Tema 1. ÁRBOLES GENERALES y BINARIOS

Estructuras de Datos y Algoritmos II
Grado en Ingeniería Informática

M José Polo Martín
mjpolo@usal.es

Universidad de Salamanca

curso 2018-2019

Contenido

1 Definiciones y conceptos básicos

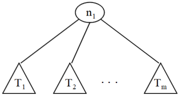
Árbol

Colección de elementos llamados nodos, uno de los cuales se distingue como **raíz**, con una relación entre ellos que impone una estructura **jerárquica**

Definición formal

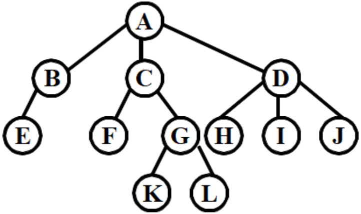
Un **árbol** T es un conjunto finito de cero o más nodos $\{n_1, n_2, ..., n_i\}$ de tal forma que

1. Hay un nodo especialmente designado llamado raíz ($n_1 = raíz(T)$)
2. Los nodos restantes $\{n_2, n_3, ..., n_i\}$ se dividen en **m** conjuntos disjuntos $T_1, T_2, ..., T_m$ ($m \geq 0$), llamados **subárboles** de la raíz, tales que cada T_j es, por sí mismo, un árbol



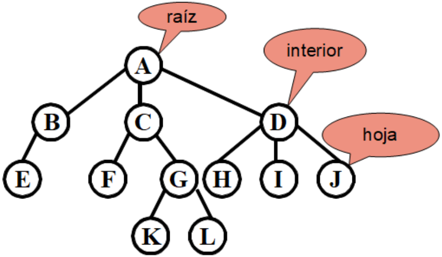
Características y propiedades

- **Recursión** característica inherente de los árboles
- Un árbol sin nodos es un **árbol vacío** o **nulo**
- Todo árbol que no es vacío tiene un **único** nodo **raíz**
- Un nodo X es **descendiente directo** de un nodo Y si existe una arista del nodo Y al nodo X (X "es hijo" de Y)
- Un nodo Z es **antecesor directo** de un nodo V si existe una arista del nodo Z al nodo V (Z "es padre" de V)
- Todos los nodos que son descendientes directos (hijos) de un mismo nodo (padre) se designan como **hermanos**
- Cada nodo puede tener un número arbitrario de descendientes directos, incluso cero



Definciones

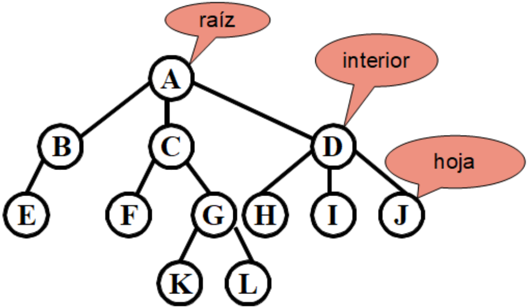
- Todos los nodos que no tienen descendientes directos se denominan nodos **hoja** o **terminales**
- Todo nodo con descendientes, es decir, que no es hoja, se designa como nodo **interno**
- **Grado de un nodo:** número de sus descendientes directos
- **Grado del árbol:** grado máximo de todos los nodos del árbol



Definiciones

- **Camino** de un nodo n_i a otro nodo n_k: secuencia de nodos $n_i, n_{i+1}, ..., n_k$ tal que n_j es el padre de n_{j+1} para $i \leq j < k$
  - **Longitud de un camino:** número de aristas que lo forman (k-1)
  - En un árbol hay exactamente un camino de la raíz a cada nodo
- Si existe un camino de un nodo n_i a otro nodo n_j, entonces n_j es **descendiente** de n_i y n_i es **antecesor** de n_j
- **Profundidad de un nodo:** número de aristas en el camino único que permite acceder a él desde la raíz. La raíz tiene profundidad cero
- **Altura de un nodo:** número de aristas en el camino más largo desde el nodo en cuestión hasta una hoja. Todos los nodos hoja tienen altura 0
- **Altura de un árbol** es igual a la altura de la raíz

