Estructura de Datos

Maria C. Torres

Maria C. Torres

Ing. Electrónica (UNAL)
M.E. Ing. Eléctrica (UPRM)
Ph.D. Ciencias e Ingeniería de la Computación y la Información (UPRM)
Profesora asociada
Dpto. Ciencias de la Computación y la Decisión

mctorresm@unal.edu.co

HORARIO DE ATENCIÓN: Martes 10:00 am a 12:00 m – Oficina 313 M8A

Contenido del Curso

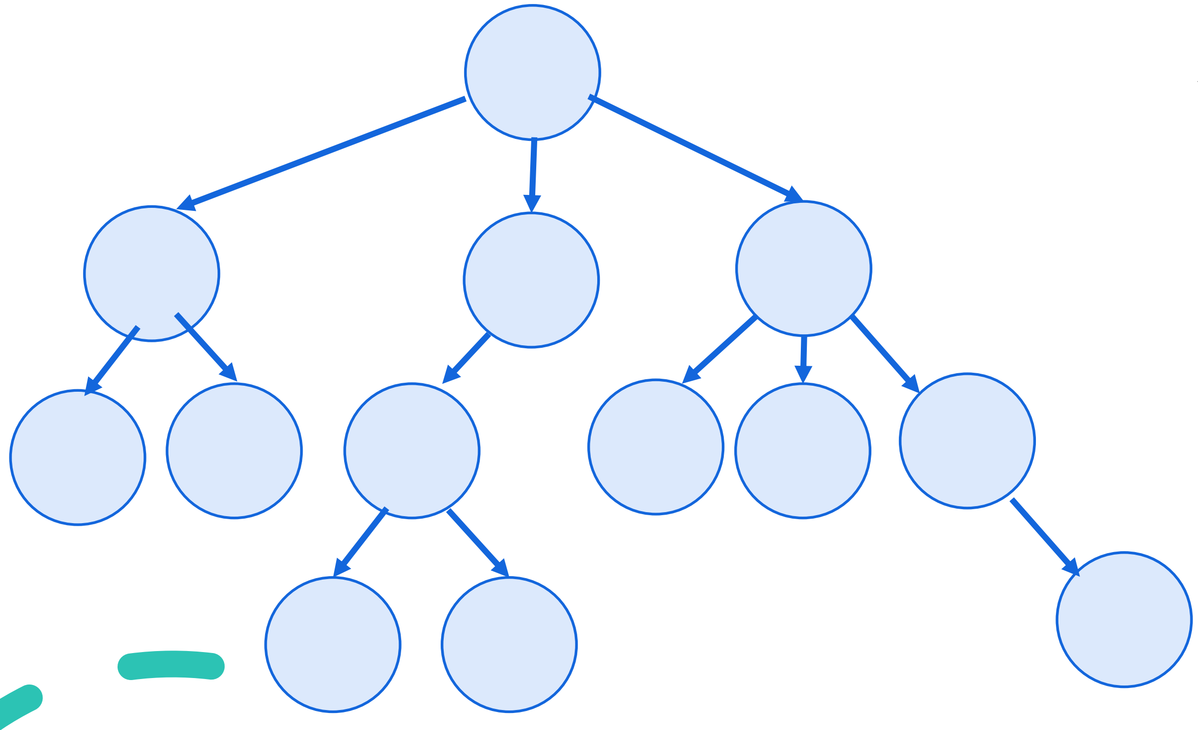
- Introducción: revisión fundamentos y POO
- Análisis de complejidad
- Arreglos
- Listas enlazadas
- Pilas y colas
- Heap
- **Arboles binarios**
- Tablas hash
- Grafos

Árbol Binario de Búsqueda

- ❑ Arboles
- ❑ Arboles binarios
- ❑ Arboles binarios de búsqueda
- ❑ Arboles balanceados

Arboles

- Estructura de datos no lineal y jerárquica.
- En los árboles en general no se maneja la relación “antes” y “después” como en las listas, sino “arriba” y “abajo”



