

## Estructuras de Datos

### Clase 12 – Árboles binarios



Dr. Sergio A. Gómez  
<http://cs.uns.edu.ar/~sag>



Departamento de Ciencias e Ingeniería de la Computación  
 Universidad Nacional del Sur  
 Bahía Blanca, Argentina

## Árboles binarios

- Un árbol binario es un árbol ordenado que cumple:
  - 1) Cada nodo tiene a lo sumo dos hijos
  - 2) Cada nodo hijo es o bien hijo izquierdo o hijo derecho
  - 3) El hijo izquierdo precede al hijo derecho en el orden de los hijos de un nodo
- El subárbol que tiene como raíz al hijo izquierdo se llama subárbol izquierdo.
- El subárbol que tiene como raíz al hijo derecho se llama subárbol derecho.
- En un árbol binario propio, cada nodo tiene 0 o dos hijos (GT también le dice full BT).
- Si un árbol binario no es propio, entonces es impropio.

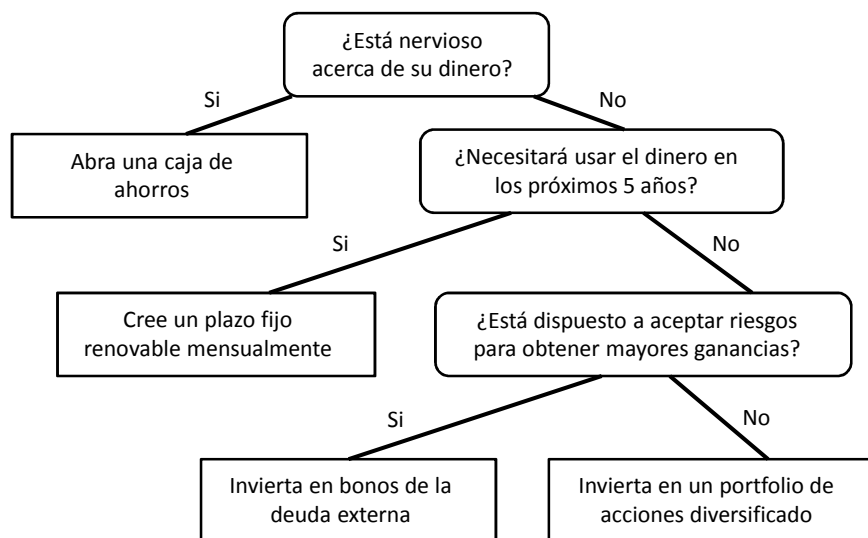
## Aplicaciones: Árboles de decisión

- Se desea representar los resultados asociados a un conjunto de preguntas con respuesta “Si” o “No”
- Cada nodo interno se asocia con una pregunta.
- Se comienza en la raíz y con cada pregunta se va a la izquierda o a la derecha dependiendo de si la respuesta a la pregunta es “si” o “no”.
- Cuando se llega a una hoja, se tiene un resultado al cual se llega partir de las respuestas dadas en los ancestros de la hoja.

Estructuras de datos - Dr. Sergio A. Gómez

3

## Aplicaciones: Árboles de decisión



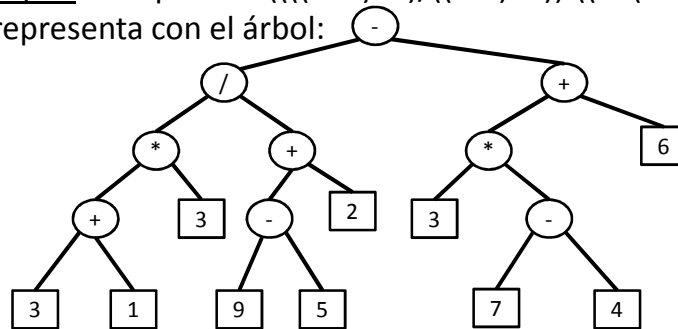
Estructuras de datos - Dr. Sergio A. Gómez

4

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Aplicaciones: Expresiones aritméticas

- Una expresión aritmética puede representarse con un árbol binario.
- Las hojas almacenan constantes o variables
- Los nodos internos almacenan los operadores +, -, \* y /.
- Ejemplo: La expresión  $((((3+1)*3)/((9-5)+2))-((3*(7-4))+6))$  se representa con el árbol:



Resolución de Problemas y Algoritmos - Dr. Sergio A. Gómez

5

## Definición recursiva de árbol binario

Un árbol binario  $T$  es o bien vacío, o consiste de:

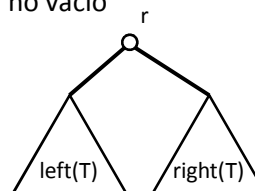
- Un nodo  $r$ , llamado la raíz de  $T$ , que contiene un rótulo (o elemento)
- Un árbol binario, llamado el subárbol izquierdo de  $T$
- Un árbol binario, llamado el subárbol derecho de  $T$