Ejemplo

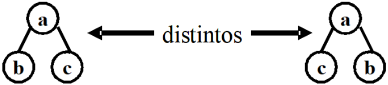


grado(A) $= 3$	profundidad(A) $= 0$	altura(A) $= 3$
grado(C) $= 2$	profundidad(C) $= 1$	altura(C) $= 2$
...	...	...
grado(K) $= 0$	profundidad(K) $= 3$	altura(K) $= 0$

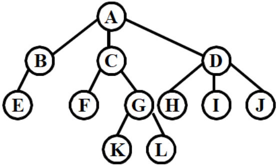
grado(árbol) $= 3$ altura(árbol) $= 3$

Árboles ordenados

- Árboles en los que los hijos de cada nodo se ordenan de izquierda a derecha

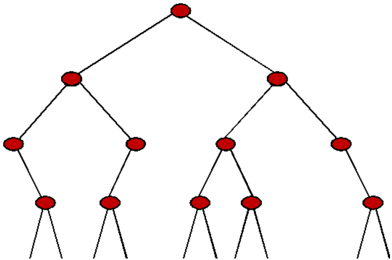


- El orden de izquierda a derecha entre nodos hermanos puede extenderse para comparar dos nodos cualesquiera con el siguiente criterio: “ si **b** y **c** son hermanos y **b** está a la izquierda de **c**, entonces todos los descendientes de **b** están a la izquierda de todos los descendientes de **c**".



2 Árboles Binarios

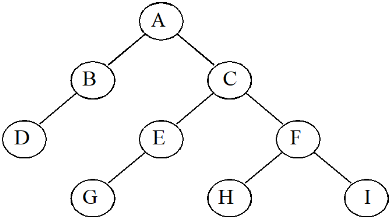
- **Definición 1:** conjunto finito de nodos que puede ser vacío o puede distribuirse en una raíz y un par de árboles binarios llamados subárboles izquierdo y derecho de la raíz, los cuales pueden también estar vacíos
- **Definición 2:** árbol ordenado de grado dos



2.1 Representación mediante Memoria Contigua

Matrices

- Los nodos se almacenan en las celdas contiguas de una matriz
- Cada celda de la matriz almacenará la información del nodo y dos enteros que indicarán los índices de la matriz donde se encuentran sus descendientes directos izquierdo y derecho (el valor 0 indicará ausencia de hijo)

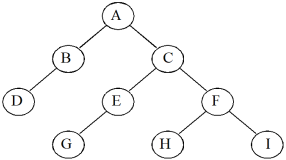


1	A	2	3
2	B	4	0
3	C	5	6
4	D	0	0
5	E	7	0
6	F	8	9
7	G	0	0
8	H	0	0
9	I	0	0
10			
11			
	...		
N-2			
N-1			
N			

Declaraciones Básicas

Consideraciones:

- Utilización de un matriz suficientemente grande para almacenar el número máximo de nodos que pueda tener el árbol
- Necesidad de una lista de disponibilidad para manejar el espacio en la matriz y un entero para indicar el índice de la matriz donde se encuentra la raíz



x.raíz → 1	A	2	3
2	B	4	0
3	C	5	6
4	D	0	0
5	E	7	0
6	F	8	9
7	G	0	0
8	H	0	0
9	I	0	0
x.disponible → 10			11
11			12
	...		
N-2			n-1
N-1			n
N			0

Algoritmo 1.1 declaraciones básicas

constantes
  $N = 100$
tipos
tipoNodo = **registro**
  información : *tipoIformación*
  izq, *der* : *entero*
fin registro
tipoÁrbol = **registro**
  nodos : *matriz*$[1, \ldots, N]$ de tipoNodo
  raíz : *entero*
  disponible : *entero*
fin registro

Ejemplo   variable x:tipoÁrbol

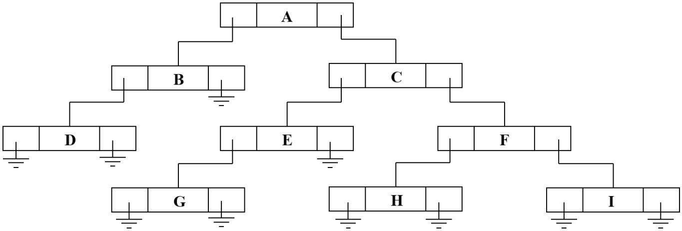
Para todo nodo situado en la celda i:

- Hijo izquierdo $\Rightarrow x.nodos[i].izq$
- Hijo derecho $\Rightarrow x.nodos[j].der$

2.2 Representación mediante Memoria Dispersa

Memoria dinámica (punteros)

- Cada celda se enlaza, a lo sumo, con otras dos siguiendo la estructura del árbol que representa
- Cada celda almacenará la información del nodo y dos apuntadores para enlazar con las raíces de los subárboles izquierdo y derecho del nodo
- Los nodos se almacenan en celdas que no tienen por qué ocupar posiciones consecutivas en memoria



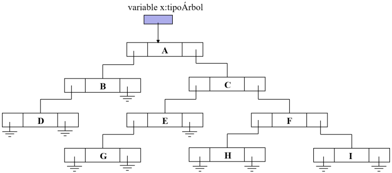
Declaraciones Básicas

Consideraciones:

- Desde la raíz de un árbol puede llegarse al resto de los nodos
- Para acceder a un árbol solo se necesita la dirección de memoria donde se almacena la raíz ⇒ **El árbol puede representarse por un puntero a un nodo de árbol binario**

Algoritmo 1.2 declaraciones básicas

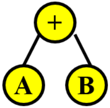
tipos
tipoNodo = **registro**
  izq :↑ *tipoNodo*
  información : *tipoIformación*
  der :↑ *tipoNodo*
fin registro
tipoÁrbol :↑ *tipoNodo*
punteroNodo :↑ *tipoNodo*

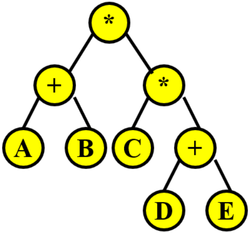


Este es el tipo de representación más común y la que utilizaremos en el resto del tema

Aplicaciones

- Numeración de capítulos y secciones de un libro
- Análisis de circuitos eléctricos
- Representación de expresiones aritméticas:





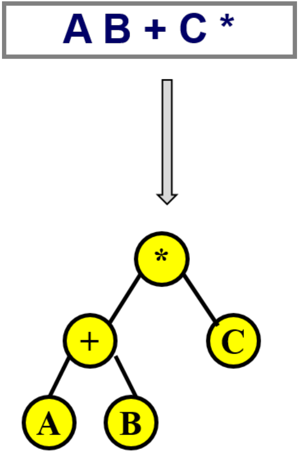
Prefija	Infija	Postfija
+AB	A+B	AB+
*+AB*C+DC	(A+B)*(C*(D+E))	AB+CDE+**

Ejemplo de aplicación: construcción de un árbol algebraico

Algoritmo 1.3 generaÁrbol(expresión:**cadena**):**tipoÁrbol**

Entrada: expresión algebraica en notación Postfija
Salida: árbol binario que reprensenta la expresión de entrada

```
a : tipoÁrbol
p : tipoPila
i : entero
símbolo : carácter
creaVacia(p)
símbolo ← expresión[1]
i ← 1
mientras símbolo ≠ FINEXPRESIÓN hacer
   casos símbolo en
     operando :
            a ← creaNodo(símbolo)
            inserta(a, p)
     operador:
            a ← creaNodo(símbolo)
            a ↑ .der ← suprime(p)
            a ↑ .izq ← suprime(p)
            inserta(a, p)
   fin casos
   i ← i + 1
   simbolo ← expresión[i]
fin mientras
devolver suprime(p)
```



Obsérvese la utilización del TAD pila en la implementación de este algoritmo

Especificación del TAD PILA utilizado en el ejemplo

- Pila cuyos elementos son punteros a nodos de árboles binarios
- Operaciones básicas:
  - **creaVacía(p):** inicia o crea la pila **p** como una pila vacía, sin ningún elemento
  - **vacía(p):** devuelve verdadero si la pila **p** está vacía, y falso en caso contrario
  - **inserta(x,p):** añade el elemento **x** a la pila **p** convirtiéndolo en el nuevo tope o cima de la pila
  - **suprime(p):** devuelve y elimina el elemento del tope o cima de la pila **p**

