Tema 1. ÁRBOLES GENERALES Y BINARIOS

ESTRUCTURAS DE DATOS Y ALGORITMOS II
GRADO EN INGENIERÍA INFORMÁTICA

María José Polo Martín

mjpolo@usal.es

curso 2015-2016

VNiVERSiDAD D SALAMANCA
CAMPUS DE EXCELENCIA INTERNACIONAL

DEPARTAMENTO DE iNFORMÁTiCA Y AVTOMÁTiCA

Tema 1. Árboles Generales y Binarios

1. Definiciones y conceptos básicos
2. Nivel de representación o implementación
   - Representación de árboles binarios
   - Representación de árboles generales
3. Recorridos en Árboles Binarios
   - En profundidad
   - En amplitud

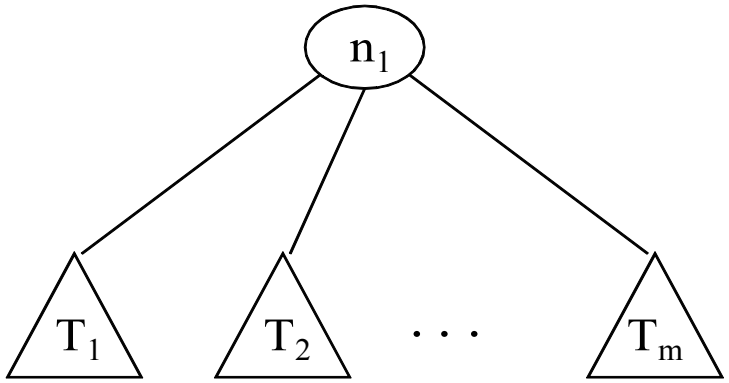
1 Definiciones y conceptos básicos

- **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 $\{n_1, n_2, ..., n_n\}$ de tal forma que

1. Hay un nodo especialmente designado llamado **raíz**

   $n_1 = \text{raíz}(T)$

2. Los nodos restantes $\{n_2, ..., n_n\}$ se dividen en **m** conjuntos disjuntos $T_1, T_2, ..., T_m (m \geq 0)$, llamados **subárboles** de la raíz, tales que cada T_i es, por sí mismo, un árbol



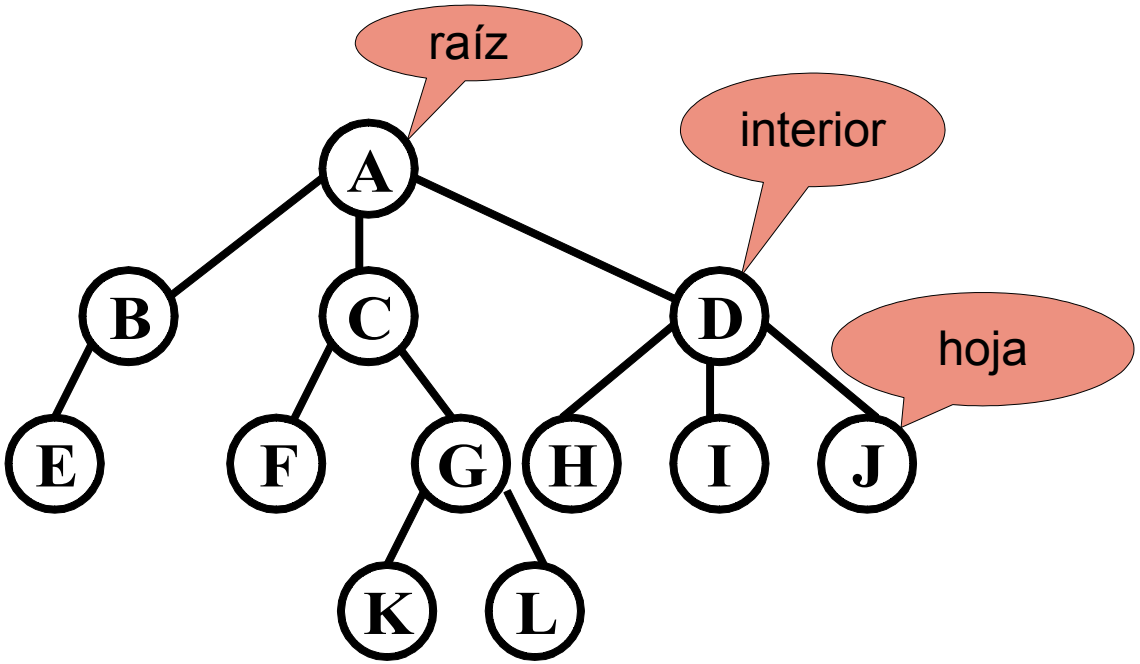
Características y propiedades

- **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 X es **antecesor directo** de un nodo Y si existe una arista del nodo X al nodo Y (X "es padre" de Y)
- 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
- 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**

Definiciones

- **Grado de un nodo**: número de sus descendientes directos
- **Grado del árbol**: grado máximo de todos los nodos del árbol
- **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. Así 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. Así todas las hojas 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

.....