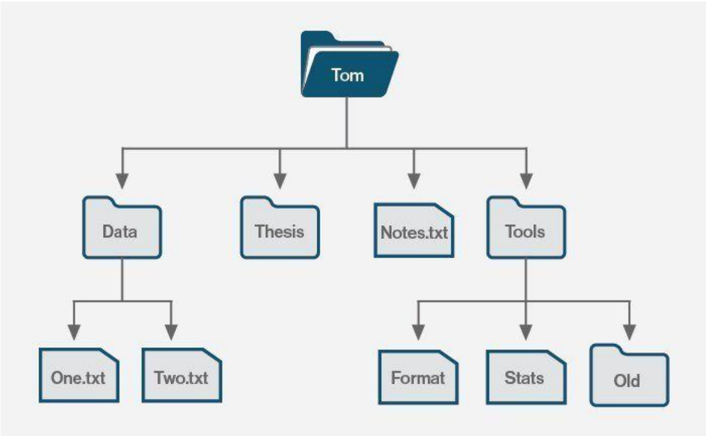
A diferencia de los **heap**, los árboles son estructuras enlazadas:

- Usualmente se implementan mediante la conexión de nodos (similar a las listas simples y dobles)

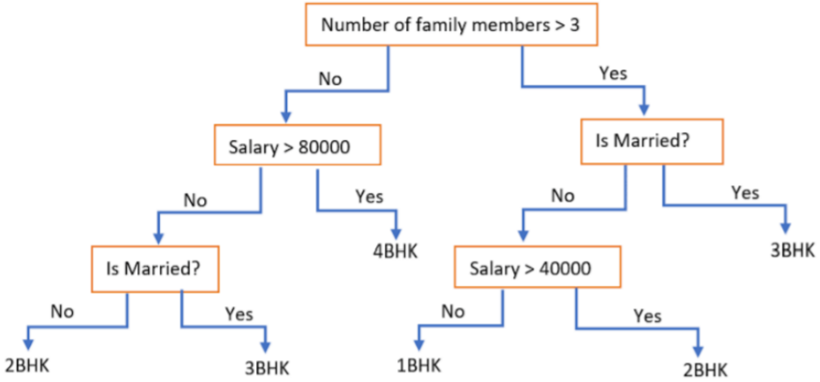
Arboles

Aplicaciones

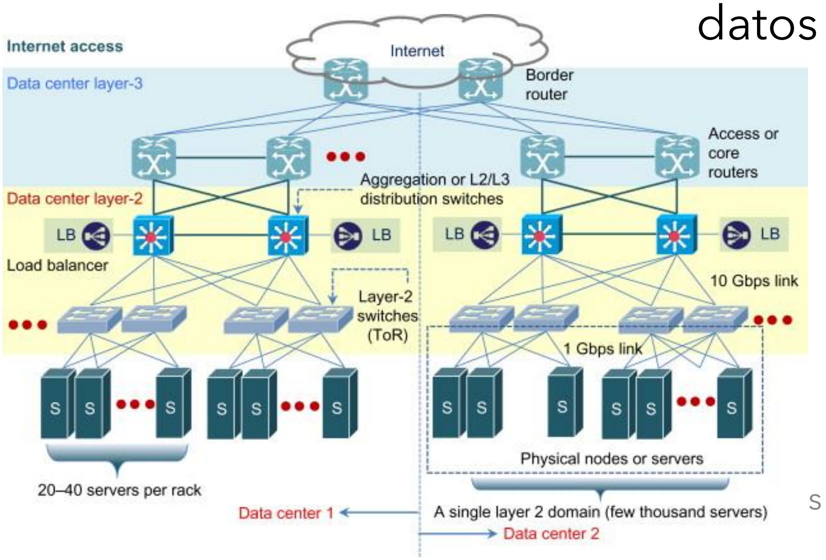
Directorio de datos



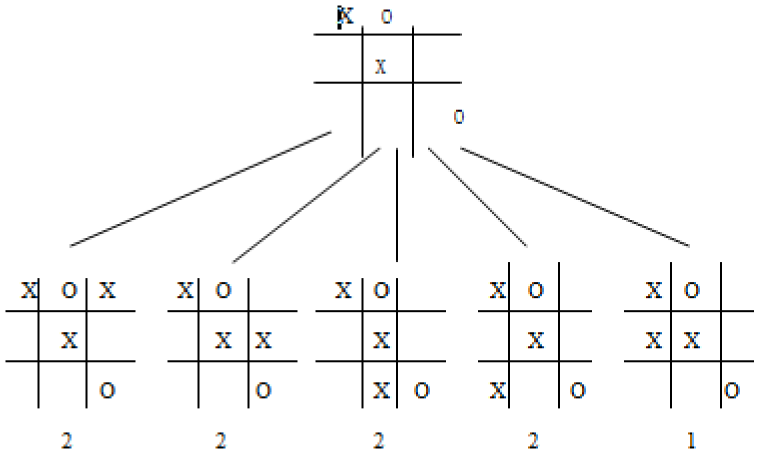
Arboles de decisión



Transmisión de datos



Juegos

