

Árboles

Una de las estructuras de datos más importantes en programación es el árbol. Pueden usarse los árboles para representar la información en una estructura jerárquica. Los árboles pueden procesarse en forma recursiva y son muy adaptables a pruebas matemáticas. El estudio de árboles ilustra las conexiones entre varios temas de la matemática discreta y ofrece oportunidades para aprovechar la matemática formal en la programación práctica.

La idea de estructura jerárquica es muy usada en la práctica. Por ejemplo, los libros son a menudo organizados como una sucesión de capítulos cada uno de los cuales son una sucesión de secciones que puede tener subdivisiones, y así sucesivamente. Una empresa puede organizarse como las colecciones de unidades comerciales cada uno de las cuales pueden tener varias secciones. Las secciones, a su vez, pueden tener secciones múltiples, y así sucesivamente.

El software es organizado como una colección de módulos cualquiera que pueden constituirse de varios submódulos, con el nivel de refinamiento que los diseñadores encuentren apropiado. En cierto nivel, los módulos se expresan en unidades básicas como los objetos, los métodos, o procedimientos.

En otros términos, las estructuras jerárquicas proporcionan una eficaz manera de organizar la información.

Los árboles proporcionan una capacidad enorme para expresar la idea de jerarquía. Ellos son objetos formales, matemáticos.

Definición 1

Un árbol o bien es un árbol vacío o es un nodo junto con una sucesión de árboles. Sea A un conjunto cualquiera:

1. $nil \in (\text{Arbol } A)$
2. $(\text{cons } a \ a1 \ a2 \dots \ a_n) \in (\text{Arbol } A)$ si i
 $(a \in A) \wedge (a_1, a_2, \dots, a_n \in (\text{Arbol } A))$

La definición es inductiva. El punto de arranque para la definición inductiva es el árbol vacío. La definición no dice lo que un árbol vacío es; esto queda como un término indefinido, y la existencia del árbol vacío se acepta como un axioma. El término "nodo" no se define, y la existencia de nodos para construir árboles también se toma como un axioma.

Más adelante cuando se usen árboles para representar entidades matemáticas específicas diremos exactamente qué entidades comprenden el la información que contiene cada nodo que compone el árbol que se está construyendo.

Definición 2

El primer nodo que se agrega a un árbol no vacío es la raíz del árbol. Cada miembro individual de la sucesión de árboles en la que se divide un árbol no vacío se denomina hijo.

Definición 3

Un árbol no vacío cuya la sucesión asociada de árboles está vacía se llama hoja. Una hoja sola es el tipo más simple de árbol no vacío. En este árbol la raíz es una hoja. En un árbol más complejo, es decir, uno que consiste en un nodo con hijos, la raíz no es una hoja.

Definición 4

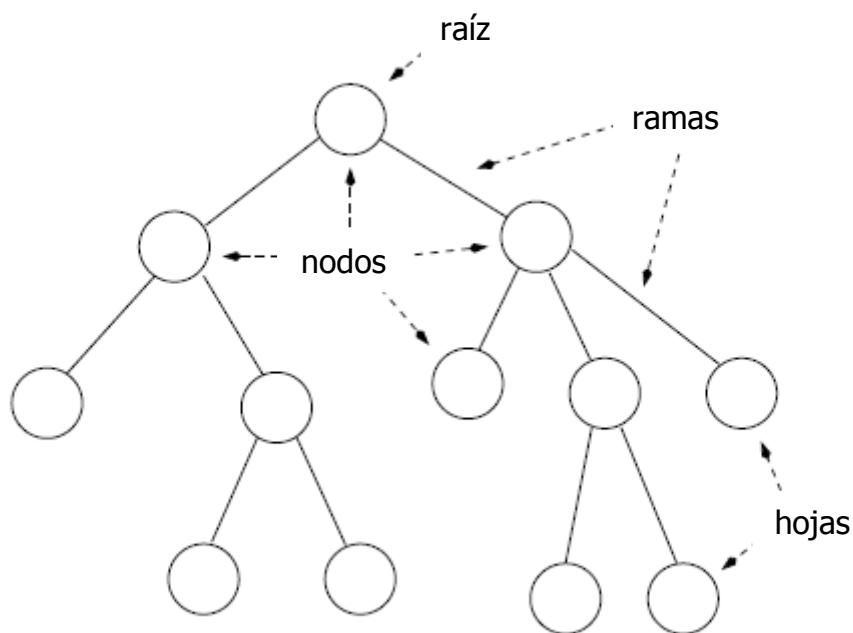
Se dice que s es un subárbol de t , si s es el propio t o si t es no vacío y s es un subárbol de uno de los hijos de t .

La definición del término subárbol también es inductiva.

Definición 5

Un árbol s es una hoja de un árbol t si s es un subárbol de t y s es una hoja.

Diagrama de un árbol, representación gráfica



Definición 6.