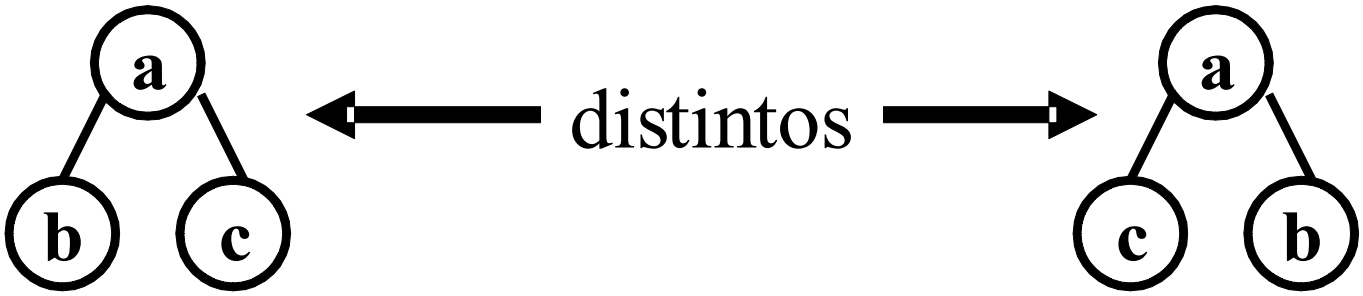
grado(K) = 0 profundidad(K) = 3 altura(K) = 0

grado(árbol) = 3

altura(árbol) = 3

Árboles ordenados

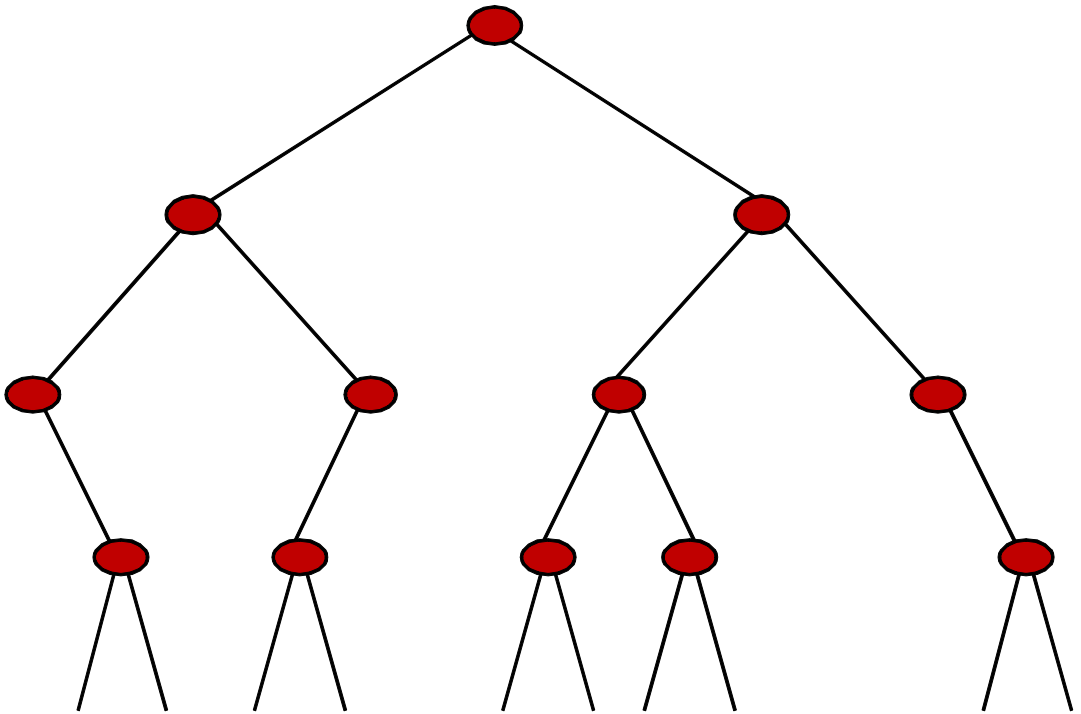
- Árboles en los que los hijos de un nodo se ordenan de izquierda a derecha



