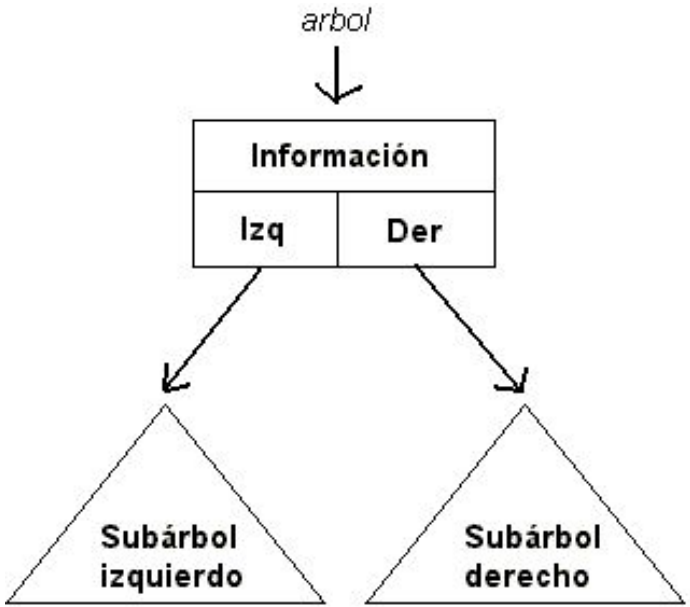
Estructuras de Datos
Árboles

Árbol Binario

Un árbol binario es un árbol en donde cada nodo posee exactamente *2 referencias* a subárboles. En general, dichas referencias se denominan izquierda y derecha, y consecuentemente se define el subárbol izquierdo y subárbol derecho del árbol.

Árbol Binario

La representación gráfica se vería así:



Árbol Binario

Yendo al código, podemos representar esta estructura de la siguiente forma:

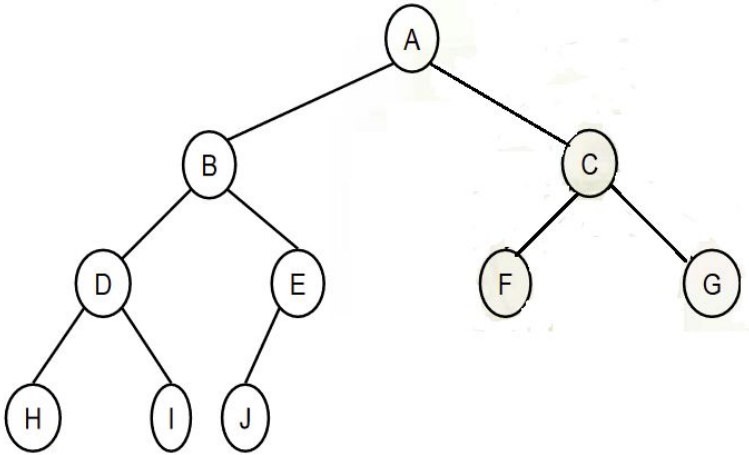
```
typedef struct _BTNodo {
  int dato;
  struct _BTNodo *left;
  struct _BTNodo *right;
} BTNodo;

typedef BTNodo *BTree;
```

vemos que, en este caso, la información a almacenar es un entero y, tengo dos punteros, uno al subárbol izquierdo y, otro al derecho.