3 Recorridos en Árboles Binarios

- Recorrer un árbol: visitar todos sus nodos de forma sistemática, de tal manera que cada nodo sea visitado una sola vez
- Un nodo es visitado cuando se encuentra en el recorrido y en ese momento se puede efectuar cualquier proceso sobre su contenido
- Categorías básicas de recorridos:
    - en PROFUNDIDAD: basados en las relaciones padre-hijo de los nodos
    - en AMPLITUD o por NIVELES: basados en la distancia de cada nodo a la raíz

Recorridos en PROFUNDIDAD

- Existen tres métodos diferentes de efectuar el recorrido en profundidad, todos ellos de naturaleza recursiva, imponiendo un orden secuencial y lineal a los nodos del árbol
- Las actividades básicas a realizar son:
  - visitar la raíz
  - recorrer el subárbol izquierdo
  - recorrer el subárbol derecho
- Los tres métodos se diferencian en el orden en que se visite la raíz, dando lugar a los tres recorridos:
  - EN-ORDEN
  - PRE-ORDEN
  - POST-ORDEN

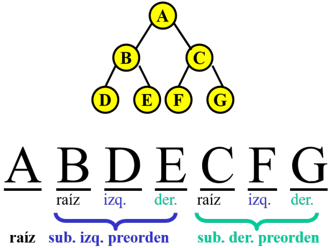
Recorrido PRE-ORDEN

1. Visitar la raíz
2. Recorrer en PRE-ORDEN el subárbol izquierdo
3. Recorrer en PRE-ORDEN el subárbol derecho

Algoritmo 1.4 preOrden(a : tipoÁrbol)

Entrada: *a* dirección del nodo raíz del árbol
Salida: procesa todos los nodos en preorden

si $a \neq NULO$ **entonces**
    $visitar(a \uparrow .información)$
    $preOrden(a \uparrow .izq)$
    $preOrden(a \uparrow .der)$
fin si



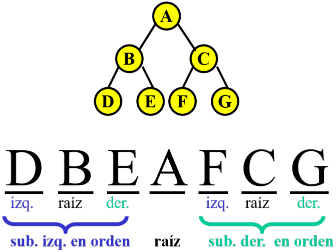
Recorrido EN-ORDEN

1. Recorrer en En-ORDEN el subárbol izquierdo
2. Visitar la raíz
3. Recorrer en EN-ORDEN el subárbol derecho

Algoritmo 1.5 enOrden(a : tipoÁrbol)

Entrada: *a* dirección del nodo ráiz del árbol
Salida: procesa todos los nodos en orden

1: **si** $a \neq NULO$ **entonces**
2:     $enOrden(a \uparrow .izq)$
3:     $visitar(a \uparrow .información)$
4:     $enOrden(a \uparrow .der)$
5: **fin si**



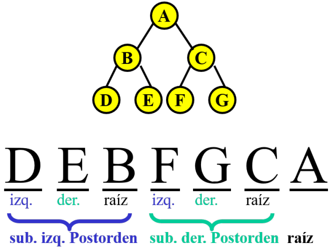
Recorrido POST-ORDEN

1. Recorrer en POST-ORDEN el subárbol izquierdo
2. Recorrer en POST-ORDEN el subárbol derecho
3. Visitar la raíz

Algoritmo 1.6 postOrden(a : tipoÁrbol)

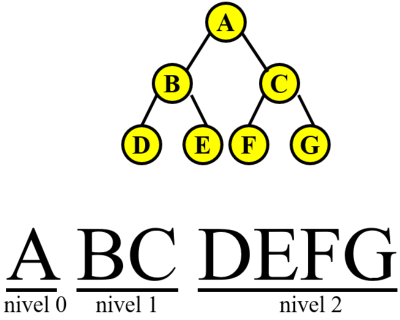
Entrada: *a* dirección del nodo ráiz del árbol
Salida: procesa todos los nodos en postorden

si $a \neq NULO$ **entonces**
    $postOrden(a \uparrow .izq)$
    $postOrden(a \uparrow .der)$
    $visitar(a \uparrow .información)$
fin si