Se dice que un nodo n ocurre en un árbol t (o pertenece a t) y se denota $n \in t$, si t es un árbol que consiste en un cierto nodo m con una sucesión de hijos (a_1, a_2, \dots, a_n) y n o bien es m , o bien n pertenece a uno de los hijos de m .

Ningún nodo pertenece al árbol vacío, por definición.

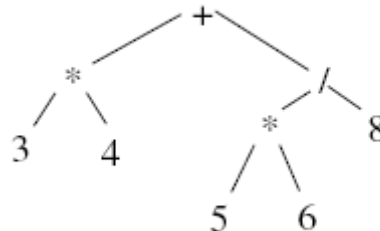
Definición 7.

Se dice que un nodo n es un nodo interior al árbol t si n pertenece a t y existe una sucesión de árboles (a_1, a_2, \dots, a_n) tal que n junto con esa sucesión es un subárbol de t .

Los árboles normalmente contienen datos adicionales en sus nodos y hojas.

La estructura del árbol (comprendiendo los nodos y hojas) proporciona una organización para los valores de los datos, haciendo que la información sea más fácil de usar que si simplemente estuviera contenida en una lista.

Se considera el árbol que representa la expresión aritmética $(3 \times 4) + ((5 \times 6)/8)$. la raíz del árbol es el + el funcionamiento, y cada subárbol representa expresiones que describen los argumentos a ser agregados. Las hojas del árbol representan los números que aparecen en la expresión. El valor de la expresión puede calcularse trabajando desde las hojas a la raíz, mientras se van calculando los valores intermedios que corresponden a cada operador.



Muchos intérpretes de lenguajes de programación y compiladores se acostumbran a representar con árboles la estructura del programa entero.

Árboles binarios

El caso particular de árboles donde cada nodo debe tener exactamente dos hijos se llama **árbol binario**.

Como se dijo antes un nodo de un árbol puede tener cualquier cantidad de hijos. Los árboles binarios normalmente se usan en las aplicaciones prácticas de computación. El ejemplo anterior de las EA se representa utilizando un árbol binario

Definición inductiva de árboles binarios

1. $nil \in (AB A)$
2. $(cons\ a\ izq\ der) \in (AB A)$ si $a \in A \wedge izq \in (AB A) \wedge der \in (AB A)$

Representación de árboles binarios en Haskell

Árbol binario de enteros.

```

data BinTreeInt = Leaf
  | Node Integer BinTreeInt BinTreeInt
  
```

Dar los diagramas de los siguientes árboles implementados en Haskell:

- `tree1 :: BinTreeInt`
`tree1 = Leaf`
- `tree2 :: BinTreeInt`
`tree2 = Node 23 Leaf Leaf`
- `tree3 :: BinTreeInt`
`tree3 =`
 `Node 4`
 `(Node 2`
 `(Node 1 Leaf Leaf)`
 `(Node 3 Leaf Leaf))`
 `(Node 7`
 `(Node 5`
 `Leaf`
 `(Node 6 Leaf Leaf))`
 `(Node 8 Leaf Leaf))`

Haskell también permite definir árboles polimórficos, dónde los datos de los nodos son de algún tipo `a`. El árbol resultado tiene tipo `BinTree a`, que significa "árbol binario con valores de tipo `a`".

Árbol binario de un tipo `a`:

```
data BinTree a = BinLeaf
               | BinNode a (BinTree a) (BinTree a) deriving
               Show
```

- `tree4 :: BinTree String`
`tree4 = BinNode "cat" BinLeaf (BinNode "dog" BinLeaf BinLeaf)`
- `tree5 :: BinTree (Integer, Bool)`
`tree5 = BinNode (23, False)`
 `BinLeaf`
 `(BinNode (49, True) BinLeaf BinLeaf)`
- `tree6 :: BinTree Int`
`tree6 = BinNode 4`
 `(BinNode 2`
 `(BinNode 1 BinLeaf BinLeaf)`
 `(BinNode 3 BinLeaf BinLeaf))`
 `(BinNode 6`
 `(BinNode 5 BinLeaf BinLeaf)`
 `(BinNode 7 BinLeaf BinLeaf))`

Recorrida de árboles

Una tarea común es recorrer uno a uno los nodos de un árbol con el fin de procesar los datos en de cada nodo, creando una lista como resultado. Un algoritmo que realiza esta función se denomina recorrida de un árbol.

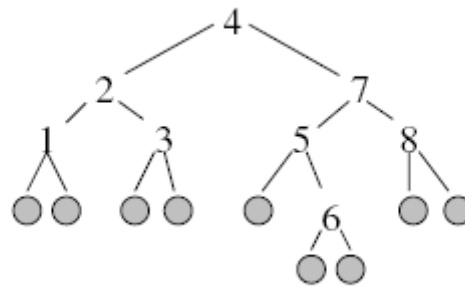
Para árboles binarios se utilizan comúnmente tres algoritmos para recorridas:

Preorden: se visita primero la raíz y a continuación, se recorre en preorden el subárbol izquierdo y luego en preorden el subárbol derecho.

