



Algoritmos y Estructuras de Datos

Cursada 2021

Prof. Alejandra Schiavoni (ales@info.unlp.edu.ar)

Prof. Catalina Mostaccio (catty@lifa.info.unlp.edu.ar)

Prof. Laura Fava (lfava@info.unlp.edu.ar)

Prof. Pablo Iuliano (piuliano@info.unlp.edu.ar)

Árboles Binarios

Agenda

- Definición
- Descripción y terminología
- Representaciones
- Recorridos
- Aplicación: Árboles de expresión

Árbol Binario: Definición

➤ *Un árbol binario es una colección de nodos, tal que:*

- *puede estar vacía*
- *puede estar formada por un nodo distinguido R , llamado **raíz**, y dos sub-árboles T_1 y T_2 , donde la raíz de cada subárbol T_i está conectado a R por medio de una arista*

Descripción y terminología

- Cada nodo puede tener a lo sumo dos nodos hijos.
- Cuando un nodo no tiene ningún hijo se denomina *hoja*.
- Los nodos que tienen el mismo nodo padre se denominan *hermanos*.

Descripción y terminología

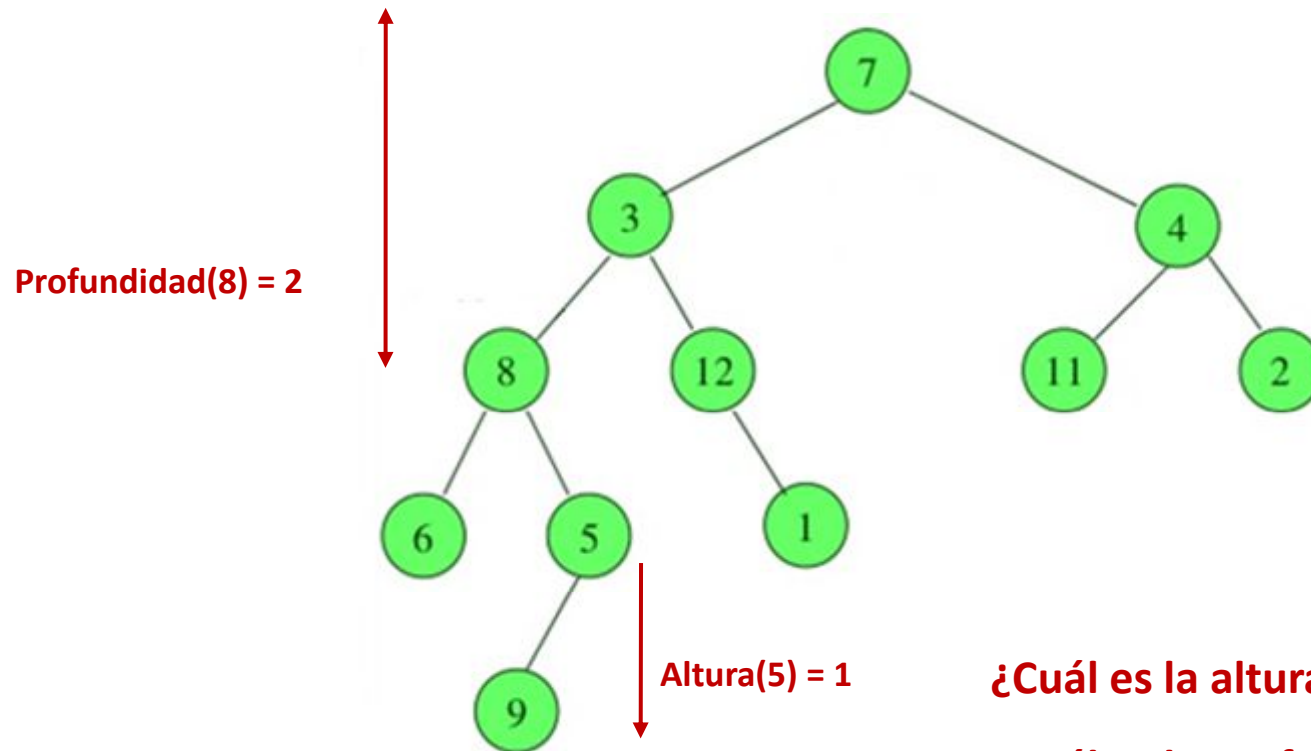
➤ Conceptos a usar:

- **Camino:** desde n_1 hasta n_k , es una secuencia de nodos n_1, n_2, \dots, n_k tal que n_i es el padre de n_{i+1} , para $1 \leq i < k$.
 - La longitud del camino es el número de aristas, es decir $k-1$.
 - Existe un camino de longitud cero desde cada nodo a sí mismo.
 - Existe un único camino desde la raíz a cada nodo.
- **Profundidad:** de n_i es la longitud del único camino desde la raíz hasta n_i .
 - La raíz tiene profundidad cero.

Descripción y terminología

- **Grado** de n_i es el número de hijos del nodo n_i .
- **Altura** de n_i es la longitud del camino más largo desde n_i hasta una hoja.
 - Las hojas tienen altura cero.
 - La altura de un árbol es la altura del nodo raíz.
- **Ancestro/Descendiente**: si existe un camino desde n_1 a n_2 , se dice que n_1 es ancestro de n_2 y n_2 es descendiente de n_1 .

Descripción y terminología

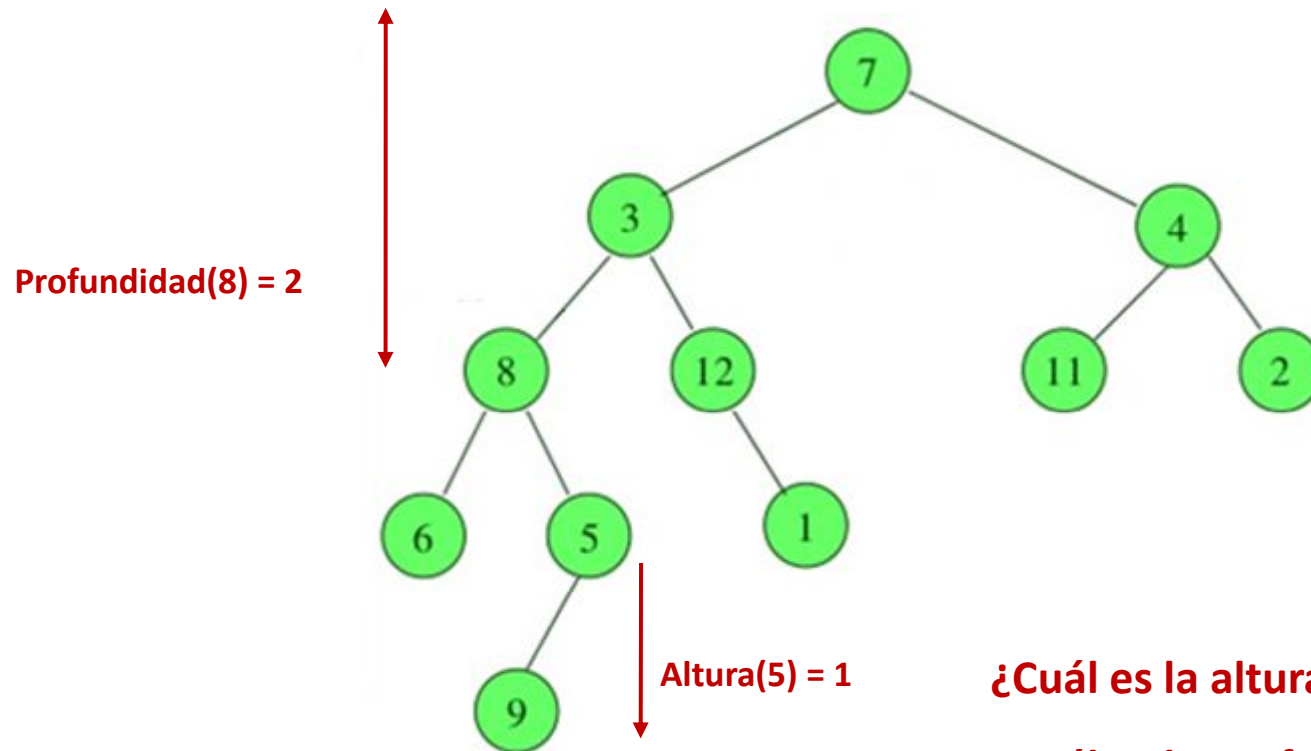


¿Cuál es la altura del nodo 8?

¿Cuál es la profundidad del nodo 12?

¿Cuáles son los ancestros del nodo 11?

Descripción y terminología



¿Cuál es la altura del nodo 8? 2

¿Cuál es la profundidad del nodo 12? 2

¿Cuáles son los ancestros del nodo 11? 7 y 4

Descripción y terminología

- *Árbol binario lleno*: Dado un árbol binario T de altura h , diremos que T es *lleno* si cada nodo interno tiene grado 2 y todas las hojas están en el mismo nivel (h).

Es decir, recursivamente, T es *lleno* si :

- 1.- T es un nodo simple (árbol binario lleno de altura 0), o
- 2.- T es de altura h y sus sub-árboles son llenos de altura $h-1$.

Descripción y terminología

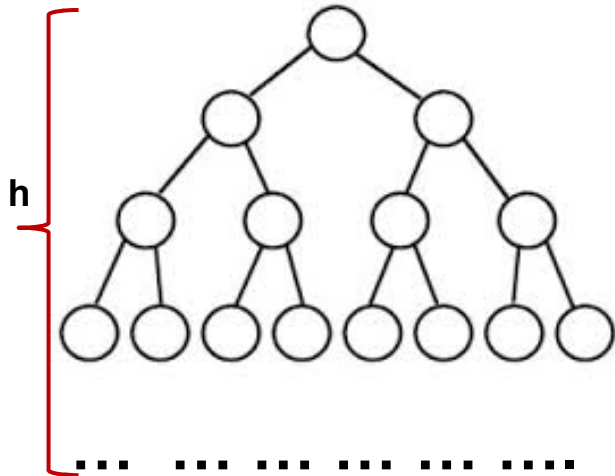
- *Cantidad de nodos en un árbol binario lleno:*

Sea T un árbol binario lleno de altura h , la cantidad de nodos N es $(2^{h+1} - 1)$

Descripción y terminología

- *Cantidad de nodos en un árbol binario lleno:*

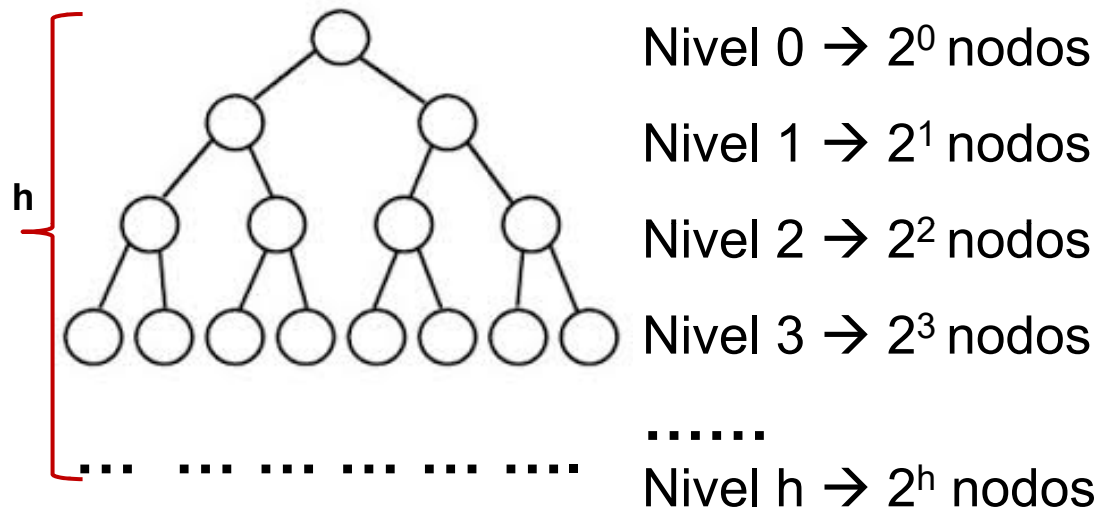
Sea T un árbol binario lleno de altura h , la cantidad de nodos N es $(2^{h+1} - 1)$



