



# Análisis y Diseño de Algoritmos.

**Sesión 6.** 30 de septiembre de 2015.

Maestría en Sistemas Computacionales.

Por: Hugo Iván Piza Dávila.

# ¿Qué veremos hoy?

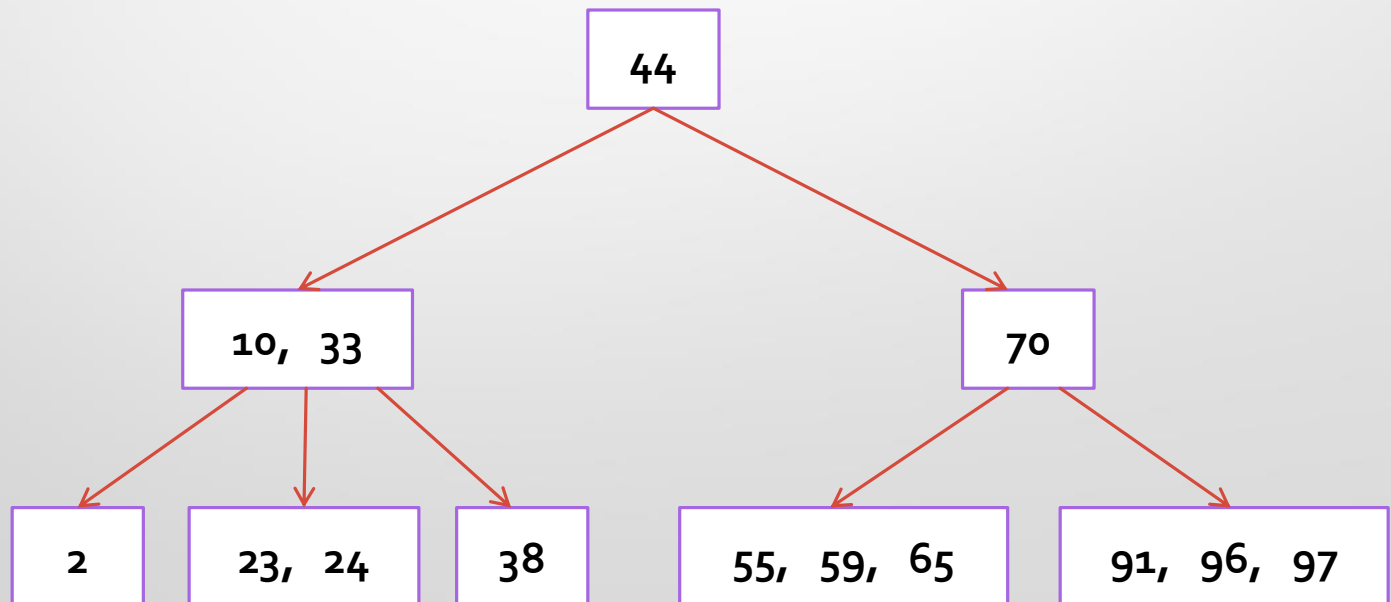
- Búsqueda con Árboles 2-3-4.
  - Componentes.
  - Lógica de creación.
  - Implementación de algoritmos de creación y búsqueda.

# Arboles balanceados

- Para eliminar los peores casos que se presentan en la búsqueda por *árbol binario*, el árbol tiene que mantenerse balanceado
- Arboles balanceados conocidos:
  - Arboles 2-3-4
  - Arboles *rojo-negro*
- Ofrecen muy buen desempeño ( $\approx \lg N$ ) pero no son nada fácil de programar.
  - Créase el árbol una vez, búsquese N veces