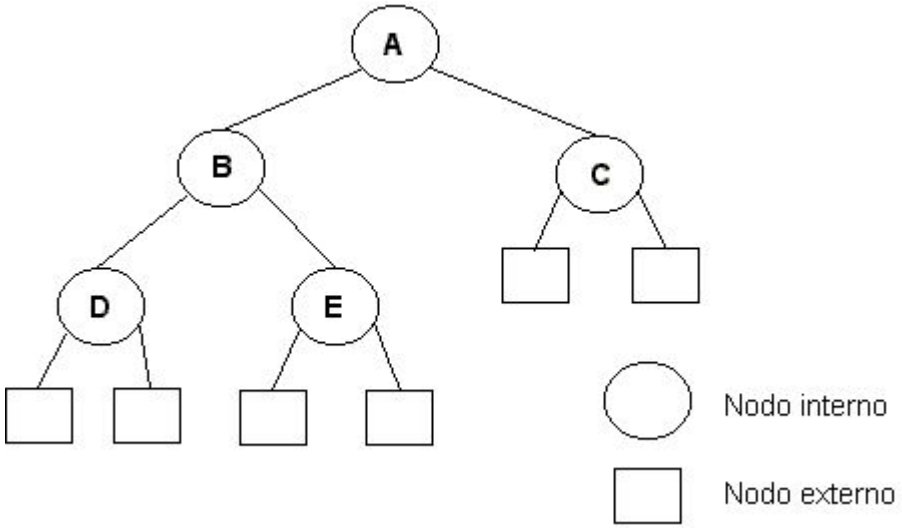
Árbol Binario

Un árbol binario de altura `k` está completo si está lleno hasta altura `k-1`, es decir, cada nodo tiene sus dos hijos y, el último nivel está ocupado de izquierda a derecha. Por ejemplo:



Árbol Binario

Los nodos en sí que conforman un árbol binario se denominan nodos internos, y todas las referencias que son *null* las llamamos nodos externos.



Árbol Binario - Propiedades

Podemos ver algunas propiedades de árboles binarios en base a esta definición:

1) Si se define i = número de nodos internos, e = número de nodos externos, entonces se tiene que:

$$e = i+1$$

Árbol Binario - Propiedades

Podemos ver algunas propiedades de árboles binarios en base a esta definición:

2) Sea *n = número de nodos internos*. Se define:

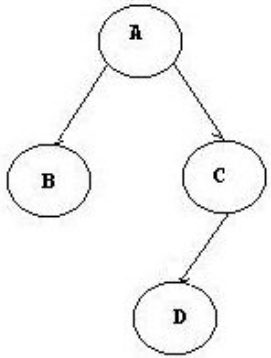
- I_n = suma del largo de los caminos desde la raíz a cada nodo interno (largo de caminos internos).
- E_n = suma del largo de los caminos desde la raíz a cada nodo externo (largo de caminos externos).

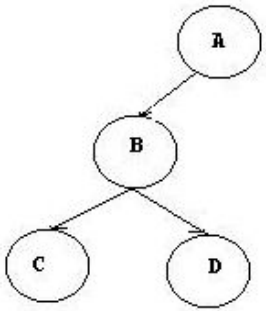
Se tiene que:

$$E_n = I_n + 2n$$

Árbol Binario - Propiedades

Diremos que dos árboles binarios son distintos si su estructura es distinta, es decir, si con la misma información existe, al menos, un nodo en diferente lugar.
En este caso podemos ver que el nodo C es hijo de A y en el árbol izquierdo y, es hijo de B en el árbol derecho.





Árbol Binario - Propiedades

Podemos ver algunas propiedades de árboles binarios en base a esta definición:

3) ¿Cuántos árboles binarios distintos (b_n) se pueden construir con *n* nodos internos? Para los primeros n números tenemos:

n	b_n
0	1
1	1
2	2
3	5

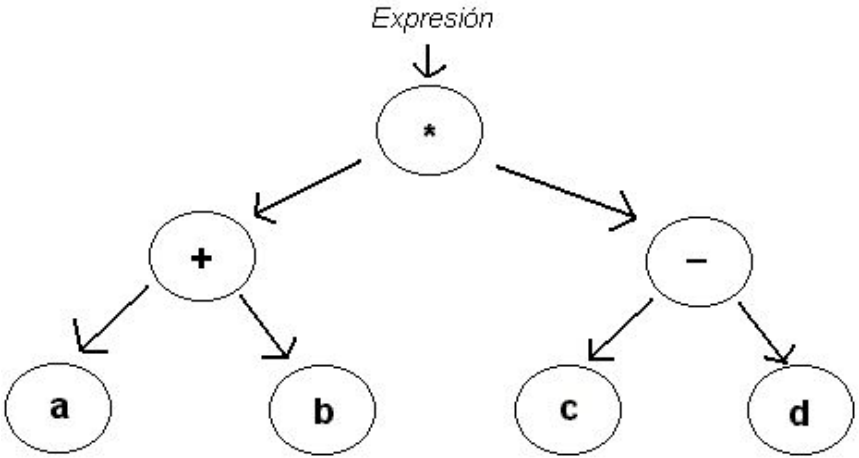
pero, generalizando, este número es: $$b_n = \frac{1}{n+1}\binom{2n}{n}$$

La serie de números que genera b_n se conoce como números de Catalan y converge a

$$b_n \approx \frac{4^n}{n\sqrt{\pi n}}$$

Árbol Binario - Árboles de Expresiones Aritméticas

Veremos un ejemplo de árboles binarios: los árboles de expresiones aritméticas.
En un árbol de expresiones las hojas corresponden a los *operandos* de la expresión (variables o constantes), mientras que los nodos restantes contienen *operadores*. Dado que los operadores matemáticos son binarios (o unarios como en el caso del operador negación), un árbol de expresiones resulta ser un árbol binario.



Árbol Binario - Árboles de Expresiones Aritméticas

Usando esta idea podemos construir un árbol para la expresión aritmética:

$$((7-a)/5)\times((a+b)\uparrow 3)$$

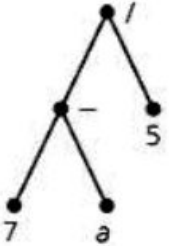
donde ↑ representa la potencia.

En este caso vamos a construir el árbol que representa la expresión de las hojas a la raíz.

Árbol Binario - Árboles de Expresiones Aritméticas

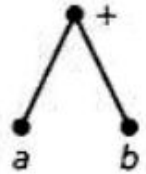
En primer lugar construimos un subárbol para la expresión `7-a`:

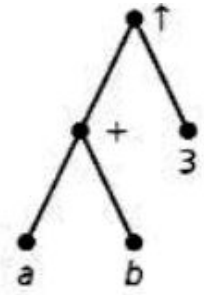
Una vez que lo tenemos, vamos a crear el subárbol de la expresión `(7-a)/5` que tiene al anterior como subárbol izquierdo.



Árbol Binario - Árboles de Expresiones Aritméticas

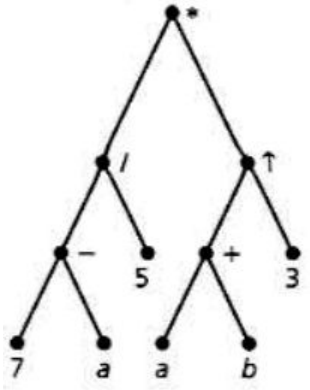
Luego, de modo similar, construimos los árboles binarios correspondientes a la expresión: (a+b)↑3.





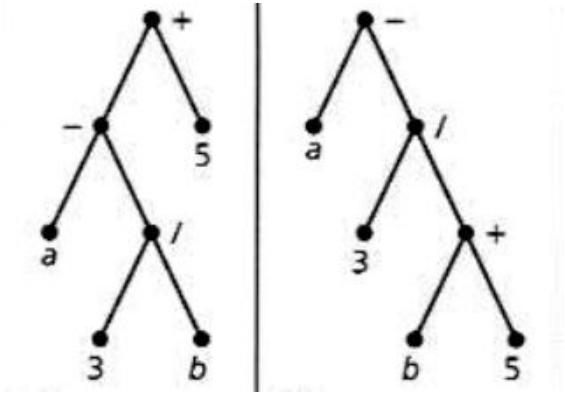
Árbol Binario - Árboles de Expresiones Aritméticas

Obteniendo, finalmente, el árbol de la expresión aritmética buscado:



Árbol Binario - Árboles de Expresiones Aritméticas

Los siguientes árboles binarios representan las expresiones: `(a-(3/b))+5` y `a-(3/(b+5))`



Árbol Binario - Recorridos

Un árbol de expresiones, se puede evaluar de la siguiente forma:

- Si la raíz del árbol es una constante o una variable se retorna el valor de ésta.
- Si la raíz resulta ser un operador, entonces recursivamente se evalúan los subárboles izquierdo y derecho, y se retorna el valor que resulta al operar los valores obtenidos de las evaluaciones de los subárboles con el operador respectivo.

Árbol Binario - Recorridos

Podemos ver que, al ser el árbol una estructura recursiva los algoritmos que se aplican sobre ellos suelen serlo también.

Esto es porque el algoritmo está escrito como una función que toma como dato la raíz del árbol que está procesando.

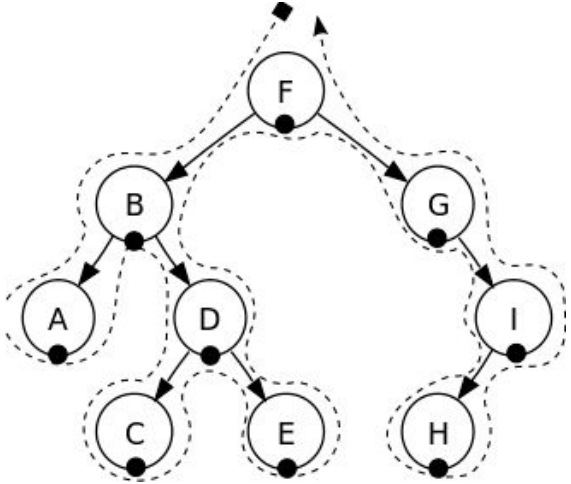
Árbol Binario - Recorridos

El primer algoritmo que vamos a ver para recorrer un árbol binario se denomina DFS (depth-first search) al que se llama Búsqueda en profundidad.

La idea de este recorrido es seguir un camino desde la raíz tan profundo como sea posible, cuando no es posible continuar este camino (por ejemplo cuando estamos en una hoja), se retrocede hasta el último nodo del camino en donde sea posible tomar una decisión diferente, y se avanza en ese otro sentido.

Árbol Binario - Recorridos

La siguiente imagen ilustra este recorrido.



Árbol Binario - Recorridos

Usando el recorrido podemos tener diferentes formas de visitar los vértices que forman un árbol binario. Las 3 formas usan DFS pero, lo que cambia en cada una de ellas es el momento en que se muestra la información del vértice en que estamos.

Debido a la expresión resultante obtenida es que estos recorridos se denominan: preorder, inorder y postorder.

A grandes rasgos, los mismos consisten en:

- *Preorden*: procesar la raíz - subárbol izquierdo - subárbol derecho.
- *Inorden*: subárbol izquierdo - procesar la raíz - subárbol derecho.
- *Postorden*: subárbol izquierdo - subárbol derecho - procesar la raíz.

La expresión que se obtiene con el recorrido en *postorden* se conoce como *notación polaca inversa*.

Árbol Binario - Recorridos

Algoritmo $preorder(s)$

Si s es vacío $Entonces$

$Retornar$

$Mostrar$ el valor de s

$L \coloneqq hijo\ izquierdo\ de\ s$

$preorder(L)$

$R \coloneqq hijo\ derecho\ de\ s$

$preorder(R)$

Árbol Binario - Recorridos

Ahora, analizaremos el algoritmo para algunos casos sencillos:

1) Si el árbol binario es vacío, nada se procesa pues, en este caso, el algoritmo termina simplemente en la línea 2.