Descripción y terminología

- *Cantidad de nodos en un árbol binario lleno:*

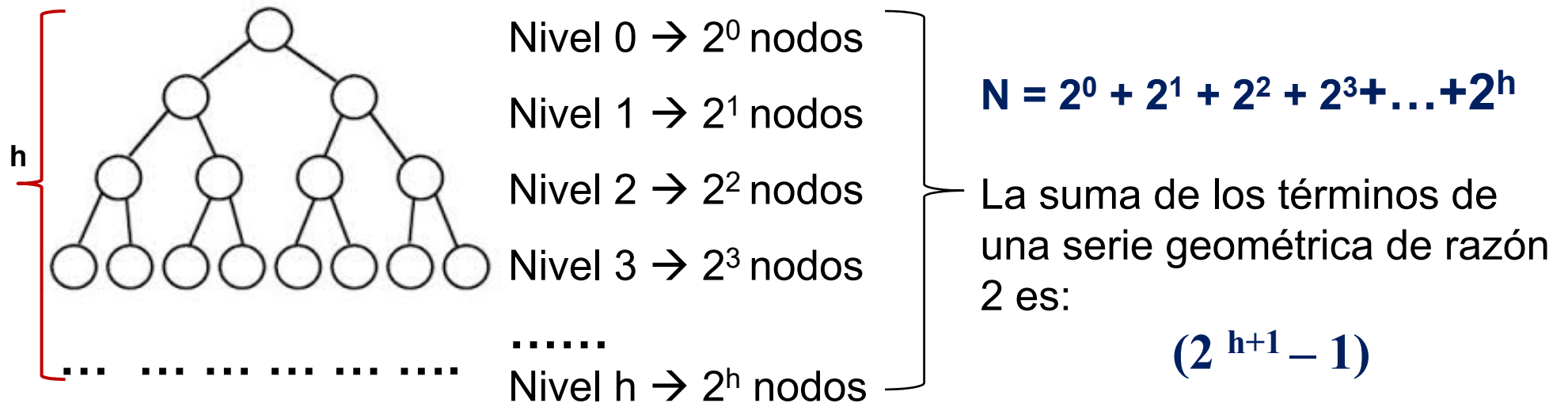
Sea T un árbol binario lleno de altura h , la cantidad de nodos N es $(2^{h+1} - 1)$



Descripción y terminología

- Cantidad de nodos en un árbol binario lleno:*

Sea T un árbol binario lleno de altura h , la cantidad de nodos N es $(2^{h+1} - 1)$

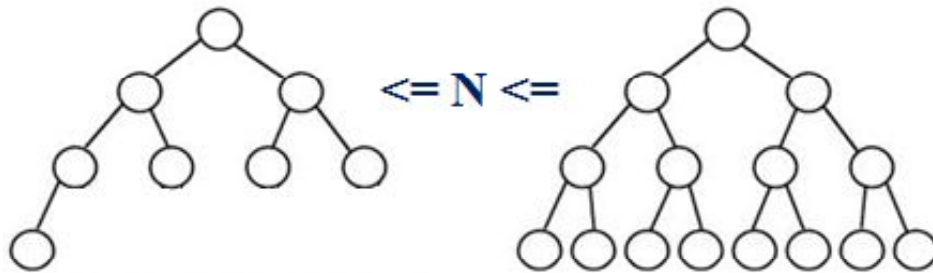


Descripción y terminología

- *Árbol binario completo*: Dado un árbol binario T de altura h , diremos que T es completo si es lleno de altura $h-1$ y el nivel h se completa de izquierda a derecha.
- *Cantidad de nodos en un árbol binario completo*:
Sea T un árbol binario completo de altura h , la cantidad de nodos N varía entre (2^h) y $(2^{h+1} - 1)$

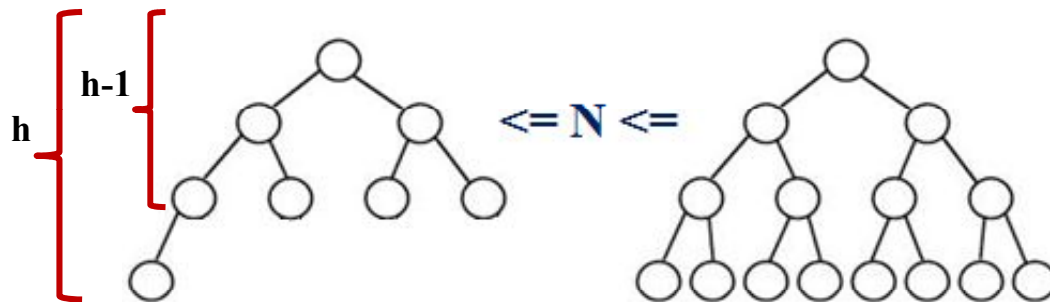
Descripción y terminología

- **Árbol binario completo:** Dado un árbol binario T de altura h , diremos que T es completo si es lleno de altura $h-1$ y el nivel h se completa de izquierda a derecha.
- **Cantidad de nodos en un árbol binario completo:**
Sea T un árbol binario completo de altura h , la cantidad de nodos N varía entre (2^h) y $(2^{h+1} - 1)$



Descripción y terminología

- **Árbol binario completo:** Dado un árbol binario T de altura h , diremos que T es completo si es lleno de altura $h-1$ y el nivel h se completa de izquierda a derecha.
- **Cantidad de nodos en un árbol binario completo:**
Sea T un árbol binario completo de altura h , la cantidad de nodos N varía entre (2^h) y $(2^{h+1} - 1)$

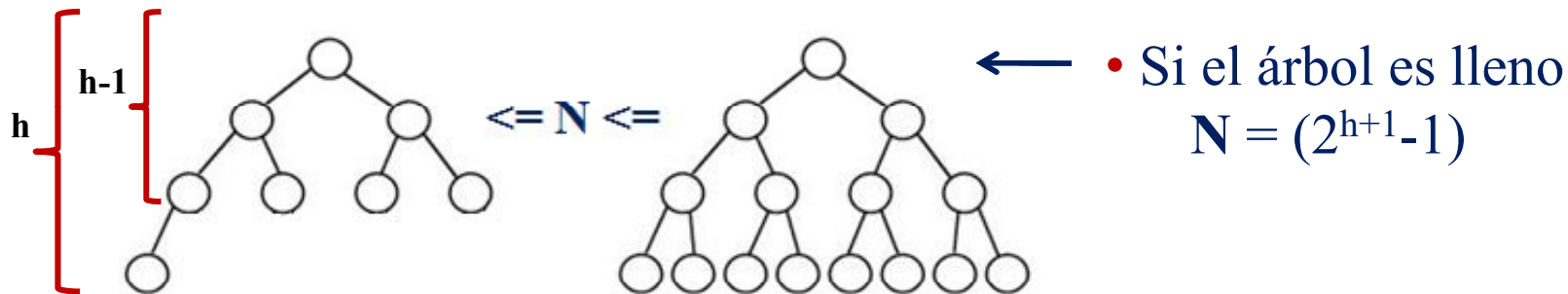


Descripción y terminología

- **Árbol binario completo:** Dado un árbol binario T de altura h , diremos que T es completo si es lleno de altura $h-1$ y el nivel h se completa de izquierda a derecha.

- **Cantidad de nodos en un árbol binario completo:**

Sea T un árbol binario completo de altura h , la cantidad de nodos N varía entre (2^h) y $(2^{h+1} - 1)$

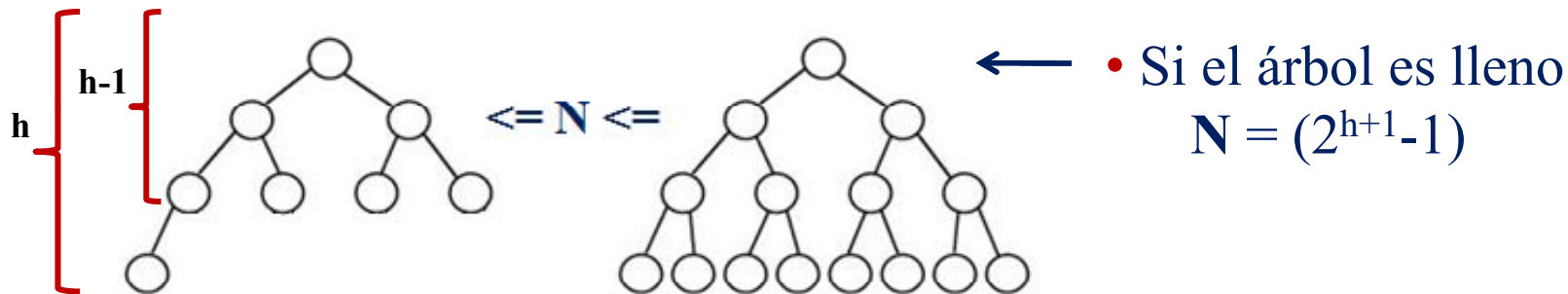


Descripción y terminología

- **Árbol binario completo:** Dado un árbol binario T de altura h , diremos que T es completo si es lleno de altura $h-1$ y el nivel h se completa de izquierda a derecha.

- **Cantidad de nodos en un árbol binario completo:**

Sea T un árbol binario completo de altura h , la cantidad de nodos N varía entre (2^h) y $(2^{h+1} - 1)$



- Si no, el árbol es lleno en la altura $h-1$ y tiene por lo menos un nodo en el nivel h :
 $N = (2^{h-1+1}-1)+1=(2^h-1 + 1)$

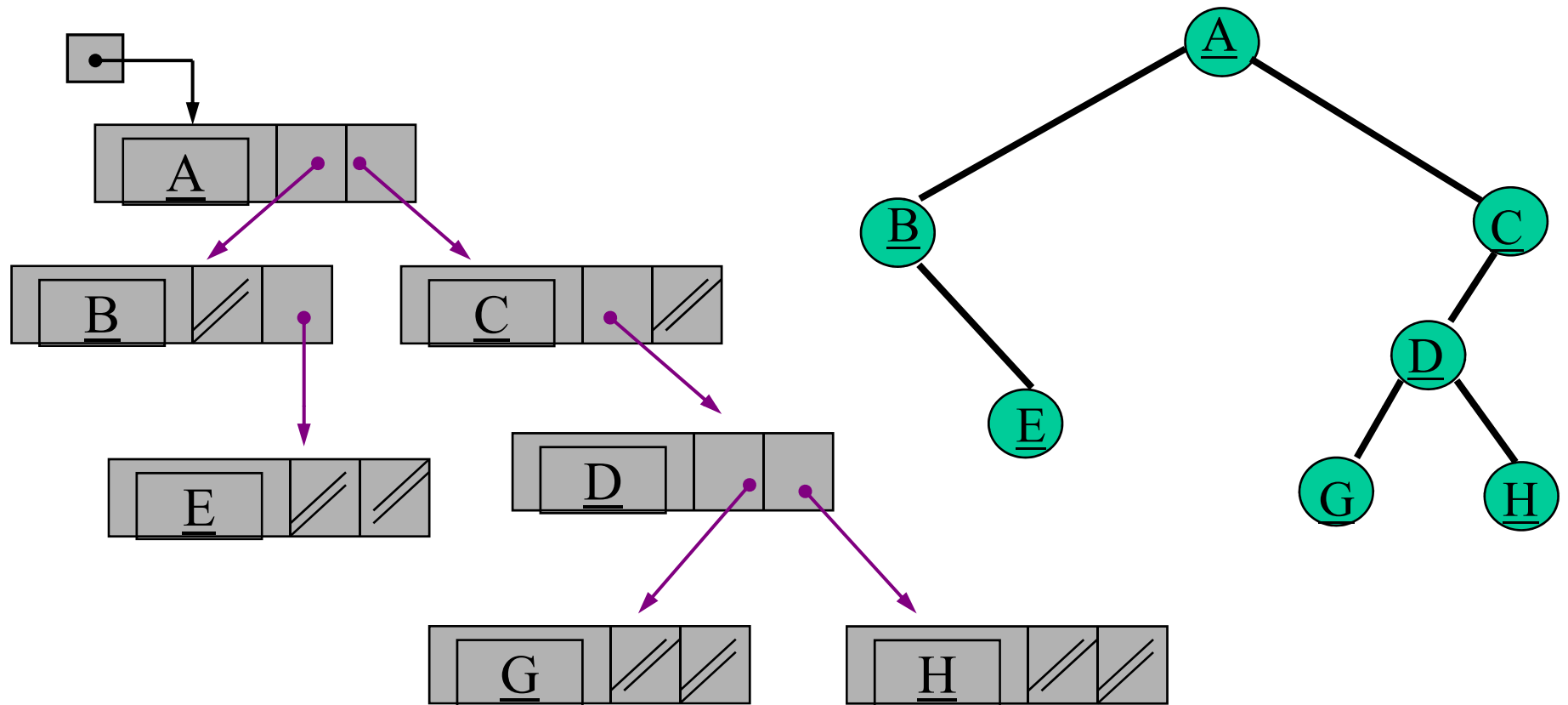
Representación

Hijo Izquierdo - Hijo Derecho

- ✓ Cada nodo tiene:
 - Información propia del nodo
 - Referencia a su hijo izquierdo
 - Referencia a su hijo derecho

Representación

Hijo Izquierdo - Hijo Derecho



Recorridos



Preorden

Se procesa primero la raíz y luego sus hijos, izquierdo y derecho.