Enorden: se visita el subarbol izquierdo en enorden, a continuación la raíz y por último el subárbol derecho en enorden.

Postorden: se visita el subárbol izquierdo en postorden, luego en postorden el subárbol derecho y por último la raíz.

Ejemplo:



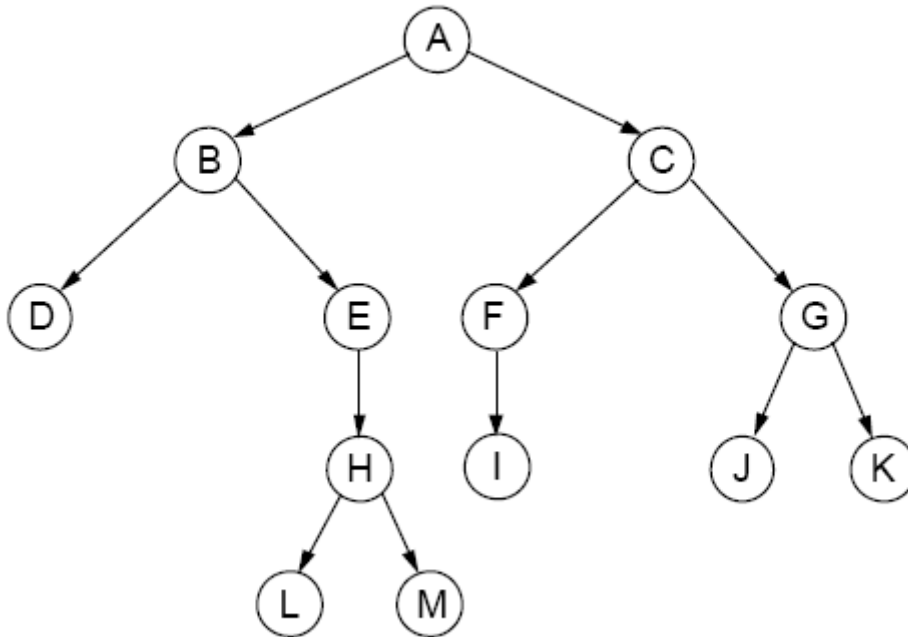
Recorrida en preorden del árbol de la figura: [4, 2, 1, 3, 7, 5, 6, 8].

Recorrida en enorden del árbol de la figura: [1, 2, 3, 4, 5, 6, 7, 8].

Recorrida en postorden del árbol de la figura: [1, 3, 2, 6, 5, 8, 7, 4]

Ejercicios:

Para el árbol binario representado por el siguiente diagrama:



1. Listar los nodos del árbol anterior en:
 - a. Preorden
 - b. Enorden
 - c. Postorden

2. Definir funciones que recorran un árbol binario en:
 - a. Preorden
 - b. Enorden
 - c. Postorden

Ejercicios:

1. Definir un tipo de datos árbol que contiene un carácter y un entero en cada nodo, y exactamente tres subárboles.
2. Definir un tipo de datos árbol que contiene un entero en cada nodo, y que permite a cada nodo tener cualquier número de subárboles.
3. Defina el tipo de datos Tree que representa árboles binarios de elementos de un tipo genérico que sólo guarda información en los nodos hojas (nodos externos). Los nodos internos no guardan información. El árbol más pequeño es una hoja.
 - a. Defina una función mapTree que dado un árbol de tipo (Tree A), para un tipo genérico A, y una función $f:A \rightarrow B$, con B un conjunto dado, retorne un árbol de tipo (Tree B) obtenido por la aplicación de la función f a cada uno de los nodos hojas del árbol parámetro.
 - b. Defina una función que cuente la cantidad de nodos hojas que posee un árbol de tipo (Tree A), para un tipo genérico A.
 - c. Pruebe que la función MapTree preserva la cantidad de nodos hojas del árbol parámetro.
 - d. Defina una función hojas que retorne una lista con las hojas de un árbol de tipo (Tree A), para un tipo genérico A. Pruebe luego que la cantidad de nodos hojas que posee un árbol de tipo (Tree A), para un tipo genérico A, es igual a la longitud de la lista resultante de aplicarle la función hojas al árbol.
4. Defina el tipo de datos (BinTree A) de árboles binarios con nodos internos de un tipo genérico A y nodos externos (hojas) de un tipo genérico B. El árbol más pequeño es una hoja.
 - a. Defina una función que cuente la cantidad de nodos externos de un árbol binario de tipo (BinTree A).
 - b. Defina una función que cuente la cantidad de nodos internos de un árbol binario de tipo BinTree.
 - c. Pruebe que la cantidad de nodos externos en un árbol binario de tipo BinTree es igual a la cantidad de nodos internos del árbol más 1.