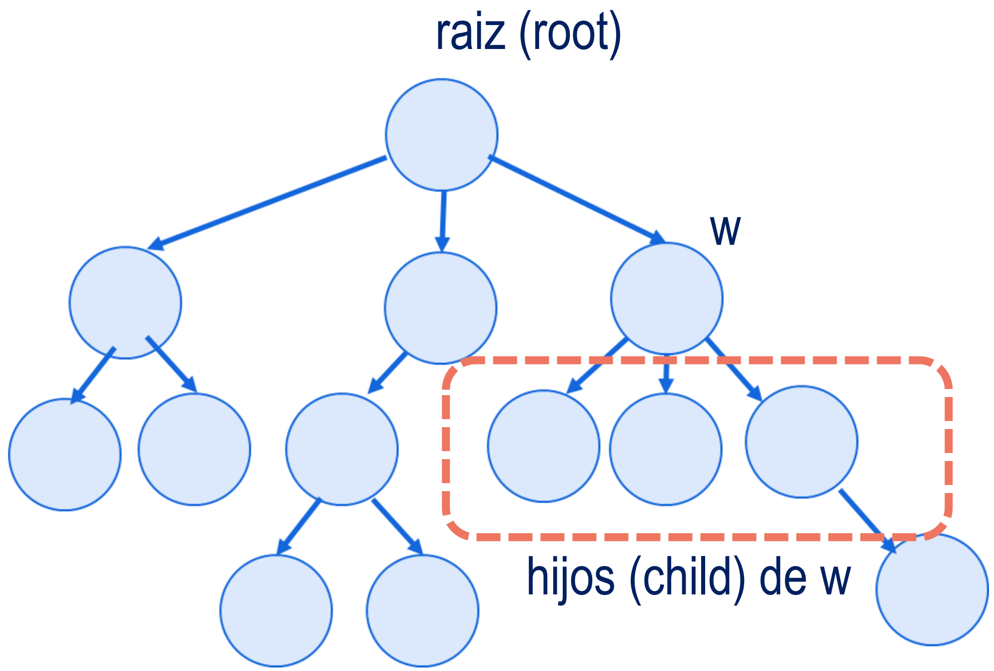


Arboles

Definición formal

Un árbol **T** es un conjunto de nodos que almacenan datos, estos nodos tienen una relación de padre-hijos, con las siguientes propiedades:

- Si **T** no está vacío, tiene un nodo especial denominado la ***raíz***. El nodo raíz no tiene padre.
- Cada nodo v de T diferente de la raíz tiene un único ***padre*** w; cada nodo con padre w es un ***hijo*** de w.