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

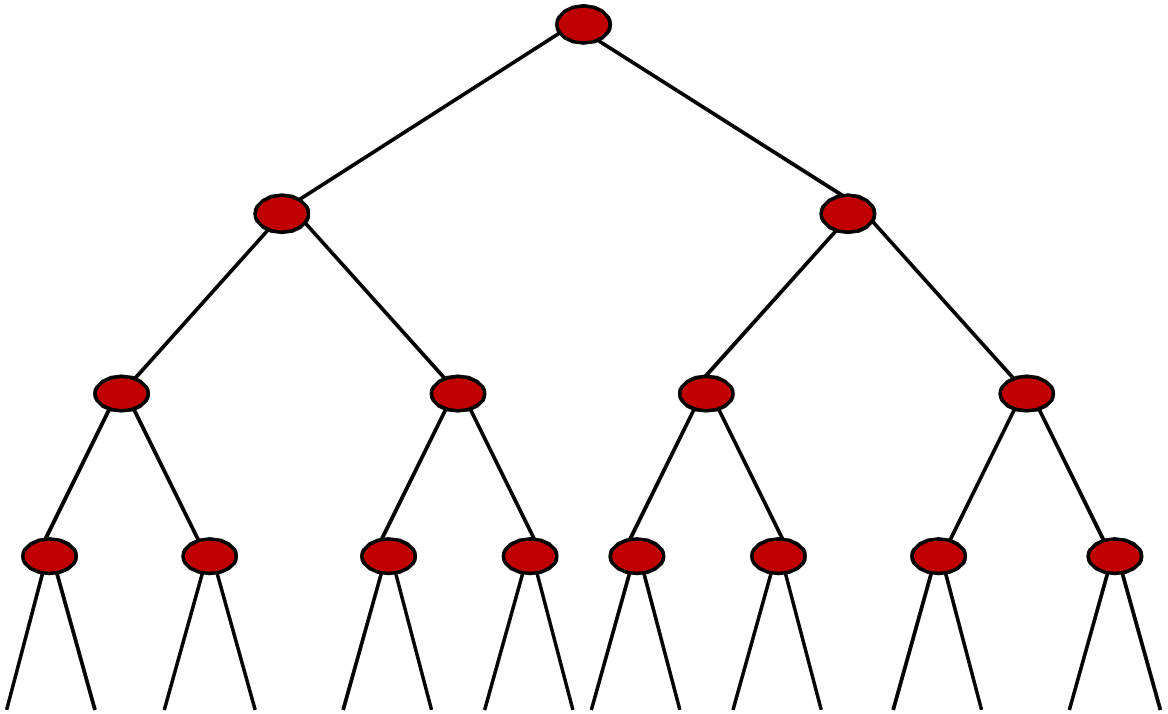
ÁRBOL BINARIO

- **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



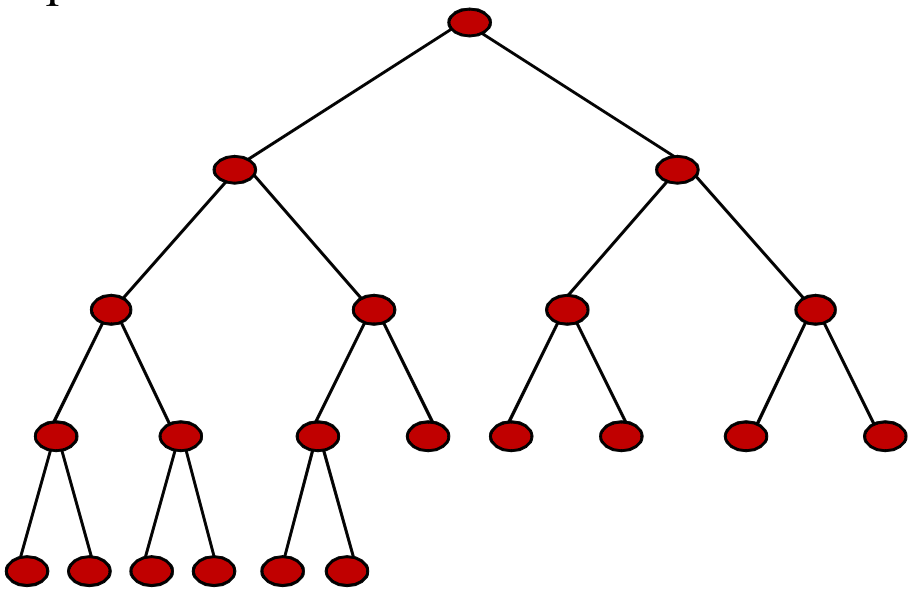
ÁRBOL BINARIO COMPLETO

- **Definición:** 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**: los árboles casi-completos tienen la altura mínima para su conjunto de nodos

- 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 ⇒ Árboles binarios CASI-COMPLETOS y sus variantes (**árboles balanceados, árboles B y árboles B$^+$**)

Altura de un árbol binario de N nodos

- Altura máxima ⇒ lista de nodos
- Altura mínima ⇒ árbol binario casi-completo

$$H_{max} = N$$

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

$$H_{\min} = h = \lceil \log_2 N \rceil$$

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

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

2 Nivel de representación o implementación

- Representación de Árboles Binarios
  1. Mediante matrices
  2. Mediante punteros
- Representación de Árboles Generales
  1. Extensión de la representación de árboles binarios
  2. Conversión de árboles generales a binarios
- Árboles Multicamino: extensiones de los Árboles Binarios de Búsqueda (tema 5) que se utilizan para búsqueda de información en memoria secundaria, se estudiaran en el tema 7, dedicado a la Organización de Índices
  1. Árboles B
  2. Árboles B+

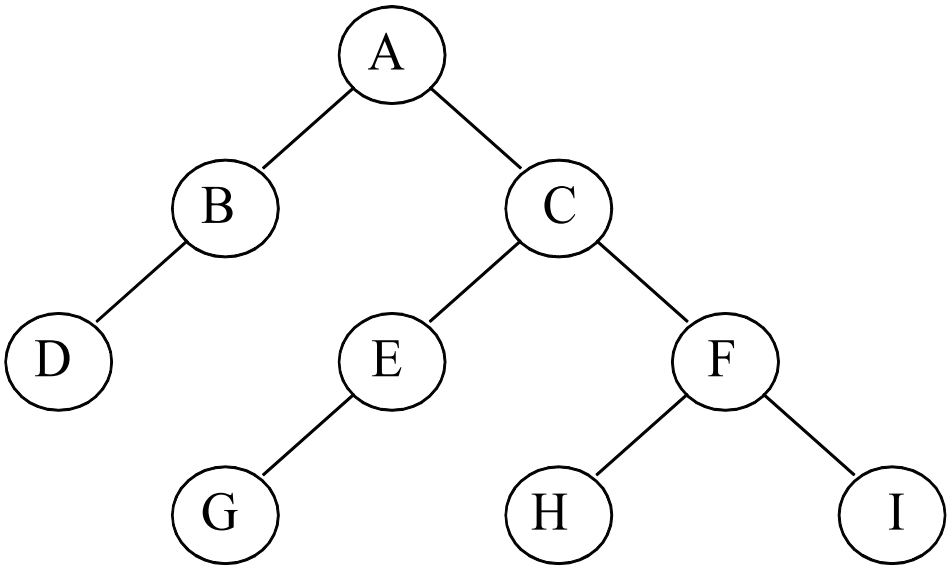
Representación mediante matrices

- 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)
- 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