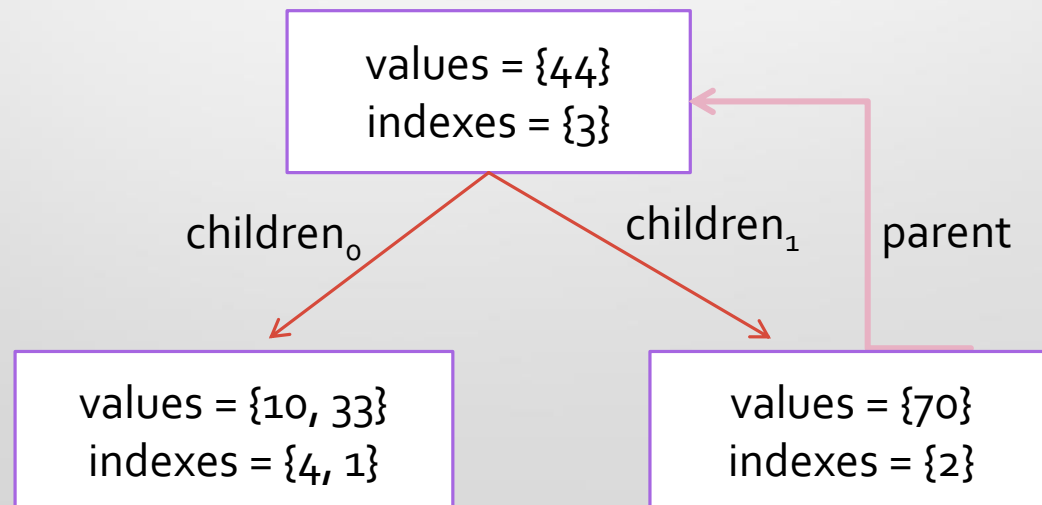
# Arboles 2-3-4

- Están compuestos por Nodos-2, Nodos-3, Nodos-4
- Tipo de nodo (2, 3, 4) = número de valores + 1 = número de hijos (sólo para nodos internos).



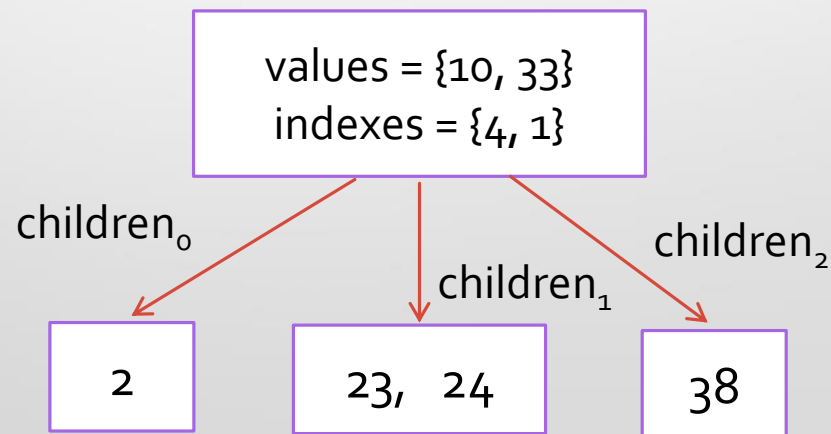
# Nodos-2

- Almacenan un valor y su índice en la lista original.
- Si el nodo no es una hoja, tiene dos hijos.
  - Todos los valores de  $children_0$  son menores que  $values_0$
  - Todos los valores de  $children_1$  son mayores que  $values_0$



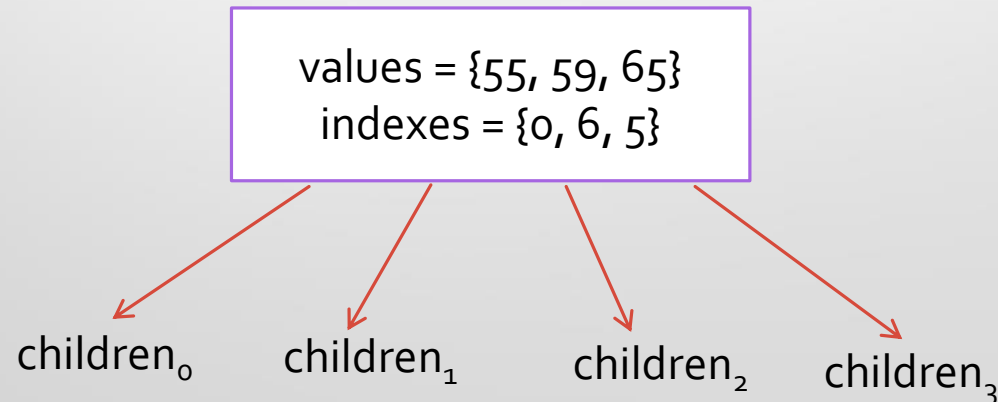
# Nodos-3

- Almacenan dos valores (y sus índices).
- Si el nodo no es una hoja, tiene tres hijos.
  - Todos los valores de  $children_0$  son menores que  $values_0$
  - Todos los valores de  $children_1$  están en el rango  $[values_0..values_1)$
  - Todos los valores de  $children_2$  son mayores que  $values_1$



# Nodos-4

- Almacenan tres valores (y sus índices).
- Si el nodo no es una hoja, tiene cuatro hijos.
  - Todos los valores de  $children_0$  son menores que  $values_0$
  - Todos los valores de  $children_1$  están en el rango  $[values_0..values_1)$
  - Todos los valores de  $children_2$  están en el rango  $[values_1..values_2)$
  - Todos los valores de  $children_3$  son mayores que  $values_2$