Inorden

Se procesa el hijo izquierdo, luego la raíz y último el hijo derecho



Postorden

Se procesan primero los hijos, izquierdo y derecho, y luego la raíz



Por niveles

Se procesan los nodos teniendo en cuenta sus niveles, primero la raíz, luego los hijos, los hijos de éstos, etc.

Recorrido: Preorden

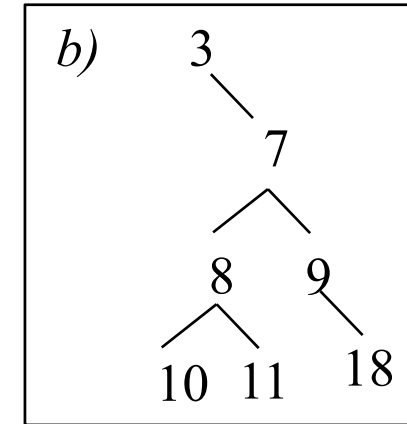
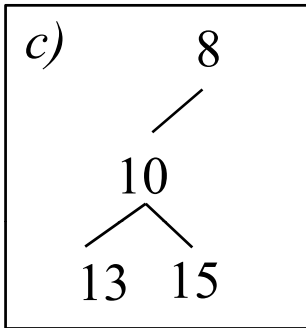
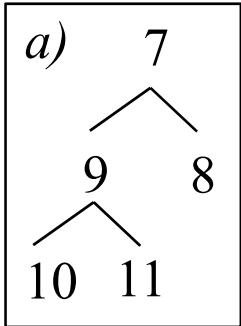
```
public void preorden() {  
    imprimir (dato);  
    si (tiene hijo_izquierdo)  
        hijoIzquierdo.preorden();  
    si (tiene hijo_derecho)  
        hijoDerecho.preorden();  
}
```

Recorrido: Por niveles

```
public void porNiveles() {  
    encolar(raíz);  
    mientras (cola no se vacíe) {  
        desencolar(v);  
        imprimir (dato de v);  
        si (tiene hijo_izquierdo)  
            encolar(hijo_izquierdo);  
        si (tiene hijo_derecho)  
            encolar(hijo_derecho);  
    }  
}
```

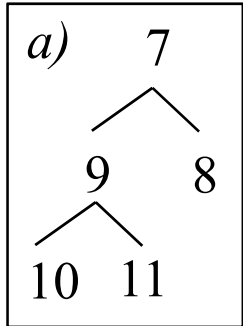

Ejercitación Árbol binario: Recorridos

Ejercicio 1



Ejercitación Árbol binario: Recorridos

Ejercicio 1

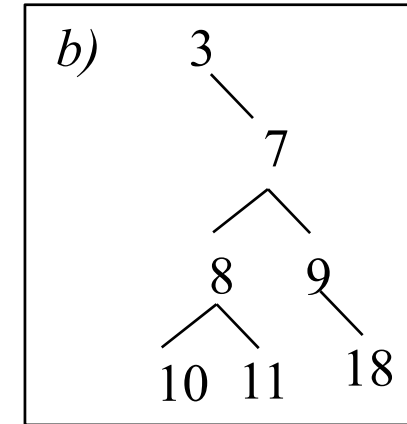
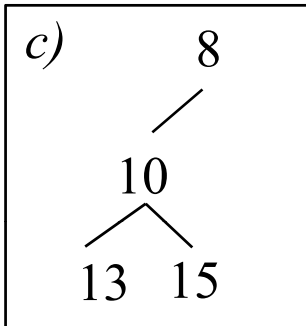


a)

✓ *inorden* : 10 9 11 7 8

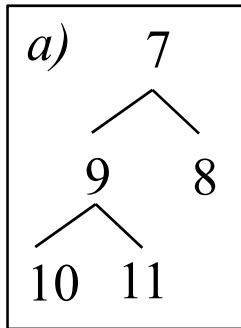
✓ *postorden* : 10 11 9 8 7

✓ *preorden* : 7 9 10 11 8



Ejercitación Árbol binario: Recorridos

Ejercicio 1

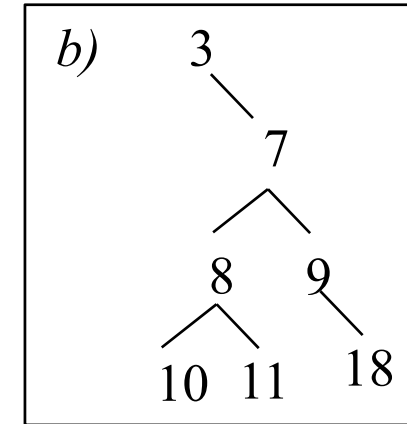
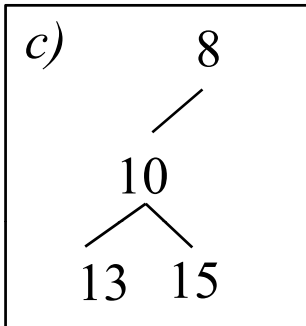


a)

✓ *inorden* : 10 9 11 7 8

✓ *postorden* : 10 11 9 8 7

✓ *preorden* : 7 9 10 11 8



b)

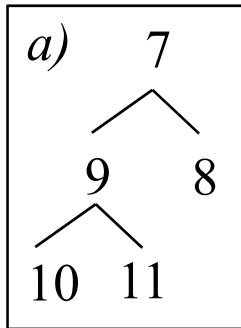
✓ *inorden* : 3 10 8 11 7 9 18

✓ *postorden* : 10 11 8 18 9 7 3

✓ *preorden* : 3 7 8 10 11 9 18

Ejercitación Árbol binario: Recorridos

Ejercicio 1

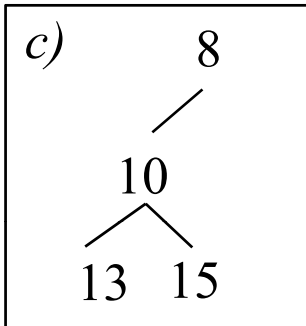


a)

✓ *inorden* : 10 9 11 7 8

✓ *postorden* : 10 11 9 8 7

✓ *preorden* : 7 9 10 11 8

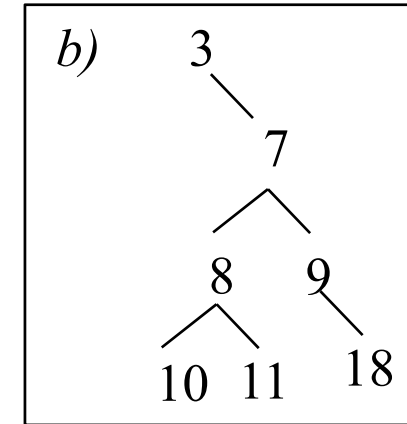


c)

✓ *inorden* : 13 10 15 8

✓ *postorden* : 13 15 10 8

✓ *preorden* : 8 10 13 15



b)

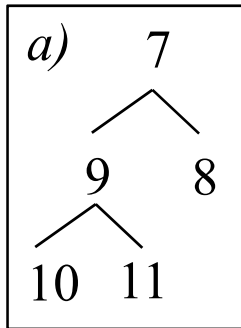
✓ *inorden* : 3 10 8 11 7 9 18

✓ *postorden* : 10 11 8 18 9 7 3

✓ *preorden* : 3 7 8 10 11 9 18

Ejercitación Árbol binario: Recorridos

Ejercicio 1

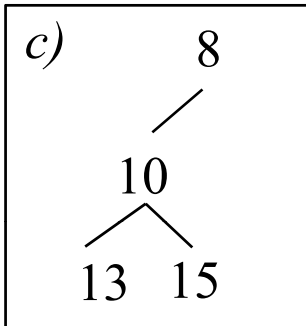


a)

✓ inorden : 10 9 11 7 8

✓ postorden : 10 11 9 8 7

✓ preorden : 7 9 10 11 8

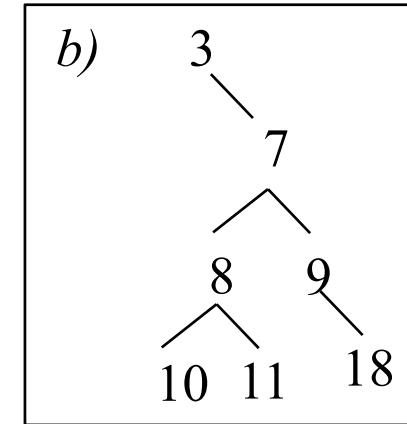


c)

✓ inorden : 13 10 15 8

✓ postorden : 13 15 10 8

✓ preorden : 8 10 13 15



b)

✓ inorden : 3 10 8 11 7 9 18

✓ postorden : 10 11 8 18 9 7 3

✓ preorden : 3 7 8 10 11 9 18

Ejercicio 2

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:

inorden: **C B F E G A D I H** y postorden: **C F G E B I H D A**

Ejercitación

Árbol binario: Recorridos

Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : C B F E G A D I H y postorden : C F G E B I H D A

Resolución:

inorden : C B F E G A D I H y postorden : C F G E B I H D A

¿Por dónde empezamos?

¿Qué información podemos obtener de los recorridos dados?

¿De qué estamos seguros?

Ejercitación


Árbol binario: Recorridos

Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : C B F E G A D I H y postorden : C F G E B I H D A

Resolución:

inorden : C B F E G A D I H y postorden : C F G E B I H D A



Raíz

¿ Cómo seguimos ?

Ejercitación

Árbol binario: Recorridos

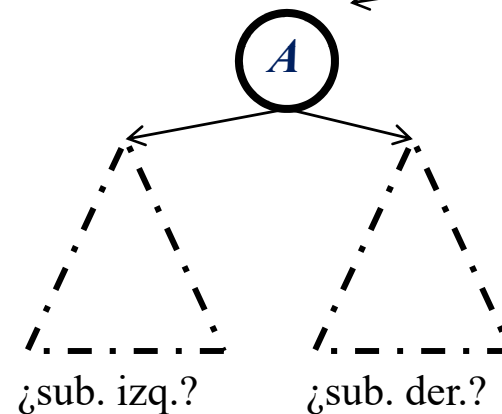
Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : **C B F E G A D I H** y *postorden* : **C F G E B I H D A**

Resolución:

inorden : **C B F E G A D I H** y *postorden* : **C F G E B I H D** **A**
Raíz

¿Cómo armamos los subárboles?
¿Qué información podemos
obtener de los recorridos dados?