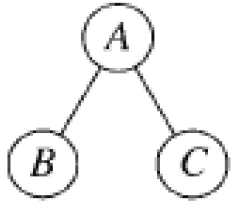
Árbol Binario - Recorridos

2) En este caso consideremos que ya contamos con un vértice. Es decir:

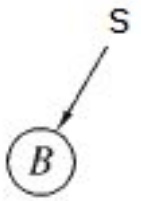
Supongamos que la entrada consta de un árbol con un único vértice. Hacemos s igual a la raíz y llamamos a $preorder(s)$. Como s no es vacío pasamos a la línea 3 donde mostramos el valor de la raíz. En la línea 5, llamamos a $preorder$ con s siendo el hijo izquierdo (vacío) de la raíz. Sin embargo hemos dicho que en la entrada de un árbol vacío $preorder$ no muestra información alguna. De manera análoga, en la línea 7 utilizamos un árbol vacío como entrada de $preorder$ y nada se procesa. Así cuando la entrada consta de un árbol con un único vértice, mostramos la raíz y terminamos.

Árbol Binario - Recorridos

3) Ahora, supongamos que la entrada es el siguiente árbol:



Hacemos s igual a la raíz y llamamos a $preorder(s)$. Como s no es vacío pasamos a la línea 3 donde mostramos el valor de la raíz (A). En la línea 5, llamamos a $preorder$ con s siendo el hijo izquierdo de la raíz:

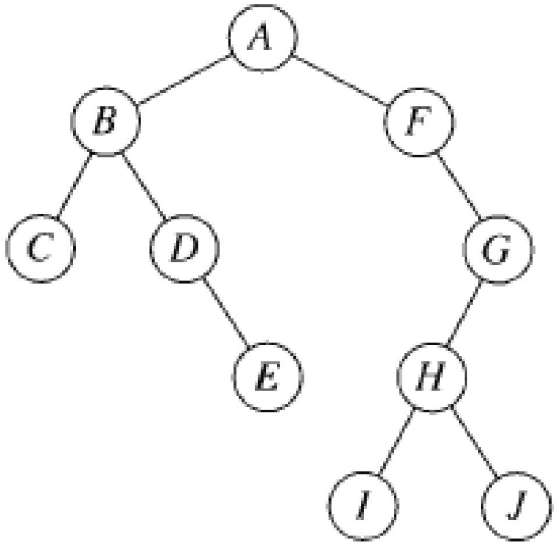


Acabamos de ver que si la entrada de $preorder$ consta de un único vértice, $preorder$ muestra ese vértice. Así, a continuación, mostramos el vértice B. De manera análoga en la línea 7, mostramos el vértice C.

Cuando el algoritmo termina, la salida obtenida es : ABC.

Árbol Binario - Recorridos

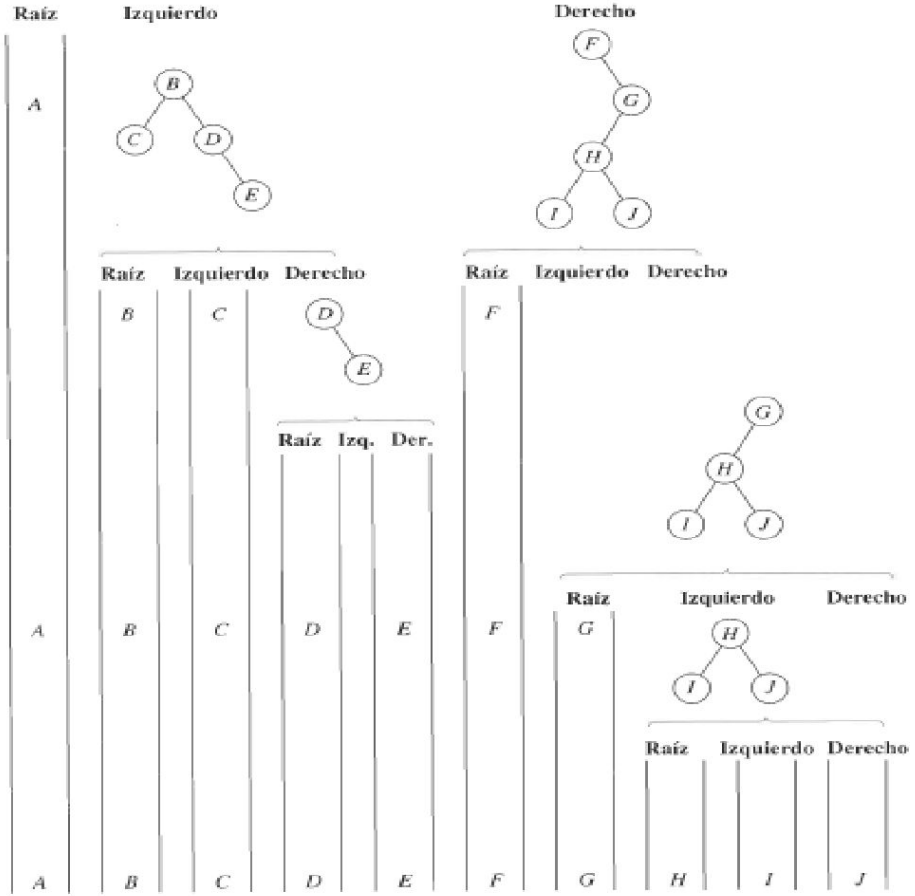
¿Cuál sería la salida del algoritmo $preorder$ con el siguiente árbol?