Declaraciones básicas

```
constantes
    N = 100
tipos
  tipoNodo = registro
            información : tipoInformación
            izq, der : entero
  fin registro
  tipoÁrbol = registro
            nodos : matriz[1..N] de tipoNodo
            raíz : entero
            disponible: entero
 fin registro
```

Ejemplo (var x:tipoÁrbol)



x.raíz

x.disponible

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
11			12
.			
.			
.			
N-2			N-1
N-1			N
N			0

Para todo nodo situado en la celda i:

- Hijo izquierdo ⇒ x.nodos[i].izq
- Hijo derecho ⇒ x.nodos[i].der

Representación mediante punteros

- Los nodos se almacenan en celdas que no tienen porque ocupar posiciones consecutivas en memoria
- 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
- 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
  - Este es el tipo de representación más común y la que utilizaremos en el resto del tema

Declaraciones básicas y ejemplo (var miArbol:tipoÁrbol)

tipos

tipoNodo = **registro**

izq: ↑tipoNodo

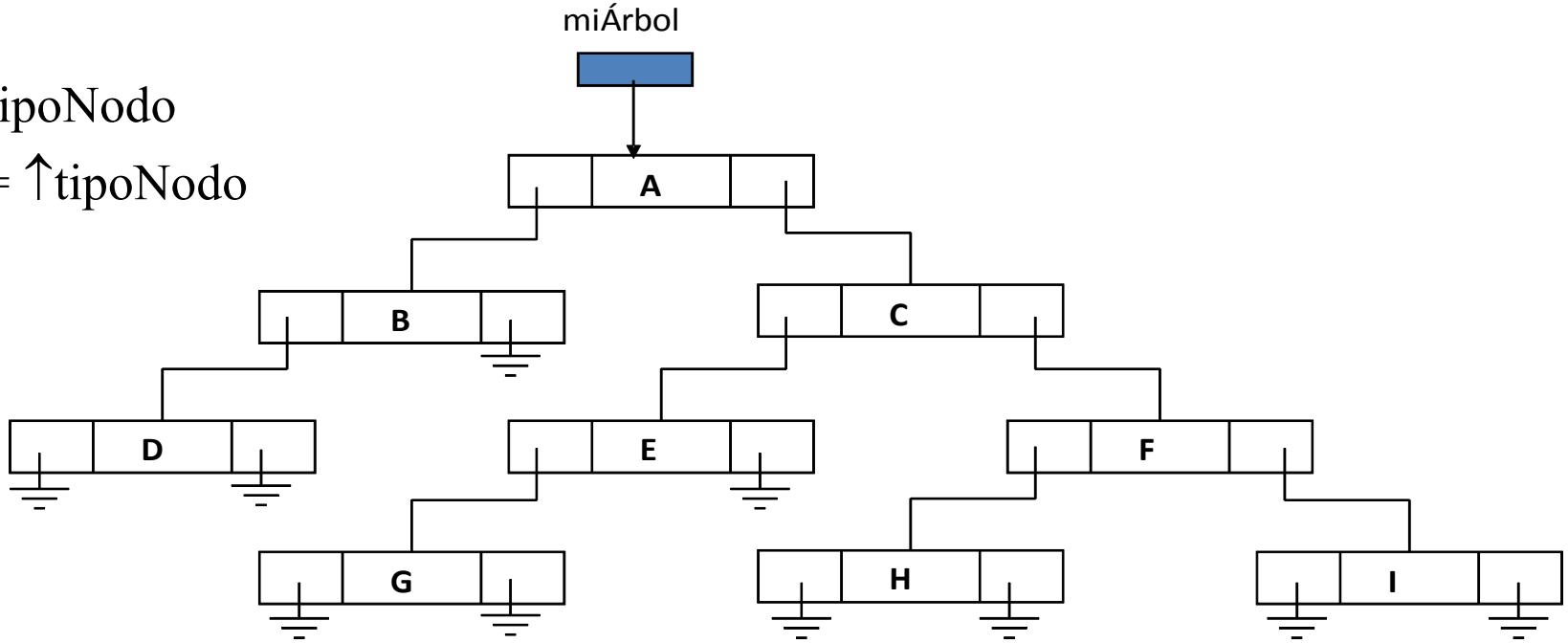
información : tipoInformación

der : ↑tipoNodo

fin registro

tipoÁrbol = ↑tipoNodo

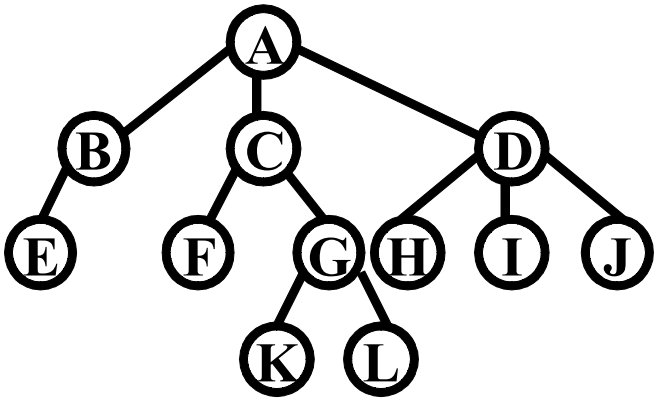
punteroNodo = ↑tipoNodo

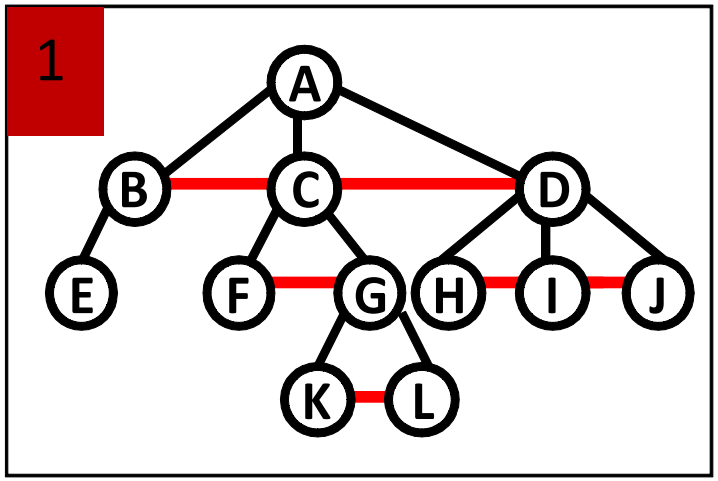


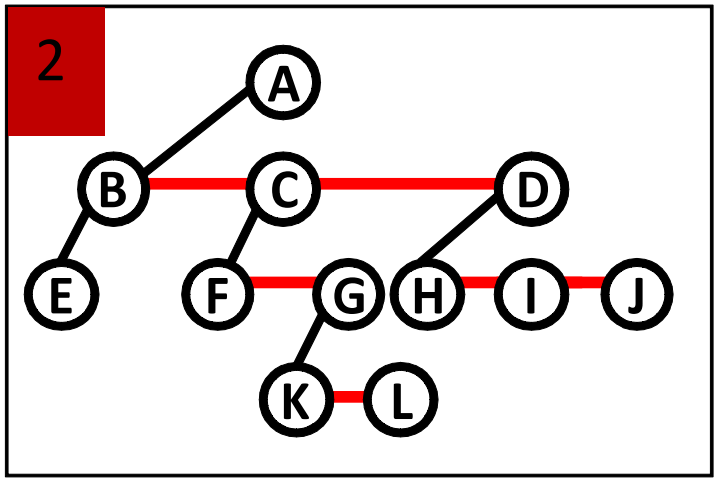
Representación de árboles generales como binarios

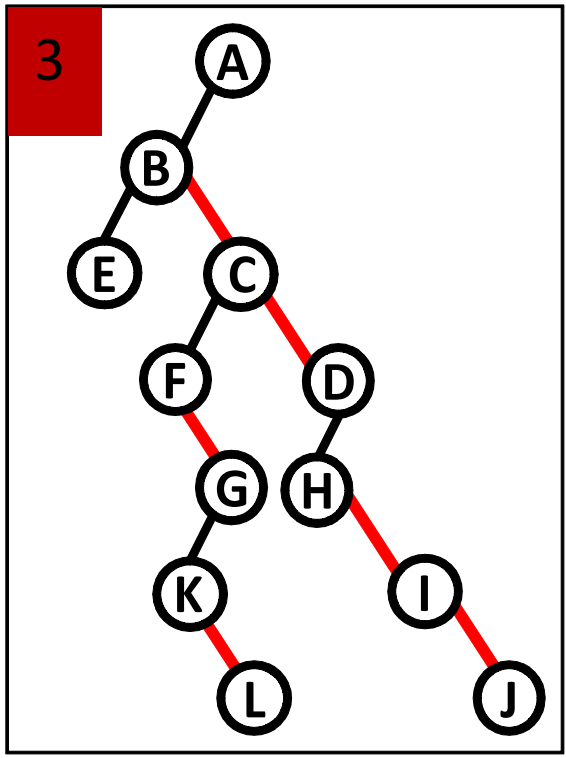
- En un árbol general no se puede cuantificar el número de descendientes directos que tendrá un nodo dado
- El espacio necesario para su representación deberá manejarse tal que:
  - a cada nodo se le permita un número variable de apuntadores
  - a cada nodo se le asigne un espacio para un número fijo de punteros, se necesiten todos o no
- **Técnica para convertir un árbol a binario:**
  1. Insertar aristas conectando a nodos hermanos
  2. Eliminar todas las aristas que conectan a un nodo con sus descendientes directos, excepto con el hijo de más a la izquierda
  3. Girar el diagrama resultante para distinguir entre los subárboles izquierdo y derecho