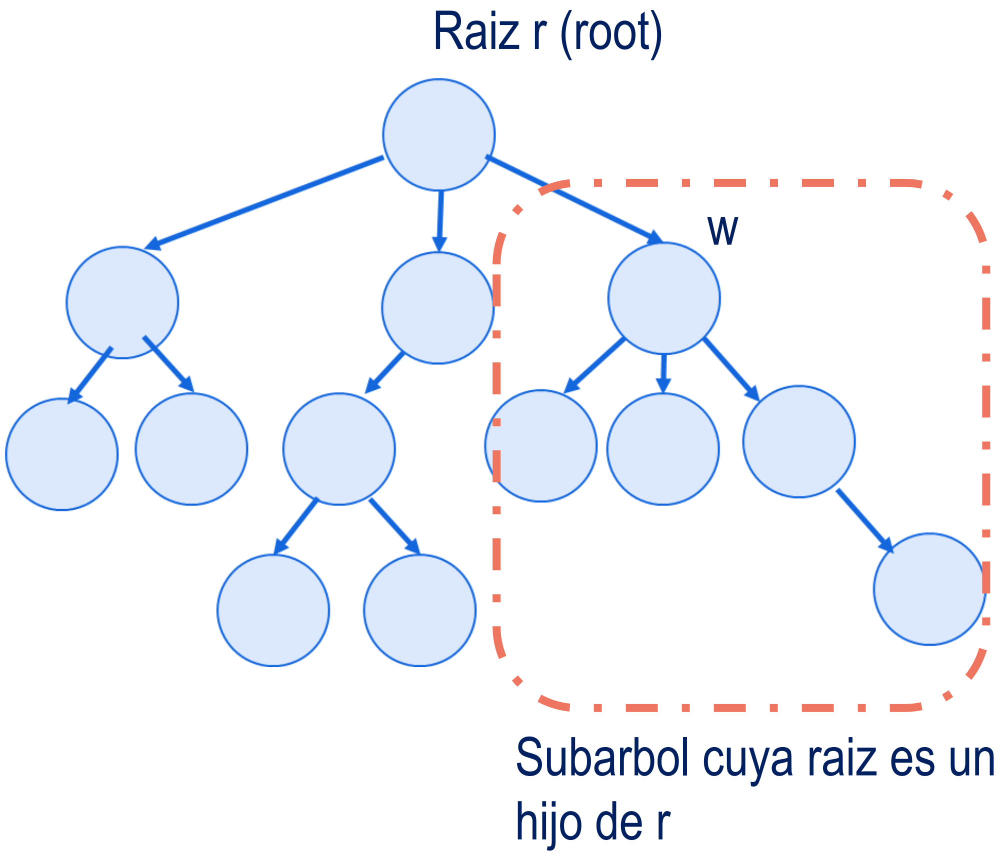


Arboles

Definición recursiva

Un árbol **T** es:

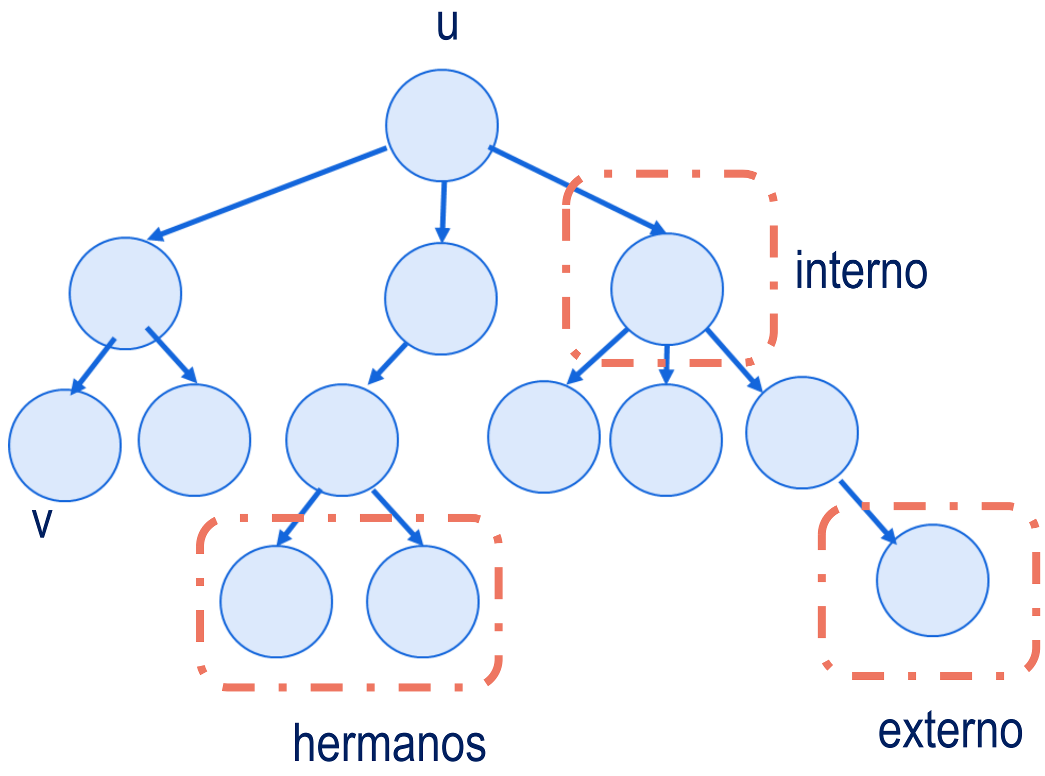
- ❑Un conjunto vacío o
- ❑Consiste de un nodo r, llamado la **raíz** de T, y un conjunto de subárboles con raíces iguales a los hijos de r