Caso base:

$T = \text{Árbol vacío}$

Caso recursivo:

$T = \text{Árbol no vacío}$



Estructuras de datos - Dr. Sergio A. Gómez

6

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Definición recursiva de árbol binario (no vacío)

Un árbol binario  $T$  es :

Hoja(n): Un nodo  $r$ , llamado la raíz de  $T$ , que contiene un rótulo (o elemento  $n$ )

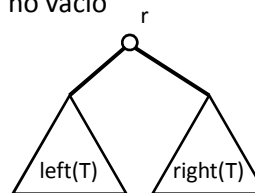
Nodo(  $n$ ,  $TI$ ,  $TD$  ):  $n$  es el rótulo y  $TI$  y  $TD$  son árboles binarios llamados hijo izquierdo y derecho, resp.

Caso base:



Caso recursivo:

$T = \text{Árbol no vacío}$



Estructuras de datos - Dr. Sergio A. Gómez

7

## Obtención del árbol binario de expresión aritmética (parsing)

**Ejemplo:**  $\text{exp} = 5 + 6 - 2 + 1 + 10 - 7$  (expresión solo formada por  $+$  y  $-$ )

**Algoritmo** Parse( exp )

**Si** exp no contiene un operador **entonces**

**retornar** un árbol binario hoja conteniendo rótulo de exp

**Sino**

$\text{op} \leftarrow \text{último\_operador}(\text{exp})$

$T_1 \leftarrow \text{Parse}(\text{izquierda}(\text{exp}, \text{op}))$

$T_2 \leftarrow \text{Parse}(\text{derecha}(\text{exp}, \text{op}))$

**retornar** un árbol binario con op como rótulo de la raíz  
y  $T_1$  y  $T_2$  como hijos izquierdo y derecho resp.

**Nota:**

- “Último\_operador” recorre exp de derecha a izquierda y puede usar un contador de paréntesis para determinar cuándo encuentra el operador principal o una pila si permitimos  $\{$ ,  $\}$ ,  $[$  y  $()$ .
- Ver versión iterativa en sección 7.3.6 de GT.

Estructuras de datos - Dr. Sergio A. Gómez

8

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
“Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Evaluación de un árbol de expresión aritmética

Cada nodo en un árbol de expresión tiene un valor asociado:

- Si el nodo es externo, su valor es el de la variable o constante
- Si el nodo es interno, su valor está definido por la aplicación de la operación a los valores de sus hijos.

**Algoritmo** Evaluar( arbol\_exp )

Si arbol\_exp es una hoja **entonces**

**retornar** el valor del rótulo de arbol\_exp

**Sino**

    op  $\leftarrow$  rótulo de raíz de arbol\_exp

$v_1 \leftarrow$  Evaluar( hijo izquierdo de arbol\_exp )

$v_2 \leftarrow$  Evaluar( hijo derecho de arbol\_exp )

**retornar** aplicar( op,  $v_1$ ,  $v_2$  ) // Aplicar realiza la operación  $v_1$  op  $v_2$

## ADT Arbol Binario [GT]

El Árbol Binario (de acuerdo a GT) es una especialización (subinterfaz) de Tree que además soporta los métodos de acceso adicionales:

- left(v): Retorna el hijo izquierdo de v, ocurre error si v no tiene hijo izquierdo
- right(v): Retorna el hijo derecho de v, ocurre error si v no tiene hijo derecho
- hasLeft(v): Testea si v tiene hijo izquierdo
- hasRight(v): Testea si v tiene hijo derecho

## Interfaz BinaryTree [GT]

```
public interface BinaryTree<E> extends Tree<E>
{
    public Position<E> left( Position<E> v )
        throws InvalidPositionException, BoundaryViolationException;
    public Position<E> right( Position<E> v )
        throws InvalidPositionException, BoundaryViolationException;
    public boolean hasLeft( Position<E> v )
        throws InvalidPositionException;
    public boolean hasRight( Position<E> v )
        throws InvalidPositionException;
}
```

Estructuras de datos - Dr. Sergio A. Gómez

11

## Estructura enlazada para el árbol binario

```
public interface BTPosition<E> extends Position<E> {
    // agrega getters y setters para element, parent, left y right.
}

public class BTNode<E> implements BTPosition<E> {
    private E element;
    private BTPosition<E> left, right, parent;

    public BTNode( E element, BTPosition<E> left, BTPosition<E> right,
        BTPosition<E> parent ) {
        this.element = element; this.left = left;
        this.right = right; this.parent = parent;
    }

    // setters y getters para element, left, right y parent.
}
```