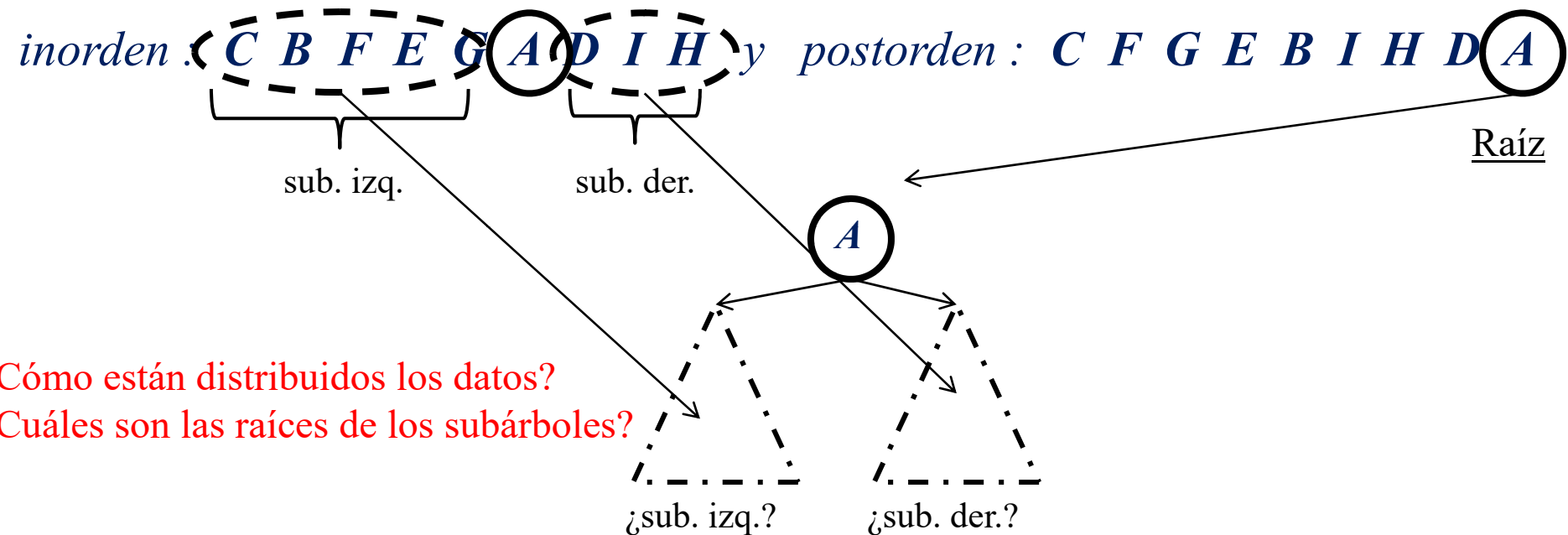
Ejercitación

Árbol binario: Recorridos

Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : **C B F E G A D I H** y *postorden* : **C F G E B I H D A**

Resolución:



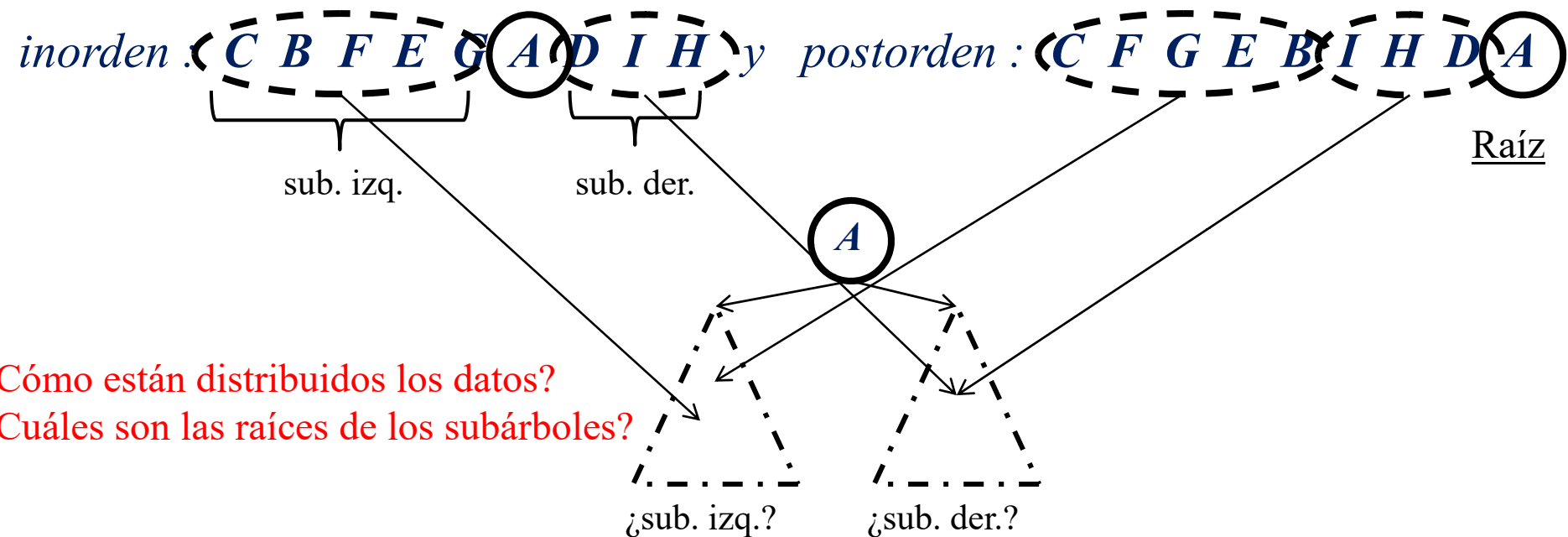
Ejercitación

Árbol binario: Recorridos

Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : **C B F E G A D I H** y *postorden* : **C F G E B I H D A**

Resolución:



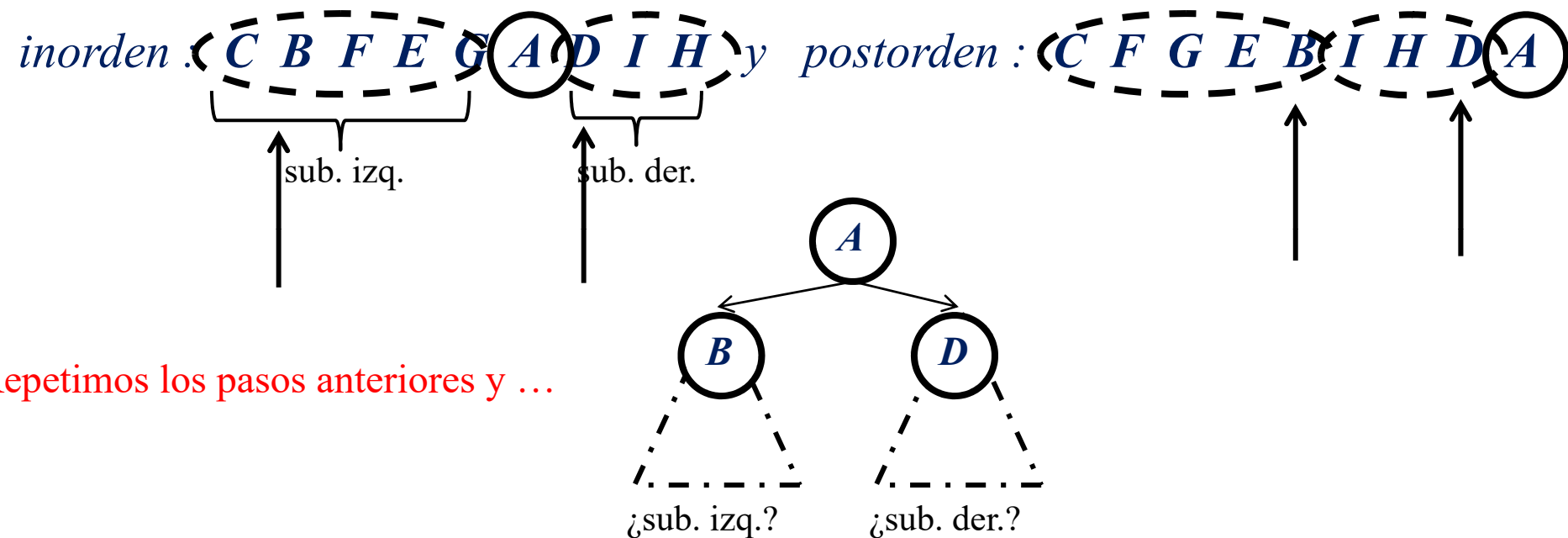
Ejercitación

Árbol binario: Recorridos

Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : **C B F E G A D I H** y postorden : **C F G E B I H D A**

Resolución:



Repetimos los pasos anteriores y ...

Ejercitación

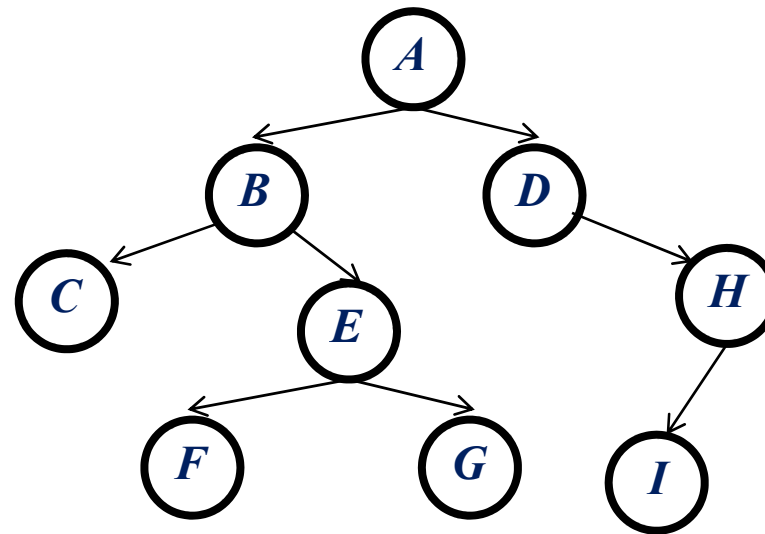
Árbol binario: Recorridos

Ejercicio 2.