Ejemplo

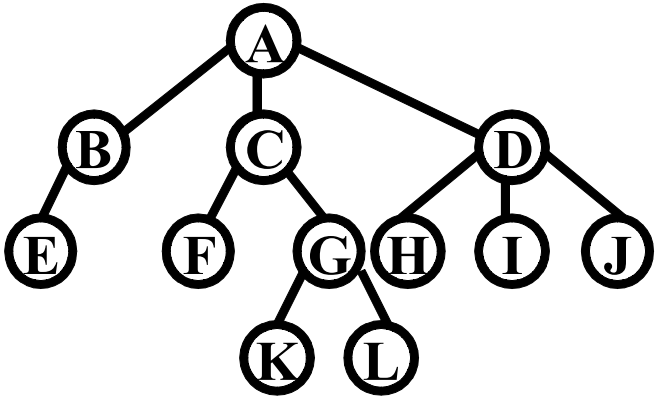

A
B
C
D
E
F
G
H
I
J
K
L

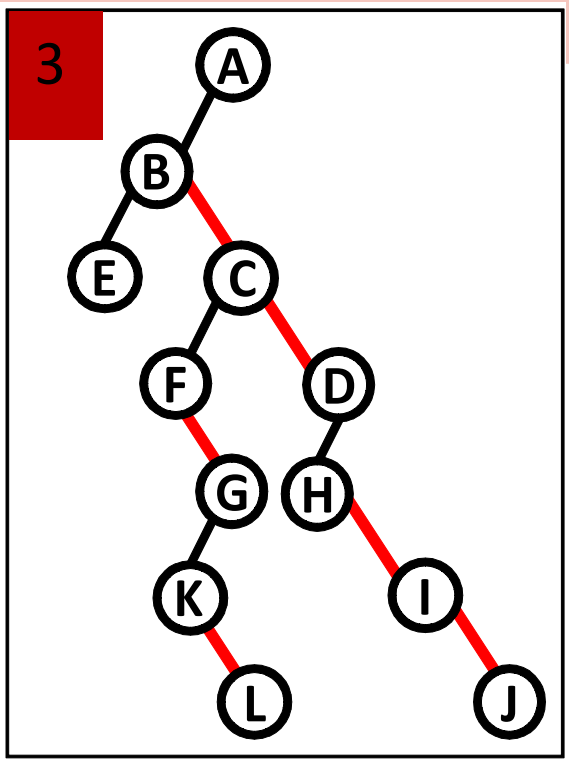

1
A
B
C
D
E
F
G
H
I
J
K
L


2
A
B
C
D
E
F
G
H
I
J
K
L


3
A
B
E
C
F
D
G
H
K
I
L
J

Resultado de la técnica de conversión





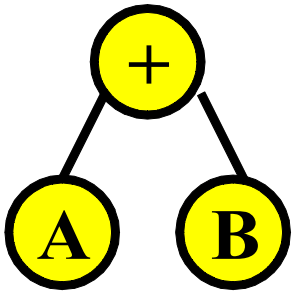
Árbol binario donde:

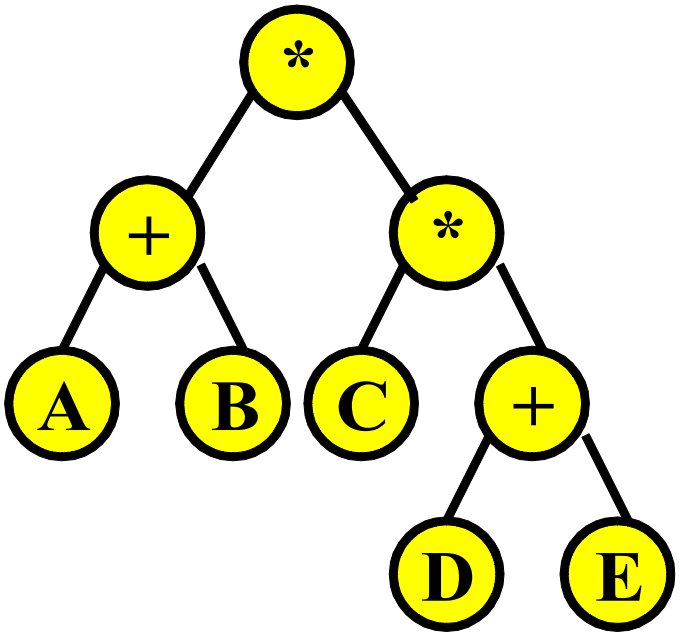
- Los punteros izquierdos van siempre de un nodo padre a su hijo de más a la izquierda en el árbol original
- Los punteros derechos van siempre de un nodo a uno de sus nodos hermanos en el árbol original
- Los hijos de cada nodo aparecen en en una lista enlazada de nodos de árbol

Siempre es posible, por tanto, reconstruir un árbol general a partir de su representación como árbol binario

Aplicaciones

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





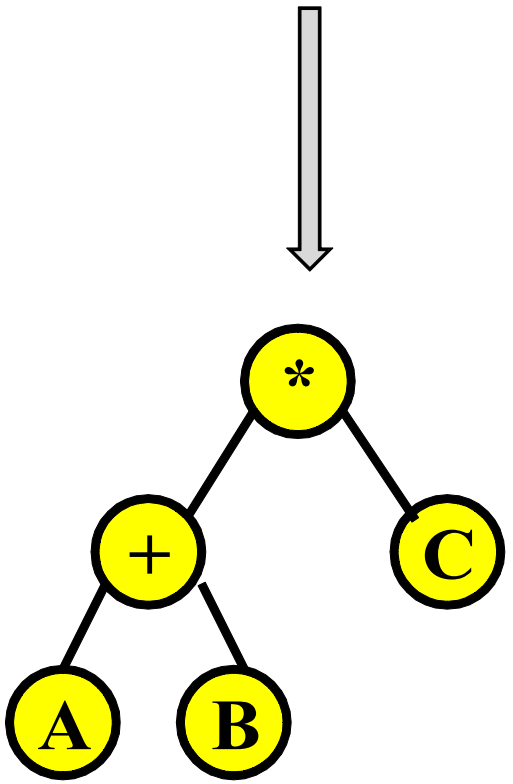
Prefija	Infija	Postfija
+AB	A+B	AB+
*+AB*C+DE	(A+B)*(C*(D+E))	AB+CDE+**

Ejemplo de aplicación: construcción de un árbol algebraico

```
función generarÁrbol (exPostfija: cadena):tipoÁrbol
1.  a: tipoÁrbol
2.  p: tipoPila
3.  i: entero
4.  símbolo : carácter
5.  creaVacía(p)
6.  símbolo ← ex[1]
7.  i ← 1
8.  mientras (símbolo ≠ FIN_EXPRESION) hacer
9.      caso símbolo en
10.       operando:      a ← creaNodo(símbolo)
11.                      inserta(a, p)
12.       operador:      a ← creaNodo(símbolo)
13.                      a↑ .der ← suprime(p)
14.                      a↑ .izq ← suprime(p)
15.                      inserta(a,p)
16.     fin caso
17.     i ← i + 1
18.     símbolo ← ex[i]
19. fin mientras
20. a ← suprime(p)
21. devolver a
```