Arboles

CONCEPTOS BASICOS

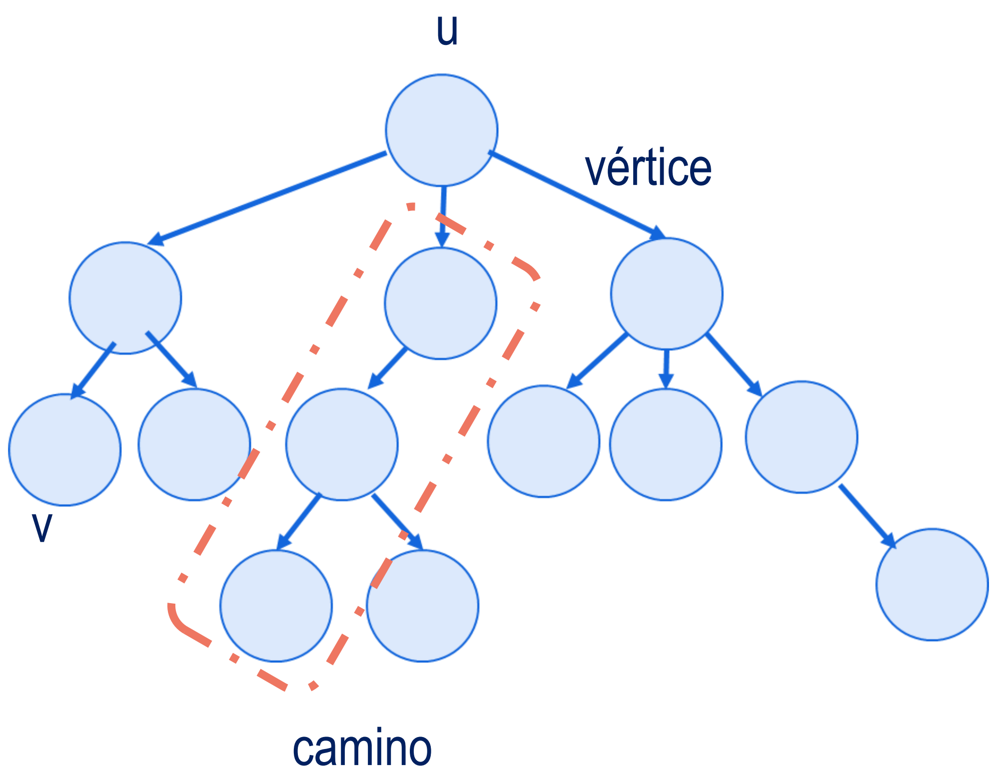
- ❑Dos o más nodos que son hijos del mismo padre son **hermanos**
- ❑Un nodo v es **externo** si v no tiene hijos, también se conocen como hojas
- ❑Un nodo v es **interno** si tiene uno o más hijos
- ❑Un nodo u es un **ancestro** de v, si u=v o si u es un ancestro del padre de v
- ❑Un nodo v es un **descendiente** de u, si u es un ancestro de v



Arboles

CONCEPTOS BASICOS

- Un **vértice** (edge) de un árbol T es un par de nodos (u,v) tal que, u es el padre de v, o viceversa.
- Un **camino** (path) es una secuencia de nodos tal que dos nodos consecutivos en la secuencia forman un vértice



Arboles

Profundidad

Considere v un nodo de un árbol T. La **profundidad** (depth) de v es el número de ancestros de v, excluyéndose a sí mismo.

La profundidad se puede definir de forma recursiva:

- ❑Si v es la raíz, la profundidad de v es 0
- ❑En otro caso, la profundidad de v es 1 más la profundidad del padre de v

Seudocodigo

```
depth(T,v)
if T.isRoot(v) //caso base
    return 0
else //llamado recursivo
    return 1 + Depth(T,v.parent())
```

Arboles

Altura

La **altura** (height) de un nodo v en árbol T se define recursivamente:

- ❑Si v es un nodo externo, entonces la altura de v es 0
- ❑Otro caso, la altura de v es uno más la máxima altura de los hijos de v