Construya el árbol binario a partir del cual se obtuvieron los siguientes recorridos:
inorden : C B F E G A D I H y postorden : C F G E B I H D A

Resolución:

inorden : C B F E G A D I H y postorden : C F G E B I H D A



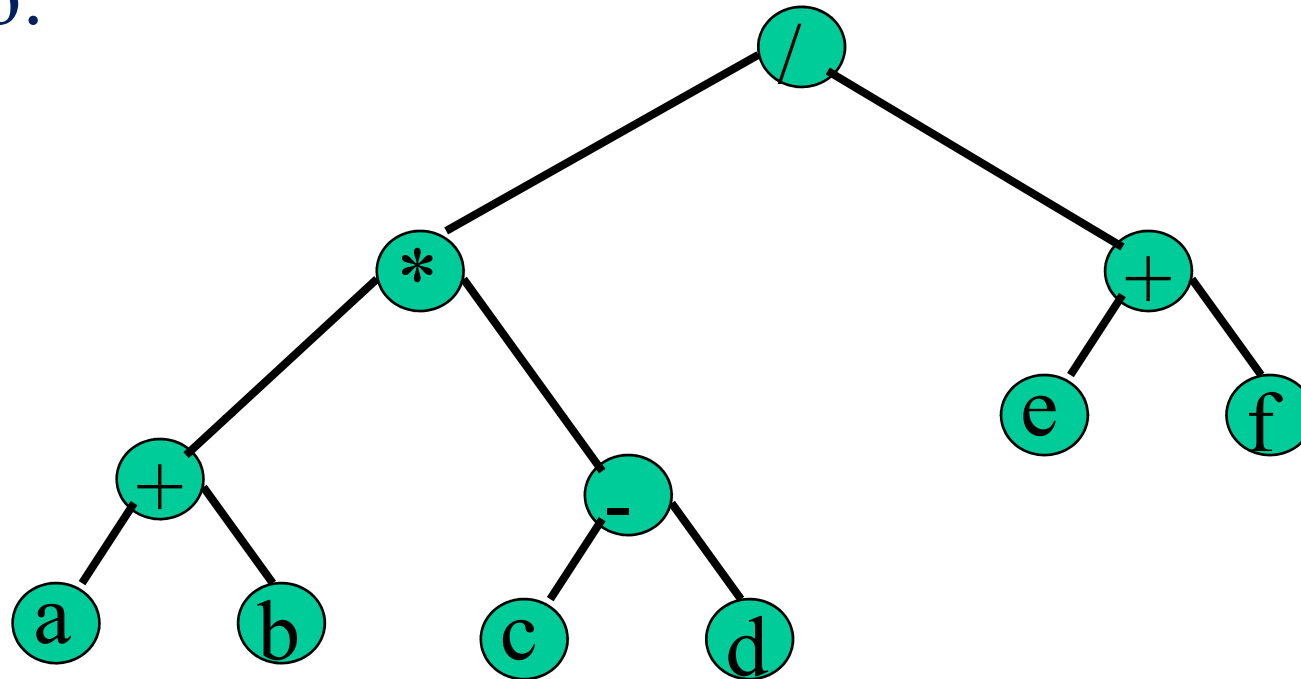
Árbol de Expresión

Es un árbol binario asociado a una expresión aritmética

- Nodos internos representan operadores
- Nodos externos (hojas) representan operandos

Árbol de Expresión

Ejemplo:



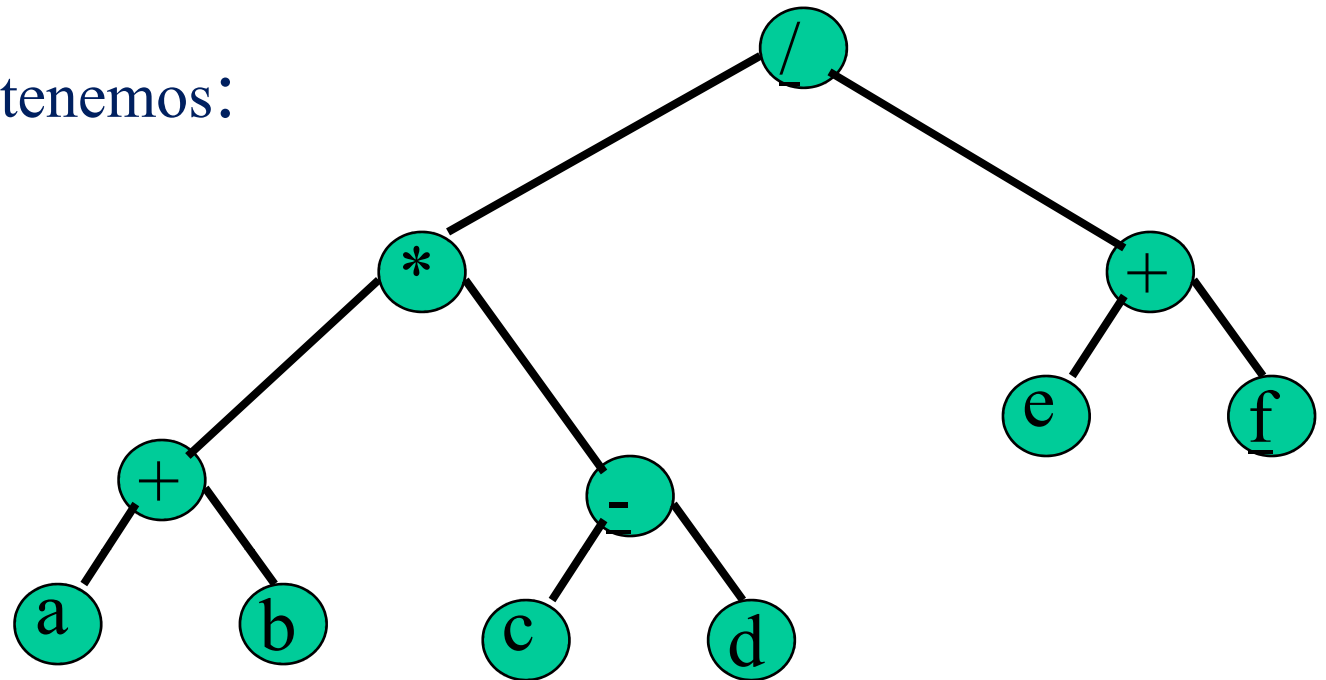
Árbol de Expresión

Aplicaciones:

- En compiladores para analizar, optimizar y traducir programas
- Evaluar expresiones algebraicas o lógicas
 - No se necesita el uso de paréntesis
- Traducir expresiones a notación sufija, prefija e infija

Árbol de Expresión

Recorriendo el árbol, obtenemos:



Inorden: $((a + b) * (c - d)) / (e + f)$

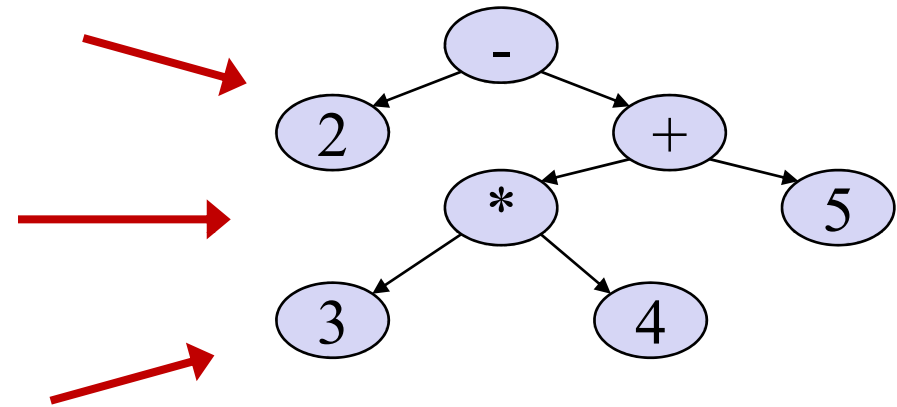
Preorden: $/*+ab-cd+ef$

Postorden: $ab+cd-*ef+ /$

Construcción de un árbol de expresión

A partir de una:

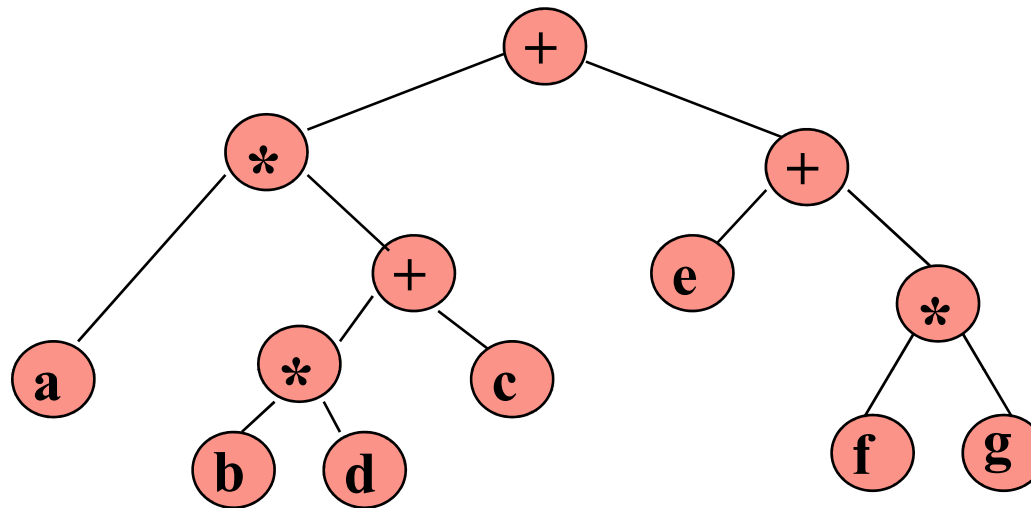
- 1) Expresión postfija
- 2) Expresión prefija
- 3) Expresión infija



Árboles binarios de expresión

Expresión algebraica :

$$a * (b * d + c) + (e + f * g)$$



Expresión **prefija** $\longrightarrow + * a + * b d c + e * f g$

Expresión **postfija** $\longrightarrow a b d * c + * e f g * + +$

Expresión **infija** $\longrightarrow ((a * ((b * d) + c)) + (e + (f * g)))$

1) Construcción de un árbol de expresión a partir de una expresión postfija

Algoritmo:

*tomo un carácter de la expresión
mientras (existe carácter) hacer*

*si es un **operando** → creo un nodo y lo apilo.*

*si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)*

*→ - **creo** un nodo R ,*

*- **desapilo** y lo agrego como hijo derecho de R*

*- **desapilo** y lo agrego como hijo izquierdo de R*

*- **apilo** R .*

tomo otro carácter

fin

1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ 

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo apilo.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

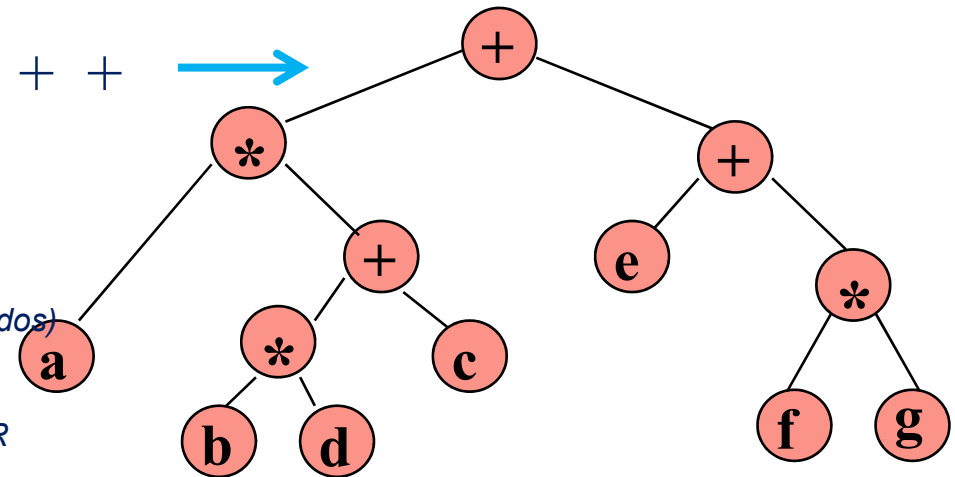
- **desapilo** y lo agrego como hijo derecho de R

- **desapilo** y lo agrego como hijo izquierdo de R

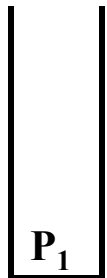
- **apilo** R .

tomo otro carácter

fin



$a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$



P_1

1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ 

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo apilo.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

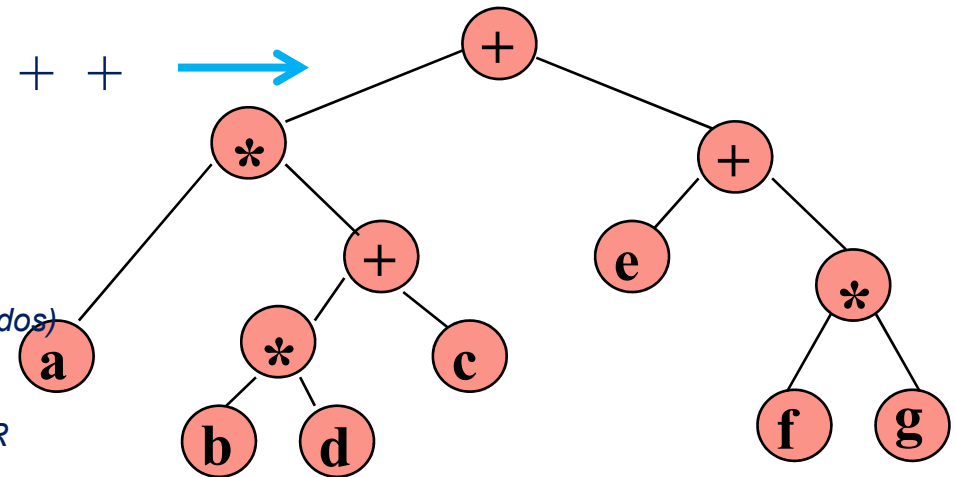
- **desapilo** y lo agrego como hijo derecho de R