Estructuras de datos - Dr. Sergio A. Gómez

12

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Implementación del árbol binario

```
public class ArbolBinarioEnlazado<E> implements BinaryTree<E> {
    protected BTPosition<E> raiz;
    protected int size;

    public ArbolBinarioEnlazado() { raiz = null; size =0; }
    public int size() { return size; }

    public boolean hasLeft( Position<E> v) throws InvalidPositionException {
        BTPosition<E> n = checkPosition(v); // Propaga excepción de pos. inválida
        return n.getLeft() != null;
    }
    public Position<E> left(Position<E> v)
        throws InvalidPositionException, BoundaryViolationException {
        BTPosition<E> n = checkPosition(v); // Propaga excepción pos. inválida
        if( n.getLeft() == null ) throw new BoundaryViolationException( "..." );
        return n.getLeft();
    }
    public boolean isInternal(Position<E> v) throws InvalidPositionException {
        return hasLeft(v) || hasRight(v); // Propaga excepción de pos. inválida
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez 13

## Implementación del árbol binario (cont).

```
public Iterable<Position<E>> children( Position<E> v )
    throws InvalidPositionException {
    PositionList<Position<E>> hijos = new MiLista<Position<E>>();
    if( hasLeft(v) ) hijos.addLast( left(v) );
    if( hasRight(v) ) hijos.addLast( right(v) );
    return hijos;
}
```

Estructuras de datos - Dr. Sergio A. Gómez 14

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:  
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

## Implementación del árbol binario (cont.)

- Métodos de modificación:
  - addRoot(e) (o createRoot(e)): Agrega un nodo raíz con rótulo e, error si ya hay raíz
  - insertLeft(v, e): Crea un nodo w hijo izquierdo de v con rótulo e, error si v ya tiene hijo izquierdo
  - insertRight(v,e): Crea un nodo w hijo derecho de v con rótulo e, error si v ya tiene hijo derecho
  - remove(v): Elimina el nodo v (si v tiene un hijo, reemplaza a v por su hijo, si v tiene dos hijos entonces error).
  - attach( v, T<sub>1</sub>, T<sub>2</sub> ): Setea T<sub>1</sub> como hijo izq de v y T<sub>2</sub> como hijo derecho de v (error si v no es hoja).
- Crea un nodo w hijo de v con rótulo e, error si v ya tiene hijo izquierdo
- Ver fragmento de código 7.18 y 7.19 en GT.

## Tiempo de ejecución

Operación	Tiempo
size, isEmpty	O(1)
iterator, positions	O(n)
Replace	O(1)
Root, parent, children, left, right, sibling	O(1)
