# GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: ÁRBOLES Y SUS RECORRIDOS

# 1. Introducción

En ciencias de la computación, el recorrido de árboles se refiere al proceso de visitar de una manera sistemática, exactamente una vez, cada nodo en una estructura de datos de árbol (examinando y/o actualizando los datos en los nodos). Tales recorridos están clasificados por el orden en el cual son visitados los nodos. De dichos recorridos abordará la siguiente guía.

# 2. Conocimientos previos

## 2.1. Árbol

Un árbol es un tipo abstracto de datos (TAD) ampliamente usado que imita la estructura jerárquica de un árbol, con un valor en la raíz y subárboles con un nodo padre, representado como un conjunto de nodos enlazados.

## 2.2. Árbol binario

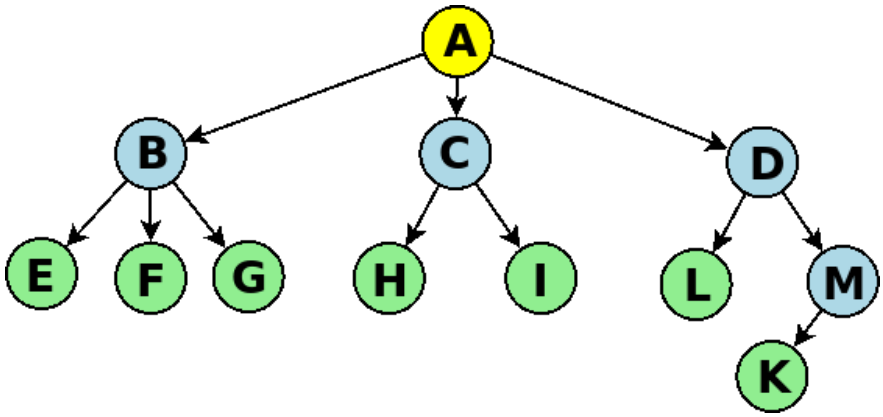
Un árbol binario es una estructura de datos en la cual cada nodo puede tener un hijo izquierdo y un hijo derecho. No puede n tener más de dos hijos (de ahí el nombre *binario*). Si algún hijo tiene como referencia a null, es decir que no almacena ningún dato, entonces este es llamado un nodo externo.

# 3. Desarrollo

Comparado a las estructuras de datos lineales como las listas enlazadas y arreglos unidimensionales, que tienen un método canónico de recorrido, las estructuras arborescentes pueden ser recorridas de muchas maneras diferentes. Comenzando en la raíz de un árbol, hay tres pasos principales que pueden ser realizados y el orden en la cual son realizados define el tipo de recorrido. Estos pasos (en ningún orden particular) son: ejecución de una acción en el nodo actual (referido como *visitando* el nodo), recorriendo al nodo hijo de la izquierda, y recorriendo al nodo hijo de la derecha. Así el proceso más fácilmente descrito a través de la recursión.

Los nombres dados para un estilo particular de recorrido vienen de la posición del elemento de raíz con respecto a los nodos izquierdo y derecho (si el árbol es binario, sino se asume como izquierdo el primer hijo y el resto de los hijos son derechos o viceversa todos los hijos son izquierdos menos el último que es el derecho.). Imagine que los nodos izquierdo y derecho son constantes en espacio, entonces el nodo raíz pudiera colocarse a la izquierda del nodo izquierdo (pre-orden), entre el nodo izquierdo y derecho (in-orden), o a la derecha del nodo derecho (post-orden).

Con el fin de ilustrar, se asume que los nodos izquierdos tienen siempre prioridad sobre los nodos derechos. Este ordenamiento puede ser invertido mientras el mismo orden sea asumido para todos los métodos de recorrido. Para explicar cada uno de los recorridos vamos a tomar como muestra el siguiente árbol:




Y para un mejor entendimiento asumiremos que los nodos hijos izquierdos serán todos menos el último hijo que será el derecho. De esta forma el nodo $B$ tiene dos hijos izquierdos ($E$,$F$) y un derecho ($G$).