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

A B + C *



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:
  1. en PROFUNDIDAD: basados en las relaciones padre-hijo de los nodos
  2. en AMPLITUD o por NIVELES: basados en la distancia de cada nodo a la raíz

Recorridos en PROFUNDIDAD

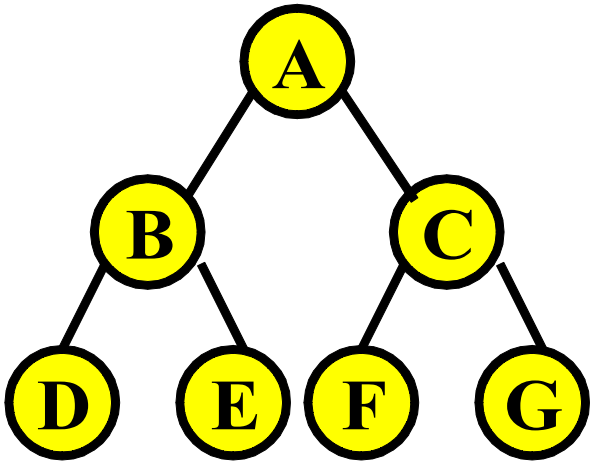
- Existen tres métodos diferentes de efectuar el recorrido en profundidad, todos ellos de naturaleza **recursiva**, imponiendo un orden secuencial y lineal a los nodos del árbol
- Las actividades básicas a realizar son:
  - visitar la raíz
  - recorrer el subárbol izquierdo
  - recorrer el subárbol derecho
- 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 en PRE-ORDEN

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

procedimiento preOrden(raíz : tipoÁrbol)

1. **si** raíz ≠ NULO **entonces**
2. visitar(raíz↑.información)
3. preOrden(raíz↑.izq)
4. preOrden(raíz↑.der)
5. **fin si**



A B D E C F G

raíz izq. der. raíz izq. der.

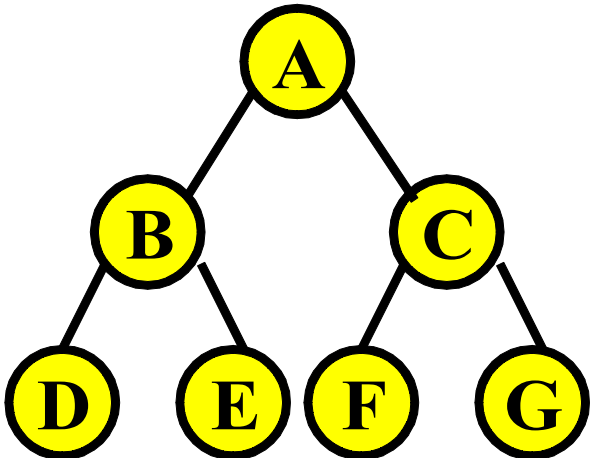
raíz **sub. izq. preorden** **sub. der. preorden**

Recorrido EN ORDEN

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

procedimiento enOrden(raíz : tipoÁrbol)

1. **si** raíz ≠ NULO **entonces**
2. enOrden(raíz↑.izq)
3. visitar(raíz↑.información)
4. enOrden(raíz↑.der)
5. **fin si**



D B E A F C G

izq. raíz der. izq. raíz der.

sub. izq. en orden raíz sub. der. en orden

Recorrido POST-ORDEN

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

procedimiento **postOrden(raíz : tipoÁrbol)**

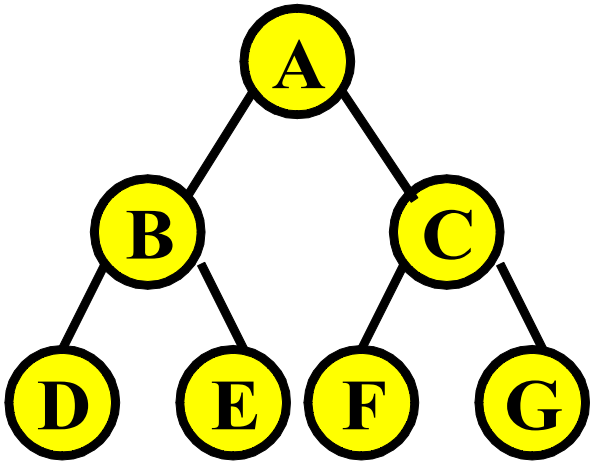
1. si raíz ≠ NULO **entonces**

2. **postOrden(raíz↑.izq)**

3. **postOrden(raíz↑.der)**

4. visitar(raíz↑.información)