- **desapilo** y lo agrego como hijo izquierdo de R

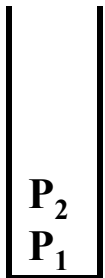
- **apilo** R .

tomo otro carácter

fin



~~a~~ b d * c + * e f g * + +



a

P_1

b

P_2

1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ \rightarrow

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo apilo.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

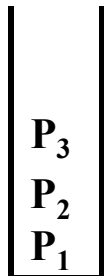
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d$~~ $*\ c\ +\ *\ e\ f\ g\ *\ +\ +$



a

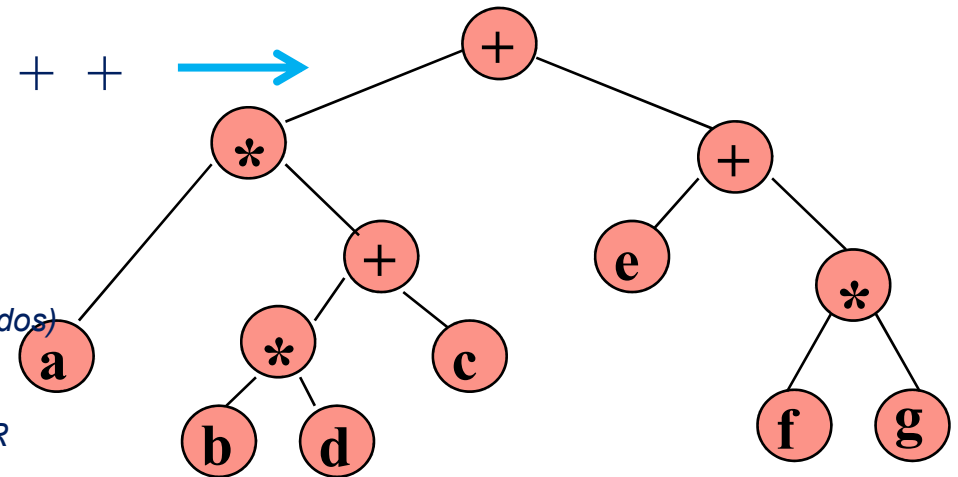
P_1

b

P_2

d

P_3



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ \rightarrow

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

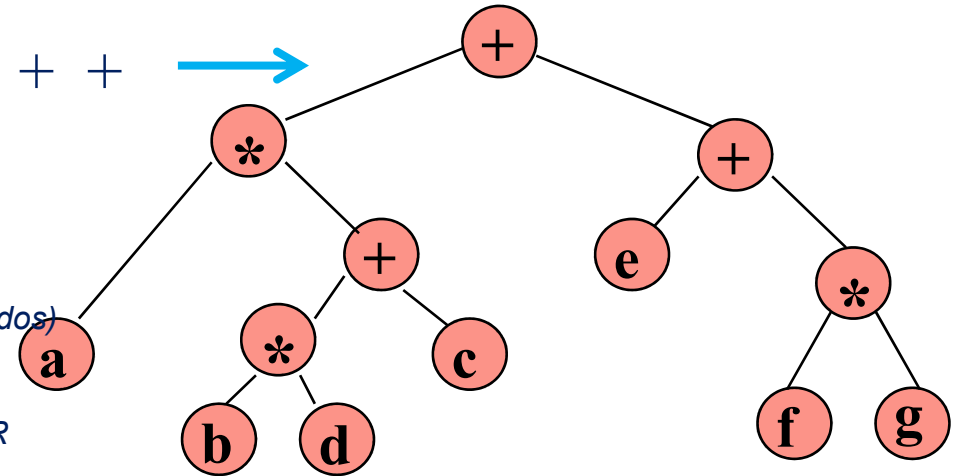
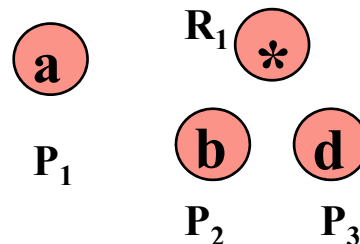
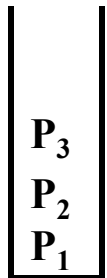
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d$~~ $*\ c\ +\ *\ e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ \rightarrow

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

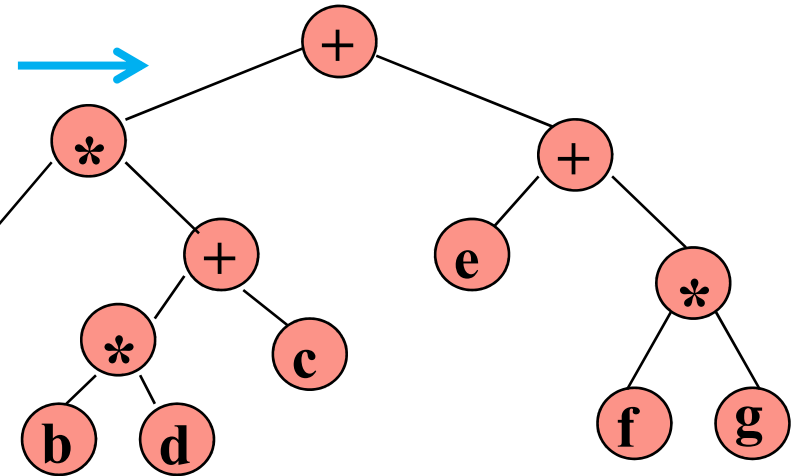
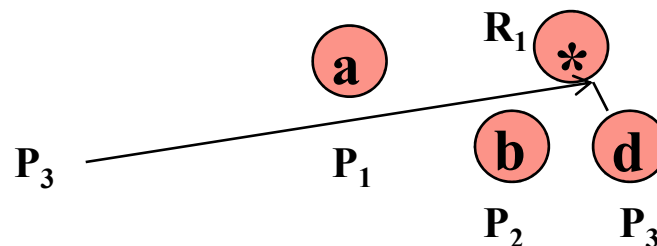
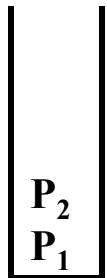
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d$~~ $*\ c\ +\ *\ e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** → **creo** un nodo y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

→ - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

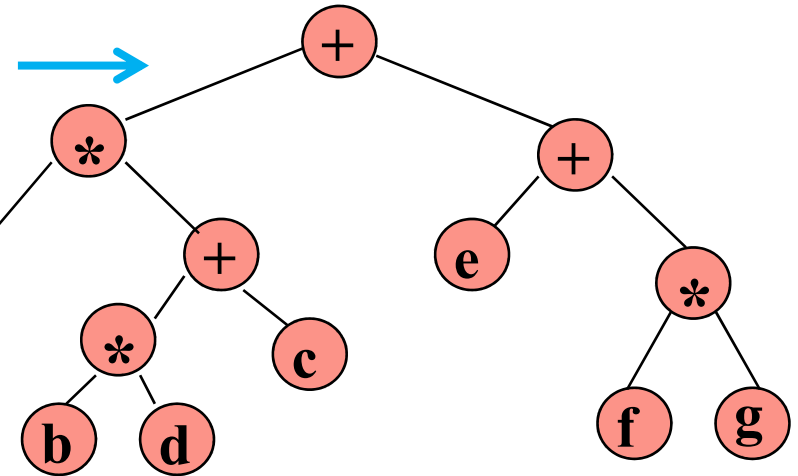
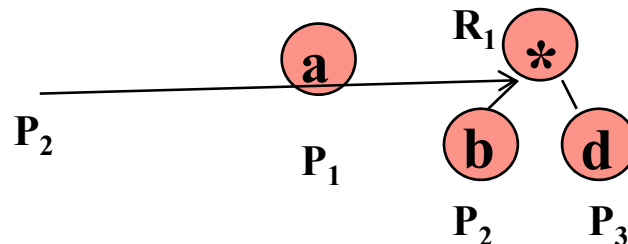
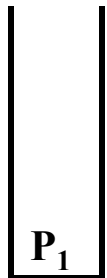
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d$~~ $\ * \ c \ + \ *\ e \ f \ g \ *\ + \ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ \rightarrow

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

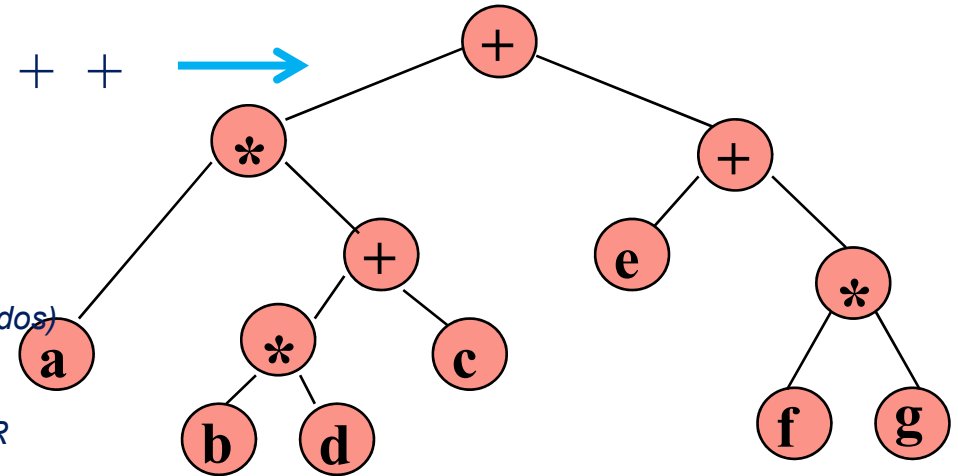
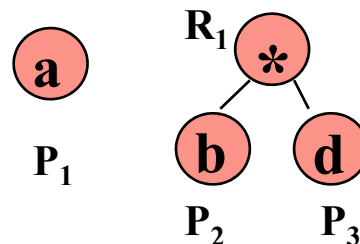
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d$~~ $*\ c\ +\ *\ e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** → **creo** un nodo y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

→ - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

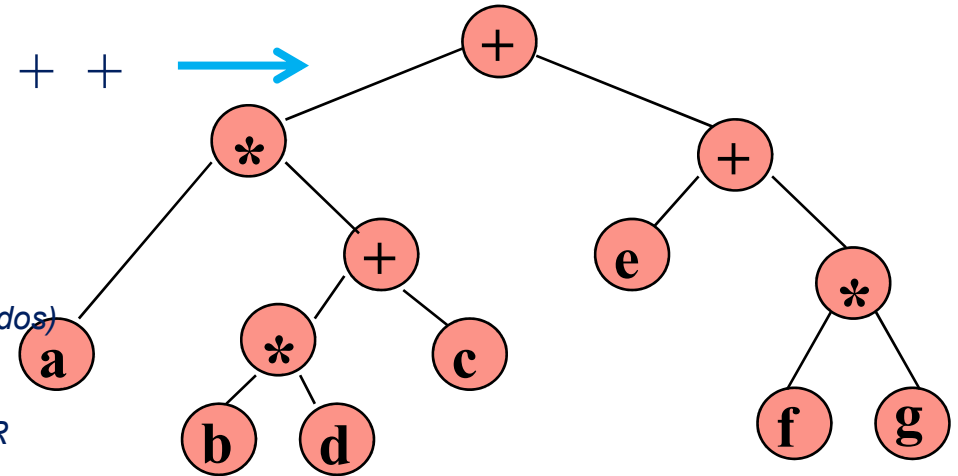
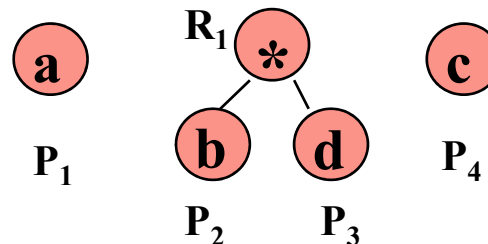
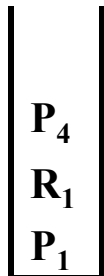
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d\ *$~~ $c\ +\ *\ e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ \rightarrow