## 3.1. Recorrido en profundidad-primero

### 3.1.1. Preorden

*(**raíz**, izquierdo, derecho)*. Para recorrer un árbol no vacío en Preorden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo raíz:

- Procesa la raíz
- Recorrer el subárbol izquierdo InOrden
- Recorrer el subárbol derecho InOrden

Si asumimos que el procesar la raiz es la impresión del valor almacenado en el nodo el recorrido en inorden en el árbol presentado de muestra anteriormente daría la siguiente secuencia o impresión de salida.

$$A, B, E, F, G, C, H, I, D, L, M, K$$

### 3.1.2. Inorden

*(izquierdo, **raíz**, derecho)*. El valor en un nodo no se procesa hasta que se procesen los valores en su subárbol izquierdo. Para recorrer un árbol no vacío en Inorden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo raíz:

- Recorrer el subárbol izquierdo InOrden
- Procesa la raíz
- Recorrer el subárbol derecho InOrden

Si asumimos que el procesar la raiz es la impresión del valor almacenado en el nodo el recorrido en inorden en el árbol presentado de muestra anteriormente daría la siguiente secuencia o impresión de salida.

$$E, F, B, G, H, C, I, A, L, D, K, M$$

#### 3.1.3. Postorden

*(izquierdo, derecho, **raíz**)*. Para recorrer un árbol binario no vacío en Postorden, hay que realizar las siguientes operaciones recursivamente en cada nodo, comenzando con el nodo raíz:

- Recorrer el subárbol izquierdo InOrden
- Recorrer el subárbol derecho InOrden
- Procesa la raíz

Si asumimos que el procesar la raiz es la impresión del valor almacenado en el nodo el recorrido en inorden en el árbol presentado de muestra anteriormente daría la siguiente secuencia o impresión de salida.

$$E, F, G, B, H, I, C, L, K, M, D, A$$

En general, la diferencia entre preorden, inorden y postorden es cuándo se recorre la raíz. En los tres, se recorre primero el sub-árbol izquierdo y luego el derecho.

- En preorden, la raíz se recorre antes que los recorridos de los subárboles izquierdo y derecho
- En inorden, la raíz se recorre entre los recorridos de los árboles izquierdo y derecho
- En postorden, la raíz se recorre después de los recorridos por el subárbol izquierdo y el derecho

Preorden (antes), inorden (en medio), postorden (después).

### 3.2. Recorrido en anchura-primero

Los árboles también pueden ser recorridos en orden por nivel (de nivel en nivel), donde visitamos cada nodo en un nivel antes de ir a un nivel inferior. Esto también es llamado recorrido en anchura-primero o recorrido en anchura. Se etiquetan los nodos según su profundidad (nivel). Se recorren ordenados de menor a mayor nivel, a igualdad de nivel se recorren de izquierda a derecha.

Si asumimos que el procesar la raiz es la impresión del valor almacenado en el nodo el recorrido en anchura-primero en el árbol presentado de muestra anteriormente daría la siguiente secuencia o impresión de salida.

$$A, B, C, D, E, F, G, H, I, L, M, K$$

# 4. Implementación

La implementaciones de los recorridos va depender en gran medida del tipo de árbol y de como fue implementado por lo que a continuación se van a presentar ideas generales de la implementación de estos recorridos asumiendo que estamos trabajando con árboles binarios.