5. fin si



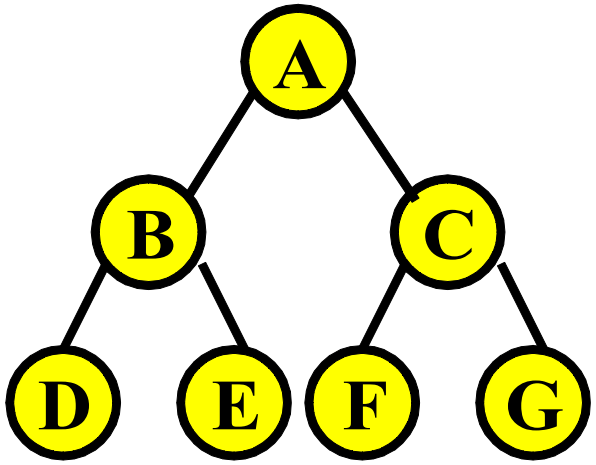
D E B F G C A

izq. der. raíz izq. der. raíz

sub. izq. Postorden **sub. der. Postorden** **raíz**

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



A BC DEFG

nivel 0 nivel 1 nivel 2

Algoritmo de recorrido en AMPLITUD

```
procedimiento amplitud (raíz : tipoÁrbol)
1.  c: tipoCola
2.  nodo : punteroNodo
3.  creaVacía(c)
4.  nodo ← raíz
5.  si nodo ≠ NULO entonces
6.    inserta(nodo, c)
7.  fin si
8.  mientras NOT(vacía(c)) hacer
9.     nodo ← suprime(c)
10.    visitar(nodo↑.información)
11.    si nodo↑.izq ≠ NULO entonces
12.           inserta(nodo↑.izq, c)
13.    fin si
14.    si nodo↑.der ≠ NULO entonces
15.           inserta(nodo↑.der, c)
16.    fin si
17. fin mientras
```

Observar 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 pila ***c*** como una cola vacía, sin ningún elemento

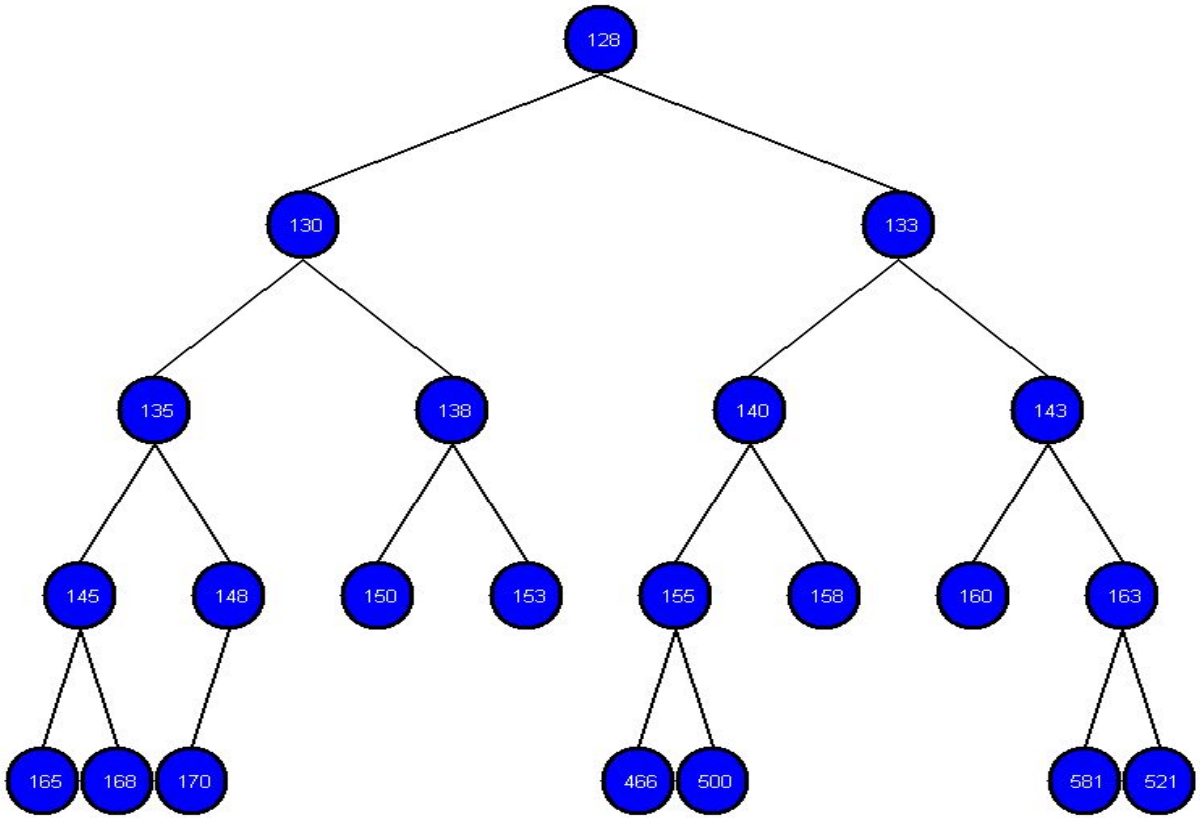
  vacía(c): devuelve verdadero si la cola ***c*** está vacía, y falso en caso contrario

  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***

Ejercicios sobre recorridos

1. Utilizando la aplicación RAED Representación de Algoritmos de Estructuras de Datos (http://raed.usal.es) analizar el comportamiento de los algoritmos de recorridos sobre diferentes ejemplos. Se dejan dos ficheros creados con la aplicación raed que pueden descargarse para ser utilizados con la misma. Uno de ellos se corresponde con el árbol que muestra la siguiente figura y el otro es similar a los árboles presentados en las transparencias de recorridos.



Ejercicios sobre recorridos

1. Dibujar **razonadamente** un árbol binario sabiendo su recorrido en preorden y en orden:
   - Preorden: 1234. En orden: 1342
   - Preorden: 1234. En orden: 3421
   - Preorden: 4321. En orden: 3124
   - Preorden: 4321. En orden: 4213
   - Preorden: ESTRUCTURAS. En orden: RTUSECUTARS
   - Preorden: CRSETUTUARS. En orden: ESTRUCTURAS