# Ejemplo de creación de Árbol 2-3-4

- Lista = {3, 1, 5, 6, 2, 0, 7, 4}

1. [3]

2. [1, 3]

3. [1, 3, 5]

4. [1, 3, 5] *Nodo-4  $\Rightarrow$  2 Nodos-2*

[3] *← Actual (nueva raíz)*

[1] [5]

[3]

[1] [5, 6]

- Lista = {3, 1, 5, 6, 2, 0, 7, 4}

5. [3]

[1, 2] [5, 6]

6. [3]

[0, 1, 2] [5, 6]

7. [3]

[0, 1, 2] [5, 6, 7]



# Ejemplo de creación de Árbol 2-3-4

- Lista = {3, 1, 5, 6, 2, 0, 7, 4}

8. [3]

[0, 1, 2] [5, 6, 7] *Nodo-4  $\Rightarrow$  2 Nodos-2*

[3, 6] *← Actual (subir nivel)*

[0, 1, 2] [5] [7]

[3, 6]

[0, 1, 2] [**4**, 5] [7]

# Lógica de inserción en Árbol 2-3-4

- Siempre comenzamos desde la raíz.
- El camino a elegir está en función del valor a insertar.
- Siempre colocamos el elemento en un nodo hoja.
- Si en el camino a la hoja encontramos un nodo-4:
  - Lo remplazamos por dos nodo-2 con los valores izquierdo (o) y derecho (2) del nodo-4.
  - El valor medio se colocará en el nodo padre.
  - Pero ¿y si el nodo actual no tiene padre?
  - Se crea un nuevo nodo raíz, con el valor intermedio.
- Nótese que el árbol crece hacia arriba.

# Ejercicio

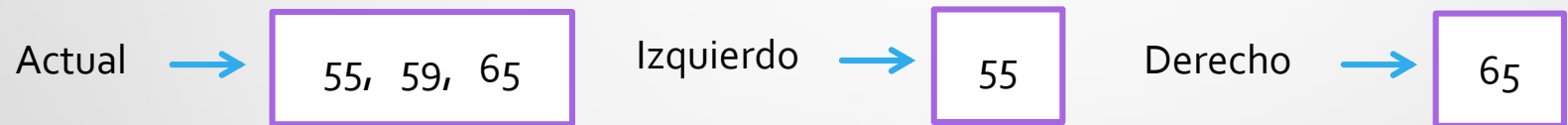
- Crea el árbol 2-3-4 que corresponde a una lista ordenada con los primeros 9 números naturales.
- ¿Cuántos brincos en el árbol tengo que dar para encontrar el elemento más lejano a la raíz?

# Algoritmo de creación del Árbol 2-3-4.

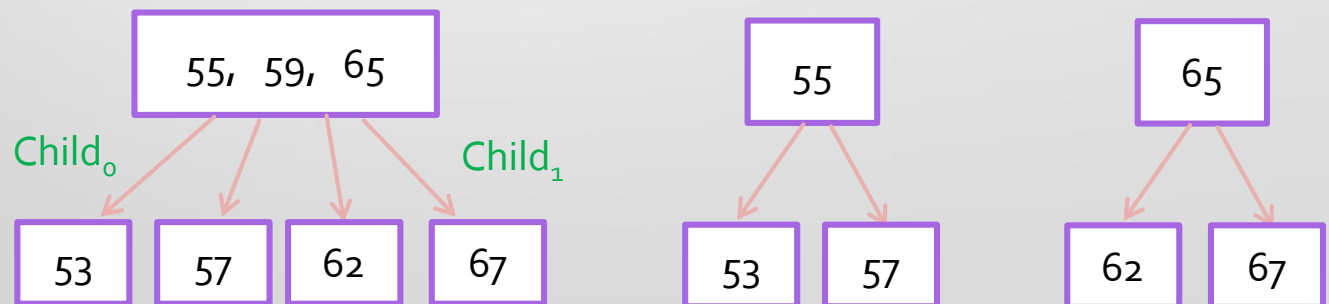
1. Crear un nodo-2 raíz con los datos del 1<sup>er</sup> elemento de la lista.
2. Por cada elemento siguiente de la lista:
  - a) Sea *actual* un nodo que apunta a la raíz.
  - b) Mientras no se haya procesado un nodo hoja:
    - i. Si *actual* es un nodo-4, desmenuzarlo (diapositiva siguiente)
    - ii. Si no, si *actual* es una hoja, colocar los datos (valor, índice) en la misma posición, tal que se mantenga la lista de valores ordenada.
    - iii. Si no, *actual* apuntará ahora a uno de sus hijos, elegido de acuerdo al valor del elemento y al tipo de nodo *actual*: nodo-2 o nodo-3.