## 4.1. Recorrido en profundidad-primero

### 4.1.1. Preorden

```
//Variante recursiva
preorden(nodo)
   si nodo == nulo entonces retorna
   imprime nodo.valor
   preorden(nodo.izquierda)
   preorden(nodo.derecha)

//Variante iterativa
iterativePreorder(node)
   if (node = null)
      return
   s = empty stack
   s.push(node)
   while (not s.isEmpty())
      node = s.pop()
      visit(node)
      if (node.right != null)
         s.push(node.right)
      if (node.left != null)
         s.push(node.left)
```

### 4.1.2. Inorden

```
//Variante recursiva
inorden(nodo)
   si nodo == nulo entonces retorna
   inorden(nodo.izquierda)
   imprime nodo.valor
   inorden(nodo.derecha)

//Variante iterativa
iterativeInorder(node)
```

```
   s = empty stack
   while (not s.isEmpty() or node != null)
      if (node != null)
         s.push(node)
         node = node.left
      else
         node = s.pop()
         visit(node)
         node = node.right
```

#### 4.1.3. Postorden

```
//Variante recursiva
postorden(nodo)
   si nodo == nulo entonces retorna
   postorden(nodo.izquierda)
   postorden(nodo.derecha)
   imprime nodo.valor

//Variante iterativa
iterativePostorder(node)
   s = empty stack
   lastNodeVisited = null
   while (not s.isEmpty() or node != null)
      if (node != null)
         s.push(node)
         node = node.left
      else
         peekNode = s.peek()
         if (peekNode.right != null and lastNodeVisited != peekNode.right)
            node = peekNode.right
         else
            visit(peekNode)
            lastNodeVisited = s.pop()
```

### 4.2. Recorrido en anchura-primero

También está el pseudocódigo para un simple recorrido en orden por nivel basado en cola.

```
orden_por_nivel(raiz)
   cola = nueva cola
   cola.encola(raiz)
   mientras not cola.vacia hacer
      nodo = cola.desencola()
      visita(nodo)
      si nodo.izquierdo != null entonces
         cola.encola(nodo.izquierdo)
```

```
    si nodo.derecho != null entonces
       cola.encola(nodo.derecha)
```

## 5. Aplicaciones

Es particularmente común usar un recorrido inorden en un árbol binario de búsqueda porque éste retornará valores en el orden del conjunto subyacente, de acuerdo al comparador que configura el árbol de búsqueda binaria (de aquí el nombre).

Para ver porqué éste es el caso, note que si n es un nodo en un árbol binario de búsqueda, entonces todo n en el subárbol izquierdo es menor que n, y todo n en el subárbol derecho es mayor o igual a n. Por lo tanto, si visitamos el subárbol izquierdo en orden, usando una llamada recursiva, y entonces visitamos a n, y después visitamos el subárbol derecho en orden, nosotros hemos visitado completamente el subárbol con raíz en n en orden. Podemos asumir que las llamadas recurrentes visitan correctamente los subárboles en orden usando el principio matemático de inducción estructural. Similarmente, el recorrer en inorden reverso da los valores por orden decreciente.

Recorriendo un árbol en preorden mientras se está insertando los valores en un nuevo árbol es una manera común de hacer una copia completa de un árbol binario de búsqueda.

También se pueden usar los recorridos preorden para conseguir una expresión prefijo (notación polaca) de árboles de expresión: recorra el árbol de expresión en preorden. Para calcular el valor de tal expresión: explore de derecha a izquierda, poniendo los elementos en un stack. Cada vez que se encuentre un operador, se sustituyen los dos símbolos superiores del stack por el resultado de aplicar al operador a esos elementos. Por ejemplo, la expresión * + 2 3 4, que en la notación de infijo es (2 + 3) * 4.

## 6. Complejidad

La complejidad tanto espacial como temporal de todos los recorridos es O($N$) donde $N$ es la cantidad de nodos que posee el árbol. Pero ten cuidado con las implementaciones recursivas y un árbol demasiado grande para ese caso analiza si se puede utilizar una implementación iterativa.

## 7. Ejercicios

A continuación una lista de ejercicios que se pueden resolver utilizando el contenido abordado en la guía:

- DMOJ - Visita a Sydney
- DMOJ - Agua Fría Clara