La altura de un árbol T es la altura de la raíz

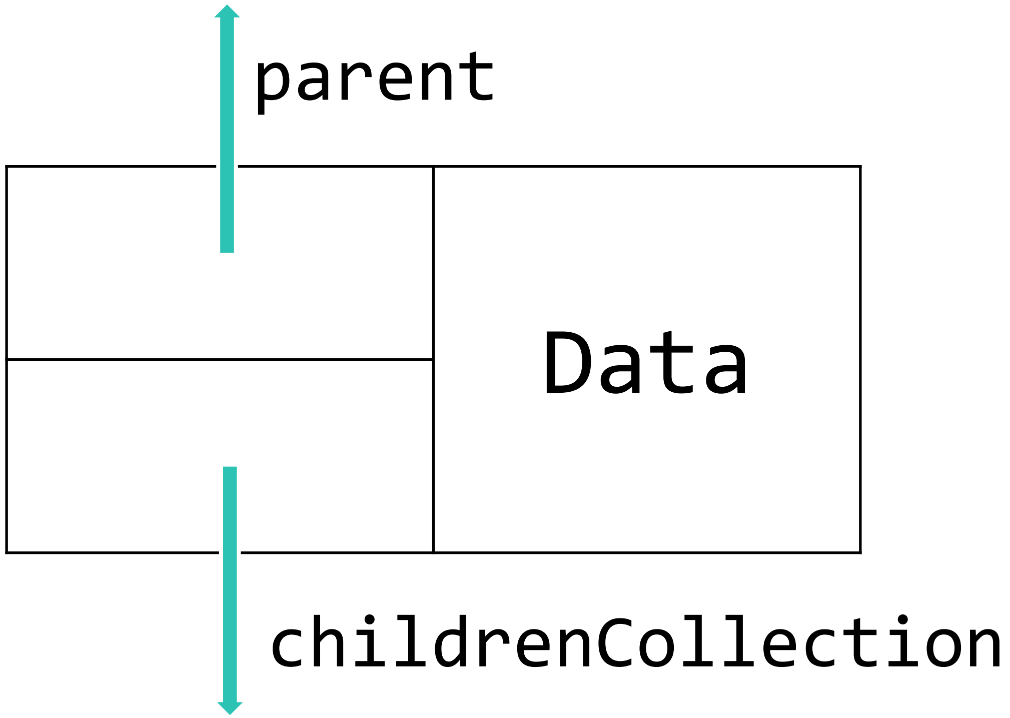
Seudocodigo

```
height(T,v)
1. if T.isExternal(v) //caso base
2.     return 0
3. else //llamado recursivo
4.     int h = 0
5.     for (each child w of v in T)
6.          h = max(h,height(t,w))
7.     return 1+h
```