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo apilo.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

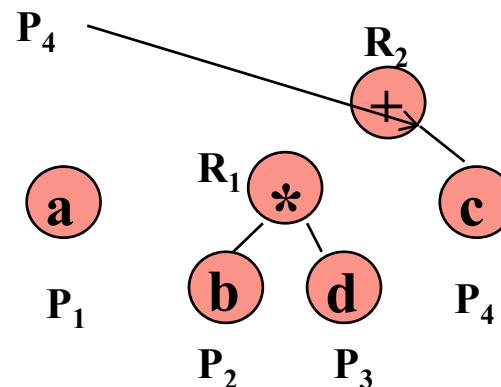
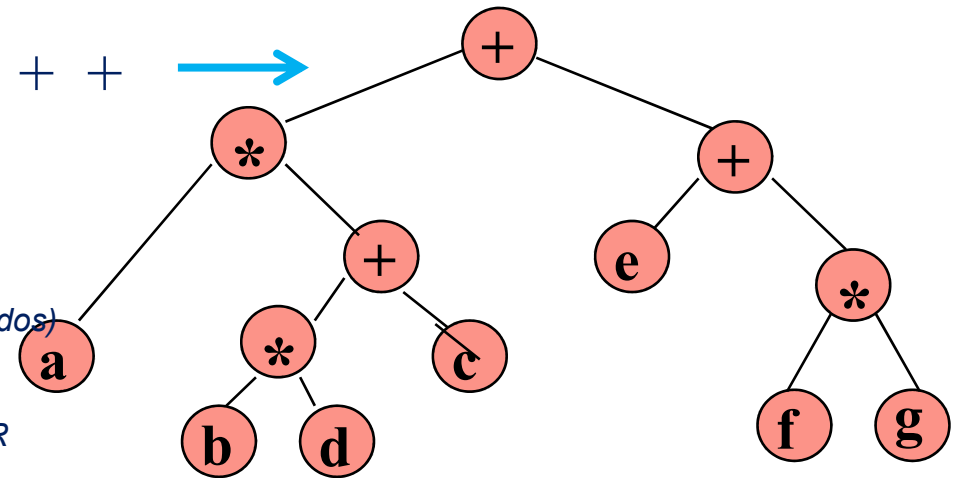
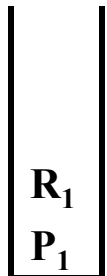
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d\ *\ c$~~ $+ \ *\ e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** → **creo** un nodo y lo apilo.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

→ - **creo** un nodo R,

- **desapilo** y lo agrego como hijo derecho de R

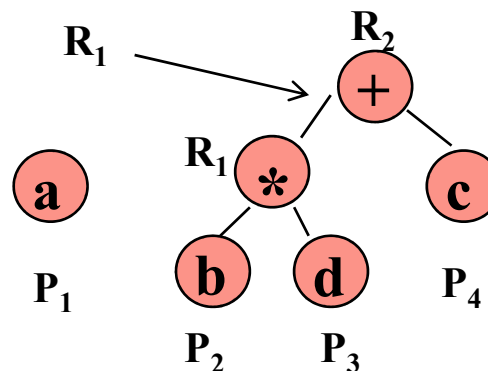
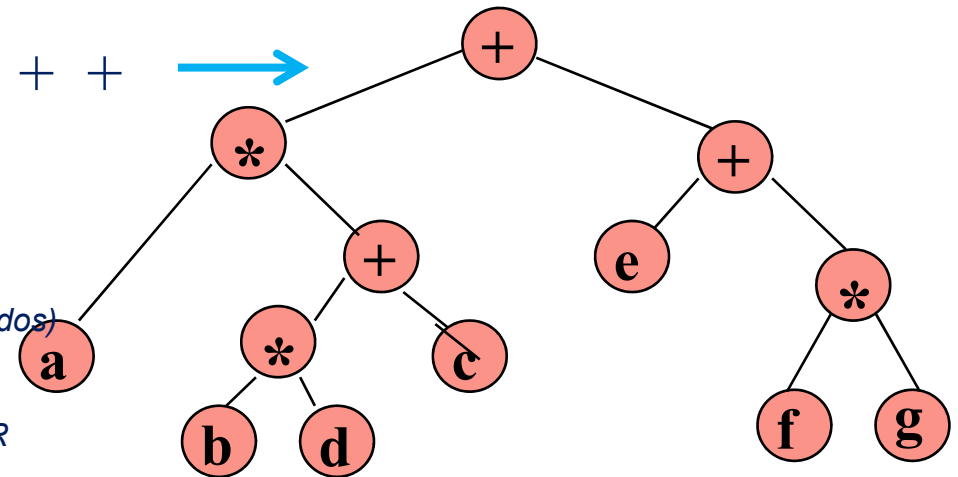
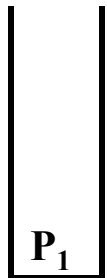
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R.

tomo otro carácter

fin

~~a~~ ~~b~~ ~~d~~ ~~*~~ ~~c~~ + * e f g * + +



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ 

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo apilo.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

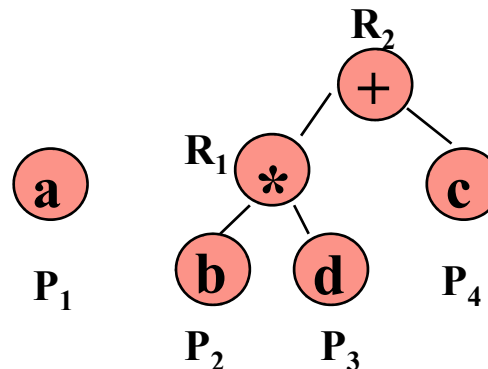
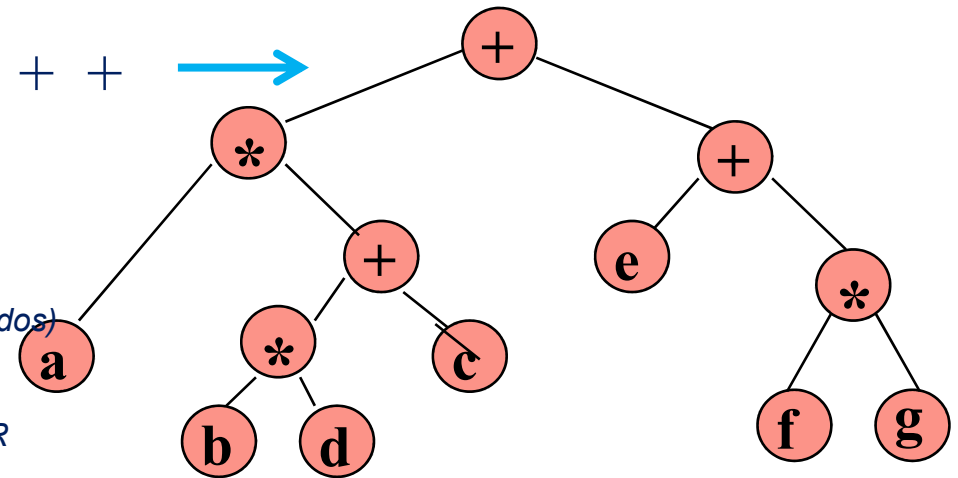
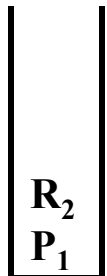
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d\ *\ c$~~ $+ \ *\ e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$ \rightarrow

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** \rightarrow **creo** un nodo y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

\rightarrow - **creo** un nodo R ,

- **desapilo** y lo agrego como hijo derecho de R

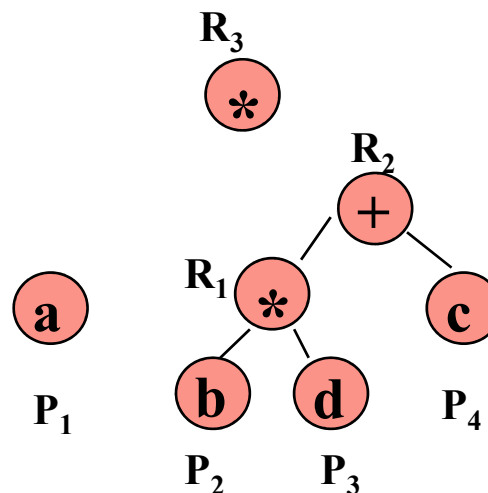
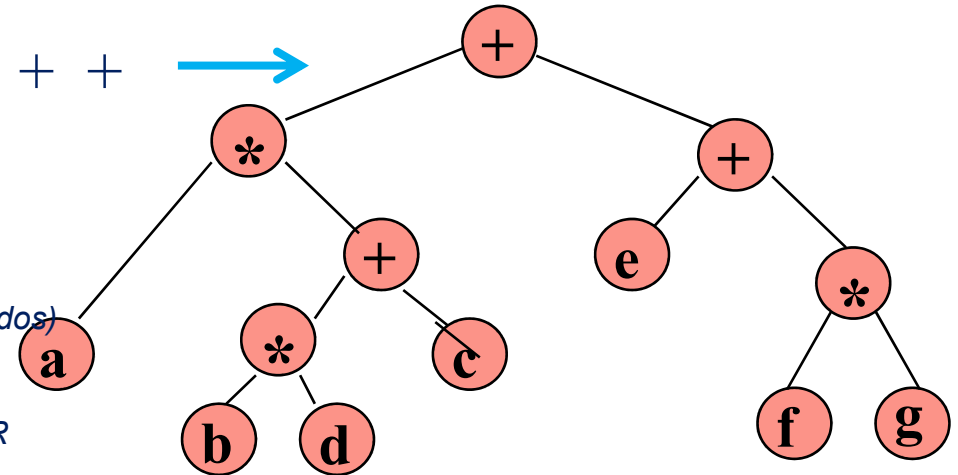
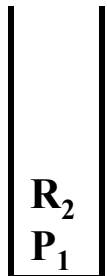
- **desapilo** y lo agrego como hijo izquierdo de R

- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d\ *\ c\ +\ *$~~ $e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a \ b \ d \ * \ c \ + \ * \ e \ f \ g \ * \ + \ +$ 

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

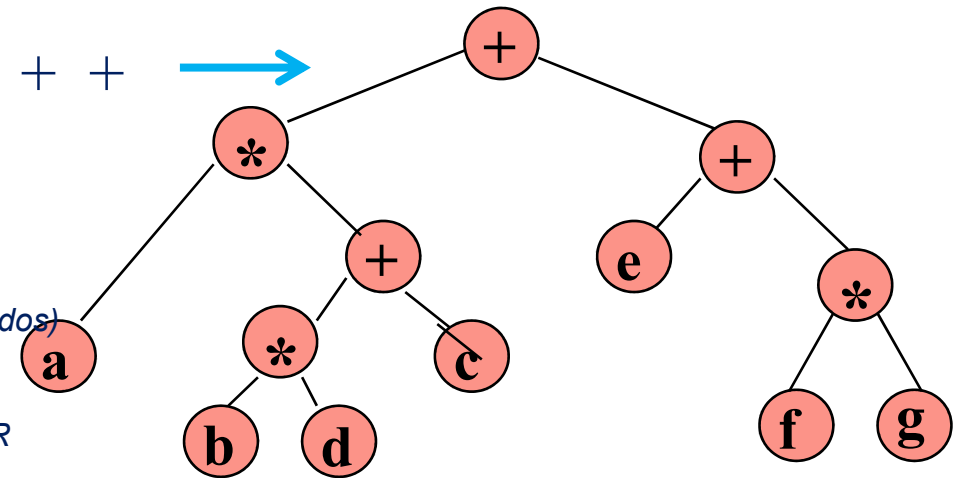
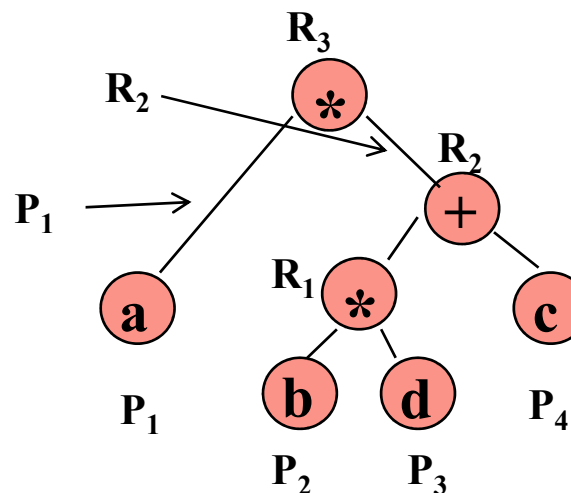
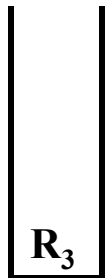
si es un **operando** → **creo un nodo** y lo **apilo**.