hasLeft, hasRight, isInternal, isExternal, isRoot	O(1)
insertLeft, insertRight, attach, remove	O(1)

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.



## Listados en árboles binarios

**Algoritmo** preorden( T, v )

Visitar v

si T.hasLeft( v ) entonces  
preorden( T, T.left(v) )  
si T.hasRight( v ) entonces  
preorden( T, T.right(v) )

**Algoritmo** posorden( T, v )

si T.hasLeft( v ) entonces  
posorden( T, T.left(v) )  
si T.hasRight( v ) entonces  
posorden( T, T.right(v) )  
Visitar v

**Algoritmo** Inorden( T, v )

si T.hasLeft( v ) entonces  
Inorden( T, T.left(v) )  
Visitar v  
si T.hasRight( v ) entonces  
Inorden( T, T.right(v) )

Estructuras de datos - Dr. Sergio A. Gómez

**Evaluación de expresión con recorrido posorden:**

**Algoritmo** Eval( T, v ) { Comienza en la raíz }

Si (T.isExternal(v) ) retornar v.element()

sino

$R_1 \leftarrow \text{Eval}( T, T.\text{left}(v) )$

$R_2 \leftarrow \text{Eval}( T, T.\text{right}(v) )$

op  $\leftarrow$  v.element()

retornar aplicar( op,  $R_1$ ,  $R_2$  )

**Nota:** El listado inorden de un árbol de expresión aritmética reproduce la expresión original ¡Programar como ejercicio!

17

## Representación del árbol binario con un arreglo

- La componente 0 no se usa
- La raíz se almacena en la componente 1
- El hijo izquierdo de la componente i se almacena en la componente  $2*i$
- El hijo derecho de la componente i se almacena en la componente  $2*i + 1$
- El padre de i está en la componente  $i / 2$
- **Nota:** Puede haber componentes vacías si el árbol no está lleno (un árbol está lleno si tiene todos los nodos; un árbol binario lleno de altura h tiene  $2^{h+1}-1$  nodos).

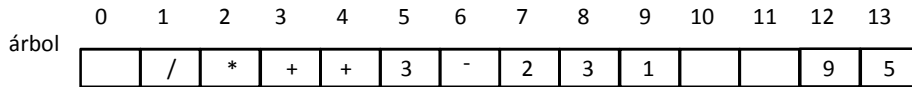
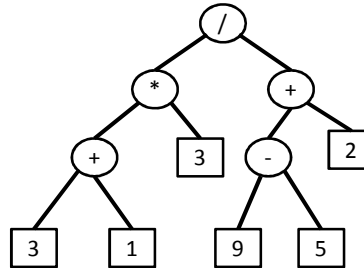
Estructuras de datos - Dr. Sergio A. Gómez

18

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

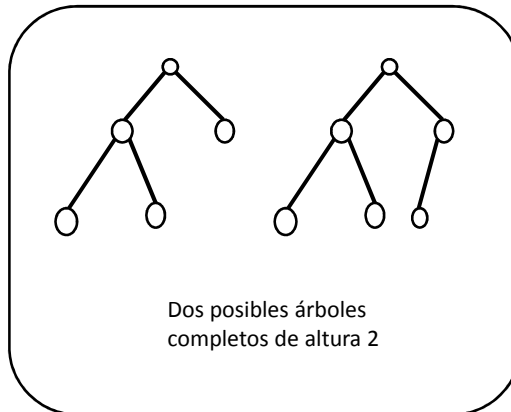
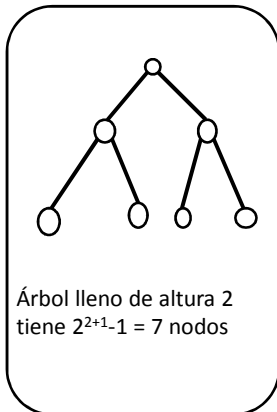
## Representación con arreglos del árbol binario

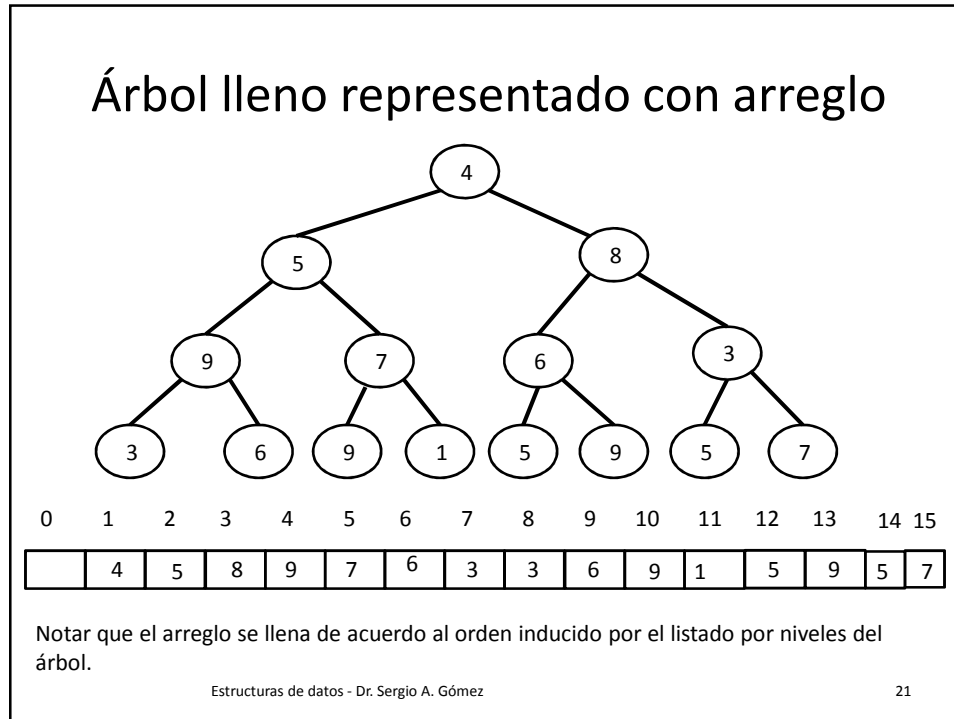
Hijo\_izquierdo(i) = 2i  
 Hijo\_derecho(i) = 2i+1  
 Padre(i) = i div 2



**Ojo:** La dimensión del vector crece exponencialmente en base 2 en la altura del árbol. Entonces está implementación sólo es viable si el árbol está o bien lleno ó completo (el árbol está completo cuando está lleno hasta el penúltimo nivel y el último nivel se completa de izquierda a derecha, lo usaremos la clase que viene).

## Árbol lleno versus árbol completo





## Bibliografía

- Capítulo 7 de Goodrich & Tamassia