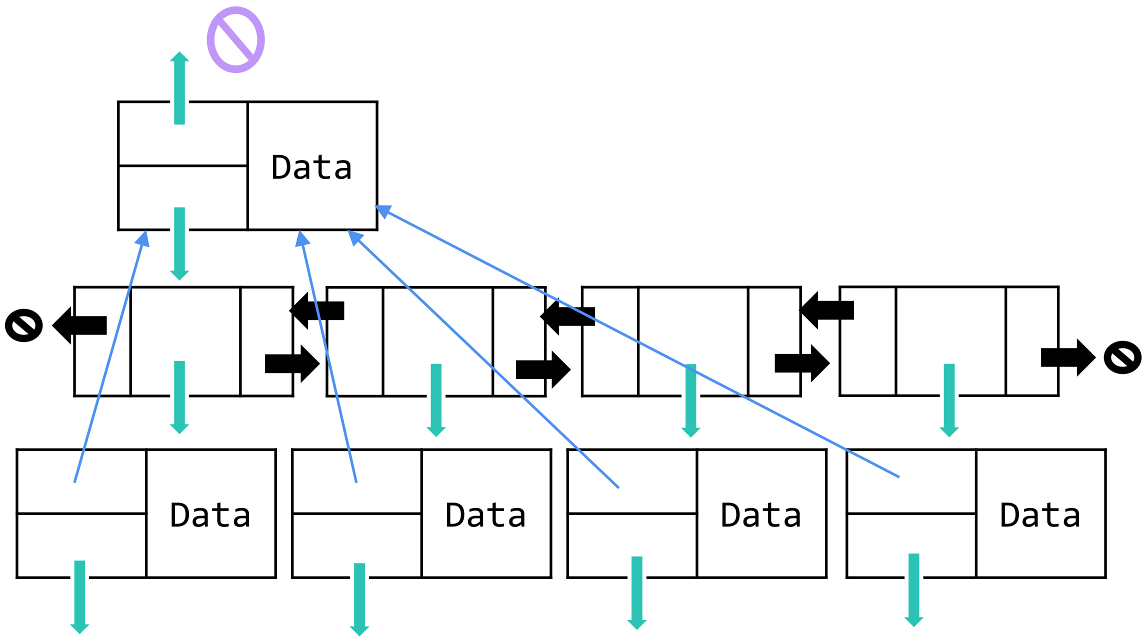
Arboles

Implementación

- Nodo de un árbol general



Representación gráfica de un nodo de un árbol genérico

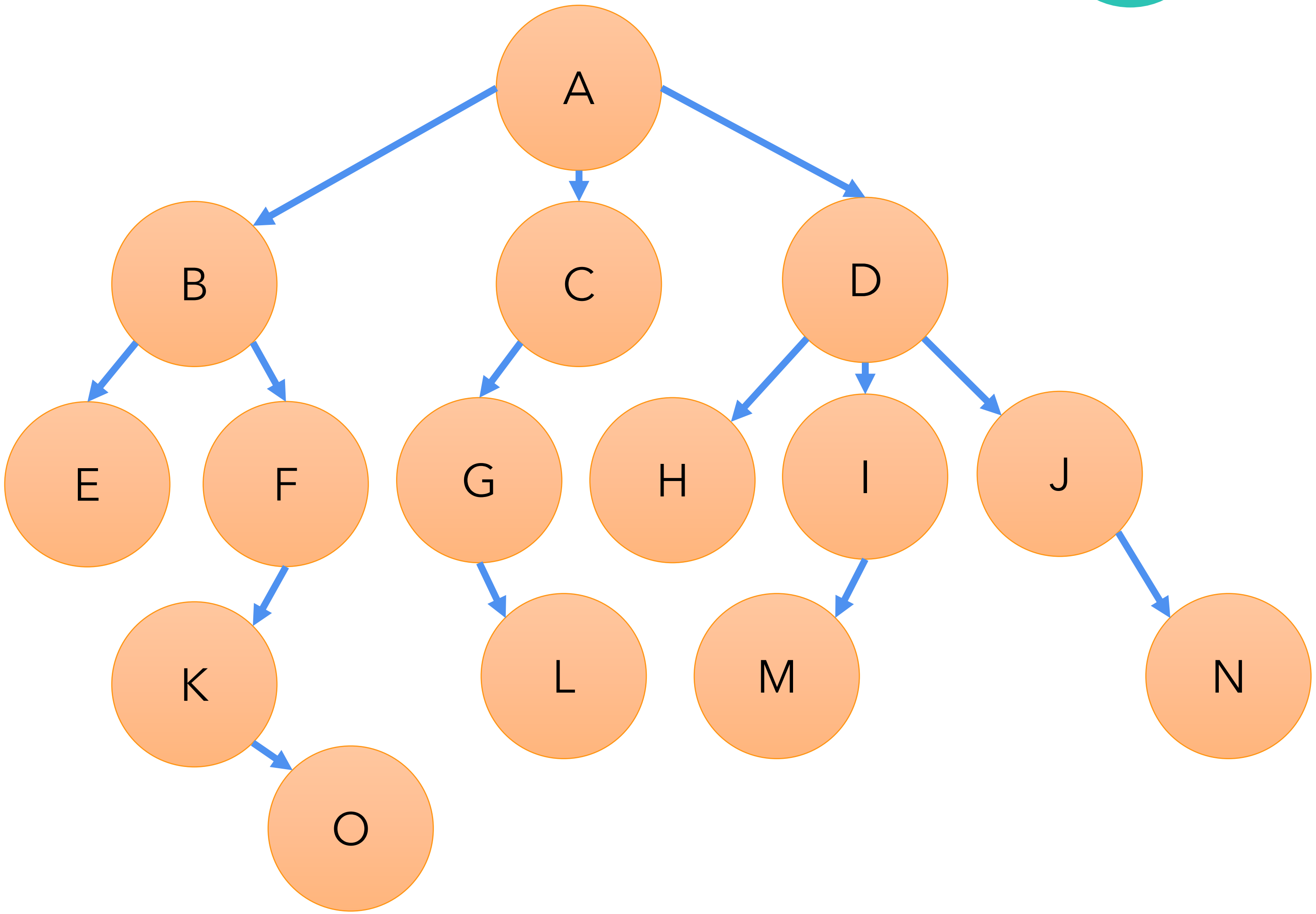


Representación gráfica de un árbol genérico

Arboles

Ejemplo

- Raíz: A
- E y F son hermanos
- D es un ancestro de N
- O es un descendiente de F
- Nodos externos: E, O, L, H, M, y N
- Nodos internos: A, B, C, D, F,G, I, J, K
- Camino
    - A-D-J-N
    - B - F- K - O