si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

- - **creo** un nodo R ,
- **desapilo** y lo agrego como hijo derecho de R
- **desapilo** y lo agrego como hijo izquierdo de R
- **apilo** R .

tomo otro carácter

fin

$$\cancel{a} \cancel{b} \cancel{d}^* \cancel{e}^+^* \quad e \quad f \quad g^* \quad + \quad +$$


1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a\ b\ d\ *\ c\ +\ *\ e\ f\ g\ *\ +\ +$

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

si es un **operando** → **creo** un nodo y lo **apilo**.

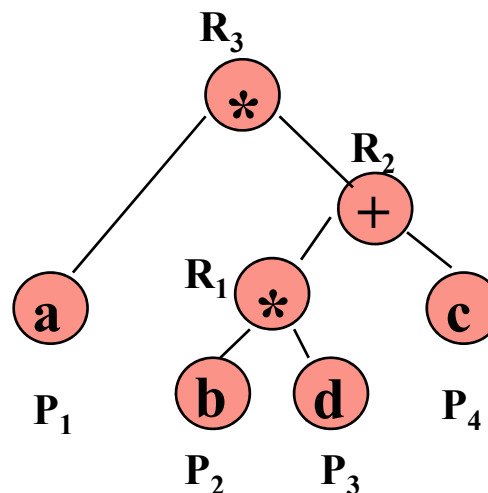
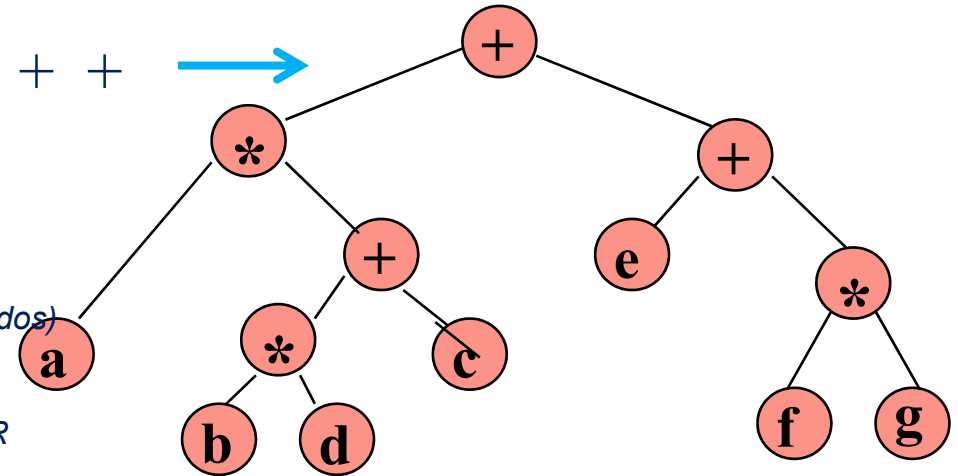
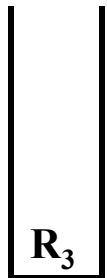
si es un **operador** (lo tomo como la raíz de los dos últimos nodos creados)

- - **creo** un nodo R ,
- **desapilo** y lo agrego como hijo derecho de R
- **desapilo** y lo agrego como hijo izquierdo de R
- **apilo** R .

tomo otro carácter

fin

~~$a\ b\ d\ *\ c\ +\ *$~~ $e\ f\ g\ *\ +\ +$



1) Construcción de un árbol de expresión a partir de una expresión postfija

Expresión postfija: $a \ b \ d \ * \ c \ + \ * \ e \ f \ g \ * \ + \ +$ 

Algoritmo:

tomo un carácter de la expresión

mientras (existe carácter) hacer

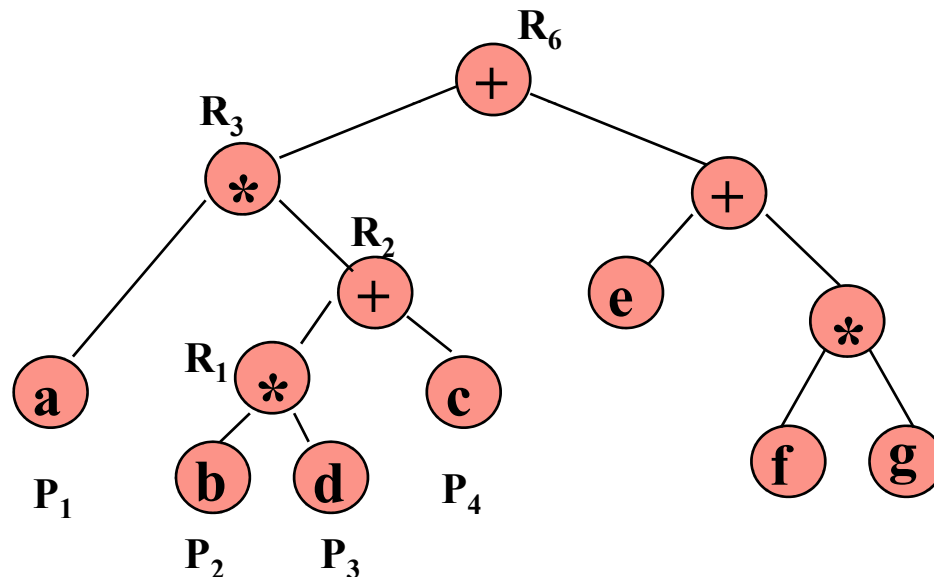
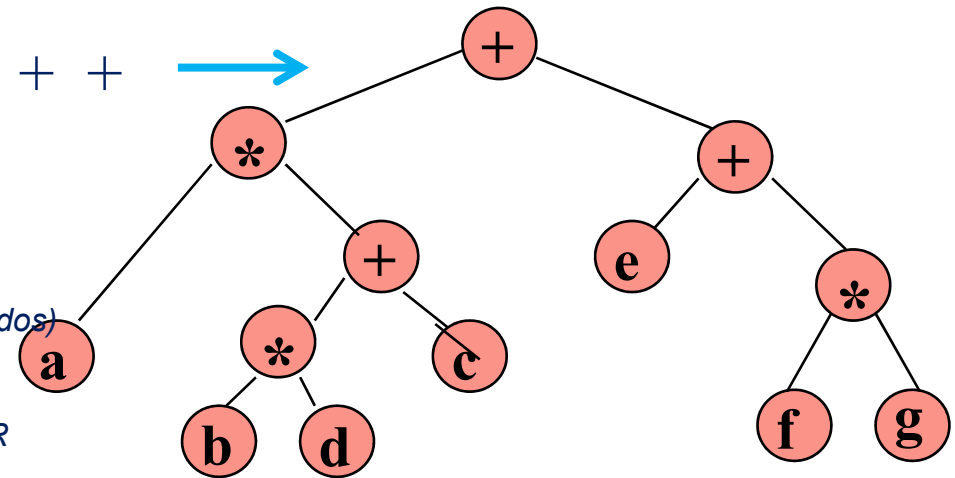
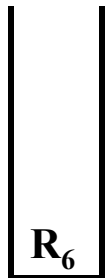
si es un **operando** → **creo un nodo** y lo **apilo**.

si es un operador (lo tomo como la raíz de los dos últimos nodos creados)

- - **creo** un nodo R ,
- **desapilo** y lo agrego como hijo derecho de R
- **desapilo** y lo agrego como hijo izquierdo de R
- **apilo** R .

tomo otro carácter

fin

$$\cancel{a} \cancel{b} \cancel{d}^* \cancel{c}^* \quad e \quad f \quad g^* + +$$


2) Construcción de un árbol de expresión a partir de una expresión prefija

Algoritmo:

ArbolExpresión (A: ArbolBin, exp: string)

si *exp nulo* \rightarrow *nada.*

si *es un operador* \rightarrow - *creo un nodo raíz R*

- *ArbolExpresión (subArbIzq de R, exp
(sin 1º carácter))*

- *ArbolExpresión (subArbDer de R, exp
(sin 1º carácter))*

si *es un operando* \rightarrow *creo un nodo (hoja)*

3) Construcción de un árbol de expresión a partir de una expresión infija

Expresión infija

(i)



Expresión postfija

Se usa una pila y se tiene en cuenta la precedencia de los operadores

2+5*3+1



253*+1+

3) Construcción de un árbol de expresión a partir de una expresión infija

Expresión infija

(i)



Se usa una pila y se tiene en cuenta la precedencia de los operadores

Expresión postfija

(ii)



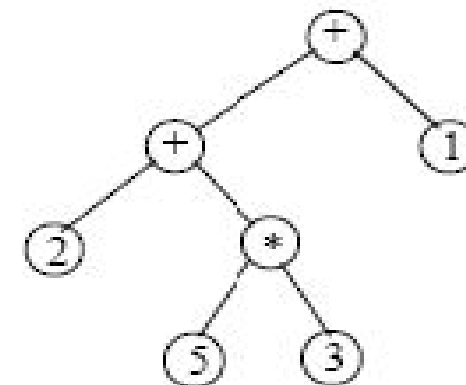
Se usa la estrategia 1)

Árbol de Expresión

2+5*3+1



253*+1+



-Convertir una expresión infija en árbol de expresión: se debe convertir la expresión infija en postfija (i) y a partir de ésta, construir el árbol de expresión (ii).

(i) Estrategia del Algoritmo para convertir exp. infija en postfija :

- a) si es un operando → se coloca en la salida.**
- b) si es un operador → se maneja una pila según la prioridad del operador en relación al tope de la pila**

operador con > prioridad que el tope → se apila
operador con <= prioridad que el tope → se desapila elemento colocándolo en la salida.

Se vuelve a comparar el operador con el tope de la pila

- c) si es un “(“ , “)” → “(“ se apila
“)” se desapila todo hasta el “(“, incluido éste**

- d) cuando se llega al final de la expresión, se desapilan todos los elementos llevándolos a la salida, hasta que la pila quede vacía.**

Operadores ordenados de mayor a menor según su prioridad:

\wedge (potencia)
 $*, /$ (multiplicación y división)
 $+, -$ (suma y resta)

Los “ (“ siempre se apilan como si tuvieran la mayor prioridad y se desapilan sólo cuando aparece un “) ” .

Ejercitación

Árbol binario de expresión

Ejercicio 1.

- ✓ Dada la siguiente expresión postfija : $IJK++AB* C -*$, dibuje su correspondiente árbol binario de expresión
- ✓ Convierta la expresión $((a+b)+c*(d+e)+f)*(g+h)$ en expresión prefija

Ejercicio 2.

- ✓ Dada la siguiente expresión prefija : $*+I+JK-C*AB$, dibuje su correspondiente árbol binario de expresión
- ✓ Convierta la expresión $((a+b)+c*(d+e)+f)*(g+h)$ en expresión postfija