Árbol Binario - Recorridos

Si vamos siguiendo las líneas $3-7$ $(raíz/izquierda/derecha)$, del algoritmo $preorder$ el recorrido se muestra como indica la figura.

Así, la salida del algoritmo es : $ABCDEFGHIJ$.



Árbol Binario - Recorridos

Los recorridos $inorder$ y $postorder$ se obtienen solamente cambiando la posición de la línea 3 (raíz). Los prefijos pre, in y $post$ se refieren a la posición de la raíz en el recorrido; es decir, $preorder$ significa que primero va la raíz, $inorder$ quiere decir que la raíz va luego de mostrar el subárbol de la izquierda y, $postorder$ significa que la raíz va al final luego de mostrar el subárbol de la izquierda y, el de la derecha.

Árbol Binario - Recorridos

Algoritmo $inorder(s)$

Si s es vacío $Entonces$

$Retornar$

$L \coloneqq hijo\ izquierdo\ de\ s$

$inorder(L)$

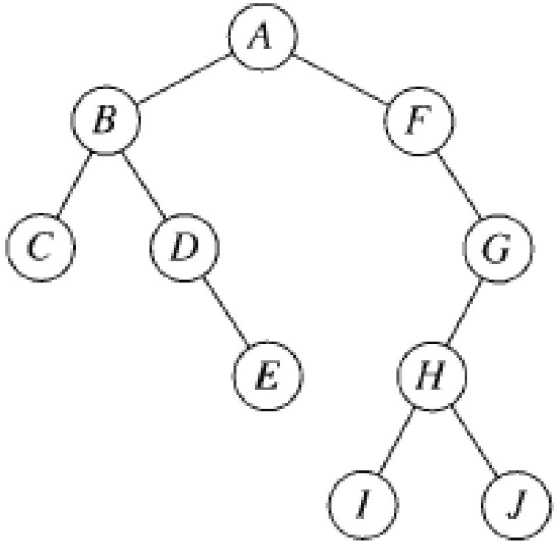
$Mostrar$ el valor de s

$R \coloneqq hijo\ derecho\ de\ s$

$inorder(R)$

Árbol Binario - Recorridos

¿Cuál sería la salida del algoritmo $inorder$ con el siguiente árbol?



Si vamos siguiendo las líneas $3 - 7$ $(izquierda/raíz/derecha)$, del algoritmo $inorder$ obtenemos como salida: $CBDEAFIHJG$.