# Desmenuzar un nodo-4

- a) Crear nodos *izquierdo* y *derecho* que almacenen los valores (e índices) de los extremos del nodo actual.

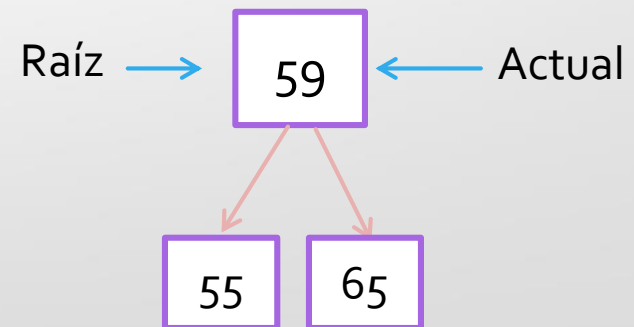


- b) Si el nodo actual no es una hoja, agregar a los nodos *izquierdo* y *derecho* los hijos correspondientes de *actual*. Nota: al agregar un *hijo*, indicar al hijo quién es su nodo *padre*.



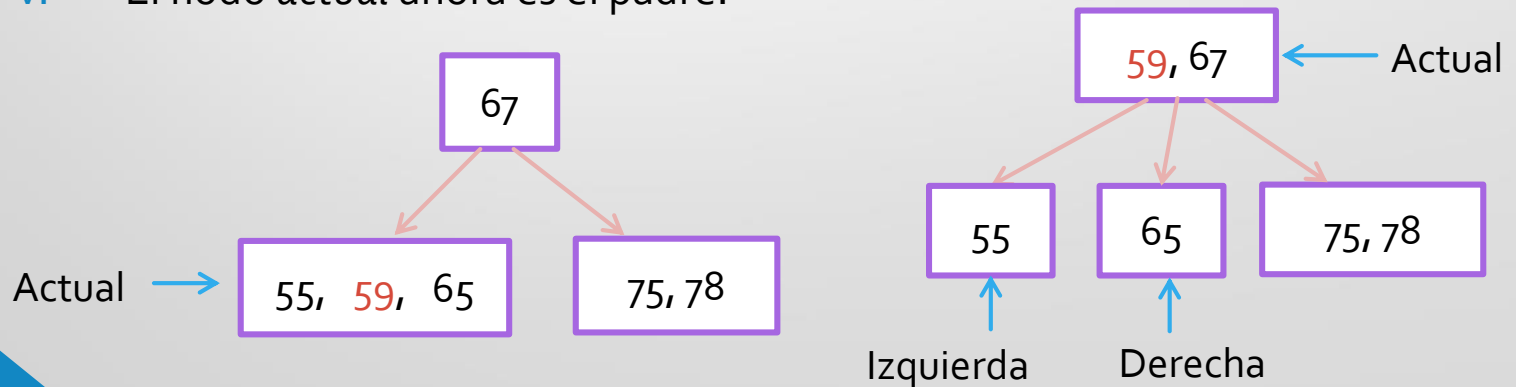
# Desmenuzar un nodo-4

- c) Si *actual* es la raíz, el árbol crece hacia arriba:
- i. Crear una nueva raíz con el valor medio de *actual*.
  - ii. Los hijos de la raíz serán los nodos *izquierdo* y *derecho*.
  - iii. El nodo *actual* será la raíz.



# Desmenuzar un nodo-4

- d) En caso contrario (*actual* no es la raíz):
- Obtener el nodo padre.
  - Insertar en el padre el valor intermedio de *actual*.
  - Insertar en el padre el nodo *izquierdo*, en la posición donde se insertó el valor, reemplazando al hijo que estaba ahí [55, 59, 65].
  - Insertar en el padre el nodo *derecho* en la posición siguiente.
  - El nodo *actual* ahora es el padre.



# Estructura de un Nodo-234

## Node234

- values : Linked list of Integer
- indexes : Linked list of Integer
- children : Linked list of Node234
- parent : Node234

+ Node234(int value, int index-in-array)

+ getValue(int i) : int i = {2, 3, 4}

+ getIndex(int i) : int

+ getChild(int i) : int

+ getType() : int

Devuelve 2, 3 ó 4.

+ isLeaf() : boolean

¿Tiene hijos?

+ insert(value, index) : int

Devuelve dónde fue insertado.

+ getParent() : Node234

+ addChild(child)

Añade al final, actualiza el padre)

+ addChildren(ch1, ch2, index)

Coloca ch1 en posición index  
sustituyendo al anterior; inserta ch2  
en index + 1. Actualiza los padres.