Recorrido en AMPLITUD o por NIVELES

- Consiste en visitar primero la raíz del árbol, después los nodos que se encuentran en siguiente nivel, etc.
- En este recorrido no importa tanto la estructura recursiva del árbol, si no la distribución de los nodos en los diferentes niveles
- Una posible implementación puede efectuarse utilizando una cola cuyos elementos son punteros a nodos del árbol

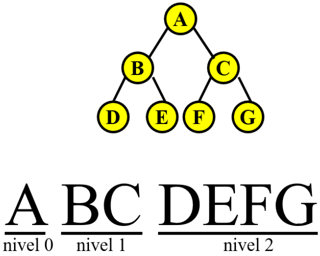


Algoritmo de recorrido en AMPLITUD

Algoritmo 1.7 niveles(a : tipoÁrbol)

Entrada: *a* dirección del nodo ráiz del árbol
Salida: procesa todos los nodos por niveles

```
nodo : punteroNodo
c : tipoCola
creaVacia(c)
si a ≠ NULO entonces
    inserta(a, c)
fin si
mientras NOT(vacia(c)) hacer
    nodo ← suprime(p)
    visitar(nodo ↑ .información)
    si nodo ↑ .izq ≠ NULO entonces
        inserta(nodo ↑ .izq, c)
    fin si
    si nodo ↑ .der ≠ NULO entonces
        inserta(nodo ↑ .der, c)
    fin si
fin mientras
```



Obsérvese la utilización del TAD cola en la implementación de este algoritmo

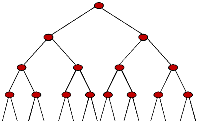
Especificación del TAD COLA utilizado en algoritmo amplitud

- Cola cuyos elementos son punteros a nodos de árboles binarios
- Operaciones básicas:
  - **creaVacía(c):** inicia o crea la cola c como una cola vacía, sin ningún elemento
  - **vacía(c):** devuelve verdadero si la cola c está vacía, y falso en caso contrario
  - **inserta(x,c):** añade el elemento x a la cola c convirtiéndolo en el último elemento de cola
  - **suprime(c):** devuelve y elimina el primer elemento la cola c

4 Árboles Binarios Completos y Casi-Completos

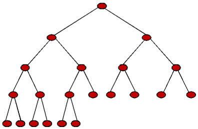
Árbol Binario Completo

Árbol binario que contiene el número máximo de nodos para su altura



Árbol Binario Casi-Completo

Árbol binario completamente lleno, con la posible excepción del nivel más bajo, el cual se llena de izquierda a derecha



Característica: tienen la altura mínima para su conjunto de nodos

Altura de un árbol binario de N nodos

- Altura máxima ⇒ lista de nodos

$$H_{max} = N$$

- Altura mínima ⇒ árbol binario casi-completo

$N = 1$	$H_{min} = 0$	$N_{min} = 1 = 2^0$	$N_{max} = 1 = 2^1 - 1$
$2 \leq N \leq 3$	$H_{min} = 1$	$N_{min} = 2 = 2^1$	$N_{max} = 3 = 2^2 - 1$
$4 \leq N \leq 7$	$H_{min} = 2$	$N_{min} = 4 = 2^2$	$N_{max} = 7 = 2^3 - 1$
$8 \leq N \leq 15$	$H_{min} = 3$	$N_{min} = 8 = 2^3$	$N_{max} = 15 = 2^4 - 1$
...	...	...	...
N	$H_{min} = h$	$N_{min} = 2^h$	$N_{max} = 2^{h+1} - 1$

$$H_{min} = h = [logN]$$

- Número de nodos de un a.b. casi-completo de altura h

$$2^h \leq N \leq 2^{h+1} - 1$$

Más aplicaciones: Árboles de Búsqueda

- Los árboles binarios suelen utilizarse para estructurar una colección de datos de forma que faciliten la búsqueda de un elemento particular
- Conviene, por tanto, organizar los datos de forma que las longitudes de los caminos de búsqueda sean mínimas$\Rightarrow$Árboles binarios CASI-COMPLETOS y sus variantes
  - Árboles Binarios de Búsqueda y Balanceados, los estudiaremos en el tema 5
  - Árboles Multicamino: extensiones de los Árboles Balanceados que se utilizan para búsqueda de información en memoria secundaria, se estudiaran en el tema 7, dedicado a la Organización de Índices (Árboles B y Árboles B+)