Árbol Binario - Recorridos

Algoritmo $postorder(s)$

Si s es vacío $Entonces$

$Retornar$

$L \coloneqq hijo\ izquierdo\ de\ s$

$postorder(L)$

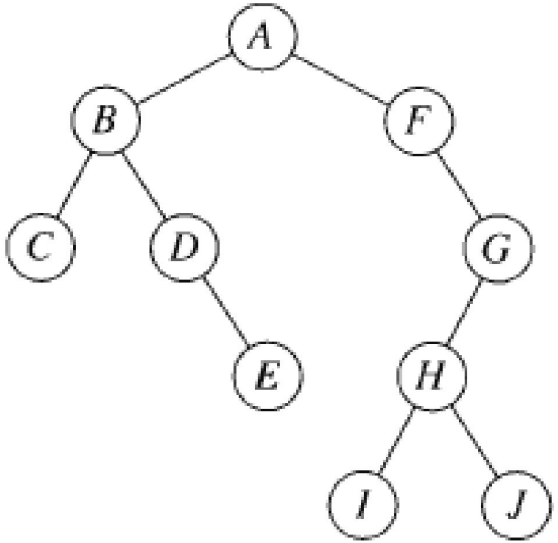
$R \coloneqq hijo\ derecho\ de\ s$

$postorder(R)$

$Mostrar$ el valor de s

Árbol Binario - Recorridos

¿Cuál sería la salida del algoritmo $postorder$ con el siguiente árbol?



Si vamos siguiendo las líneas $3-7$ $(izquierda/derecha/raíz)$, del algoritmo $postorder$ obtenemos como salida: $CEDBIJHGFA$.

Árbol Binario - Recorridos

Si volvemos al árbol del slide 10 cuando vimos expresiones aritméticas, y aplicamos los recorridos dados tendríamos:

en preorden

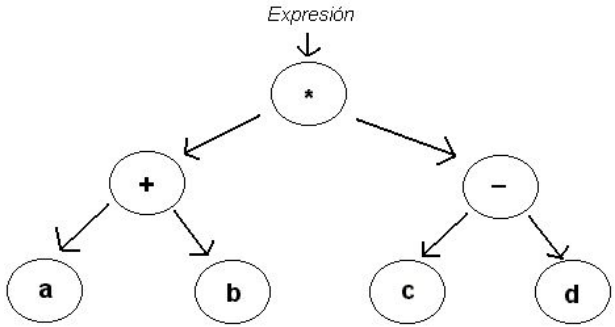
** + a b - c d*

en inorden

*a + b * c - d*

en postorden

*a b + c d - **

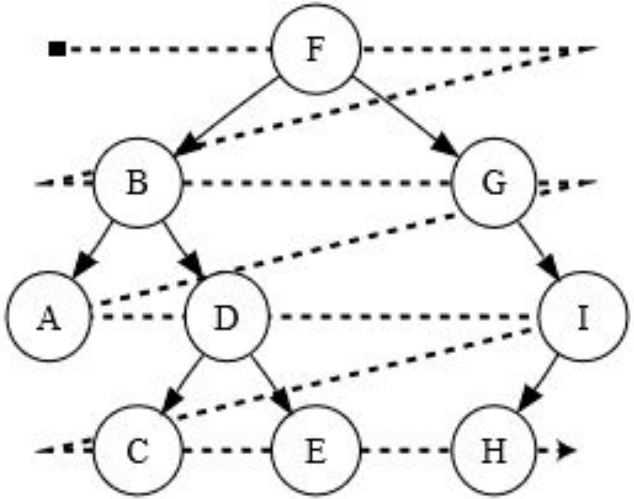


Árbol Binario - Recorridos

Otra forma de recorrer un árbol binario es usando el algoritmo BFS (Breadth-first search) el cual se conoce como Búsqueda por extensión o “a lo ancho”.
Es decir, se recorre por niveles.

Árbol Binario - Recorridos

El siguiente árbol muestra el recorrido BFS



Árbol Binario - Recorridos

Como podemos ver comenzamos desde la raíz y luego por niveles, de izquierda a derecha, se va recorriendo el árbol.