del nodo.

Recorrido de un Árbol Binario - Gráficamente

Preorden:

A B D E C F G H



Inorden:

D B E A F C H G

Postorden:

D E B F H G C A

ÁRBOLES DE BÚSQUEDA

- Son un caso particular de *árbol binario*.
- Son llamados **Árboles binarios de búsqueda** o **Árboles de búsqueda binaria**.
- Son aquellos árboles en los que el valor de cualquier nodo es mayor que el valor de su hijo izquierdo y menor que el de su hijo derecho.
- Entonces, no puede haber dos nodos con el mismo valor en este tipo de árbol.

BÚSQUEDA DE UN NODO

La versión recursiva de la función es sencilla, todo consiste en partir de la raíz y rastrear el árbol en busca del nodo en cuestión.

- si *arbol* es vacío entonces
  - Devolver error
- en otro caso si *arbol->contenido* == *dato* entonces
  - Devolver el puntero a la raíz de *arbol*
- en otro caso si *arbol->contenido* **>** *dato* entonces
  - Buscar en el hijo izquierdo de *arbol*
- en otro caso si *arbol->contenido* **<** *dato* entonces
  - Buscar en el hijo derecho de *árbol*

INSERCIÓN DE UN NUEVO NODO

La inserción del nodo es tarea fácil, sólo debemos encontrar el lugar conveniente donde insertar el nuevo nodo, esto se hace en función de su valor de manera similar a lo visto en el ejemplo anterior.

En pseudocódigo:

- Si *arbol es vacío* entonces
  - crear nuevo nodo
- en otro caso si *árbol->contenido* **>** *datoNuevo* entonces
  - Insertar ordenadamente en el hijo izquierdo de *arbol*
- en otro caso si *arbol->contenido* **<** *datoNuevo* entonces
  - Insertar ordenadamente en el hijo derecho de *arbol*

- La forma resultante de un árbol binario de búsqueda depende bastante del orden en el que se vayan insertando los datos:
  - Si los datos ya están ordenados el árbol degenera en una lista.

ELIMINACIÓN DE UN NODO I

- La eliminación de un nodo en un árbol de búsqueda binaria no es una tarea sencilla a la hora de implementar.
- Tendremos en cuenta los siguientes casos según la condición del nodo que se desea eliminar:
  - Si el nodo es una ***hoja*** del árbol,
  - Un nodo con ***un sólo hijo***, o
  - Un nodo con ***dos hijos***.
- Los dos primeros casos no tienen mayor complejidad, sin embargo el tercer caso (un nodo con dos hijos) sí requiere un mayor nivel de detalle a la hora de su programación.

ELIMINACIÓN DE UN NODO II

- Primeramente se debe ubicar el ***nodo padre*** ***del nodo por eliminar***.
- Luego analizaremos cada caso en particular:
  1. Si el nodo a eliminar es una ***hoja***, entonces solo se debe destruir su variable asociada (haciendo uso del ***free***) y, luego asignar NULL a ese puntero.
  2. Si el nodo por eliminar ***sólo tiene un subárbol***, se usa la misma idea que al eliminar un nodo interior de una lista: "hay que saltarlo" conectando directamente el nodo anterior con el nodo posterior y desechando el nodo por eliminar.
  3. Si el nodo por eliminar tiene ***dos hijos*** no se puede aplicar la técnica anterior, porque entonces habrá dos nodos que conectar y no lograríamos un árbol binario. Por lo tanto, se tendrán que realizar los pasos necesarios para eliminar el nodo deseado y renovar las conexiones de modo que se siga teniendo un árbol de búsqueda.

ELIMINACIÓN DE UN NODO III

- Primero, se debe tener en cuenta que el nodo que se coloque en el lugar del nodo eliminado tiene que ser mayor que todos los elementos de su subárbol izquierdo.
- Luego se debe buscar ese nodo, que es el predecesor del nodo por eliminar, y se debe conocer en que posición se encuentra el nodo predecesor.
- Una vez hallado el predecesor el resto es tarea sencilla, sólo es preciso copiar su valor en el nodo por eliminar y descartar el nodo predecesor.

ELIMINACIÓN DE UN NODO - ALGORITMO

- Determinar el número de hijos del nodo N a eliminar
- si N no tiene hijos entonces
  - eliminarlo
- en otro caso si N sólo tiene un hijo H entonces
  - Conectar H con el padre de N
- en otro caso si N tiene dos hijos entonces
  - Buscar el predecesor de N
  - Copiar su valor en el nodo a eliminar
  - Desechar el nodo predecesor

BIBLIOGRAFÍA

- Mark Allen Weiss - Estructuras de Datos y Algoritmos - Florida International University - Año: 1995 - Editorial: Addison-Wesley Iberoamericana .
- Joyanes Aguilar, Luis - Programación en Pascal - 4ª Edición - Año: 2006 - Editorial: McGraw-Hill/Interamericana de España, S.A.U.
- Cristóbal Pareja Flores, Manuel Ojeda Aciego, Ángel Andeyro Quesada, Carlos Rossi Jiménez - Algoritmos y Programación en Pascal.
- Joyanes Aguilar, Luis - Fundamentos de la Programación. Algoritmos, Estructuras de Datos y Objetos - 3ª Edición - Editorial: McGraw-Hill