Arboles

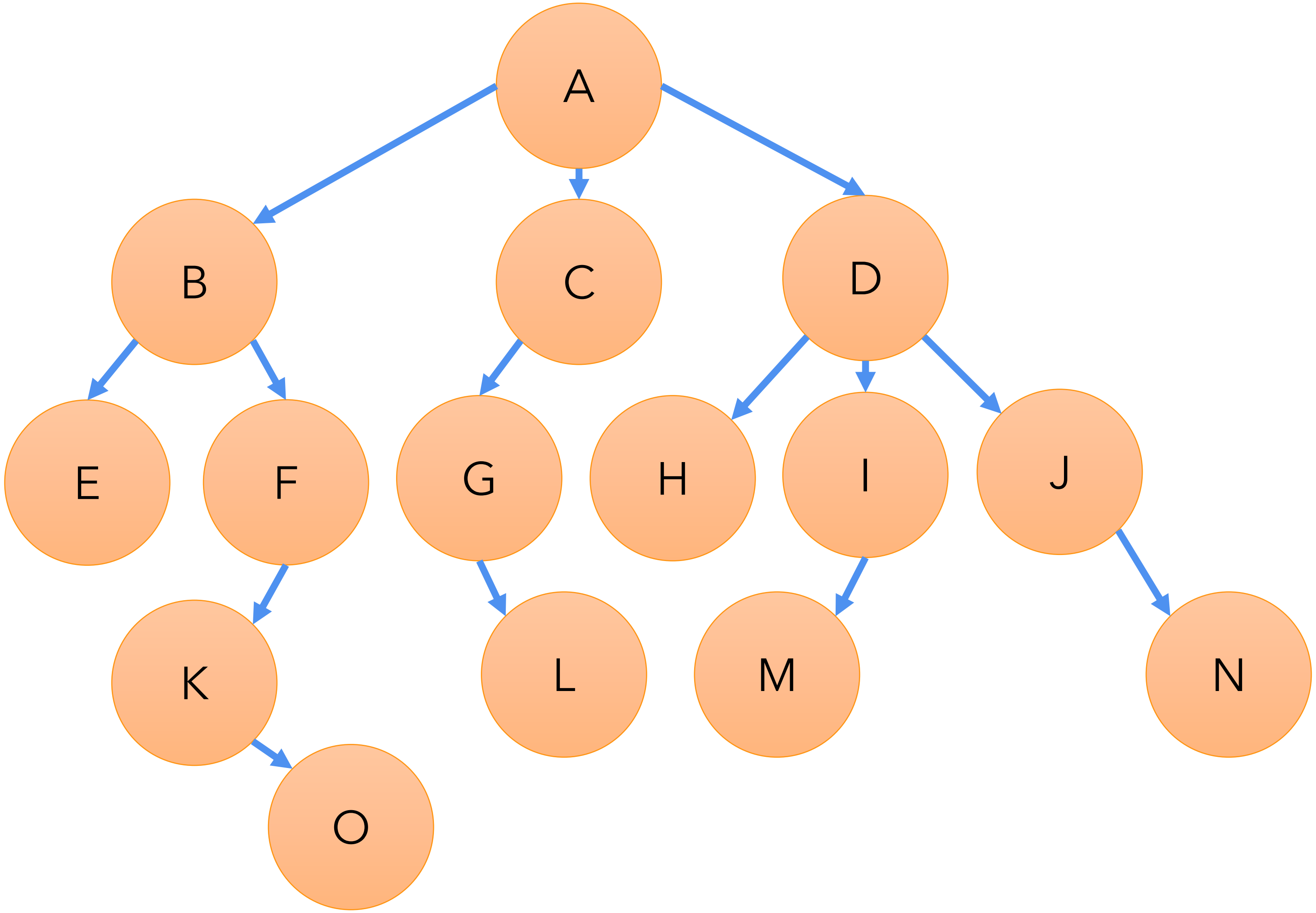
Ejemplo

Profundidad

- Depth(A) = 0
- Depth(G) = 2
- Depth(N) = 3
- Depth(O) = 4

Altura:

- Height(N) = 0
- Height(O) = 0
- Height(D) = 2
- Height(B) = 3
- Height(A) = 4



Árbol Binario de Búsqueda

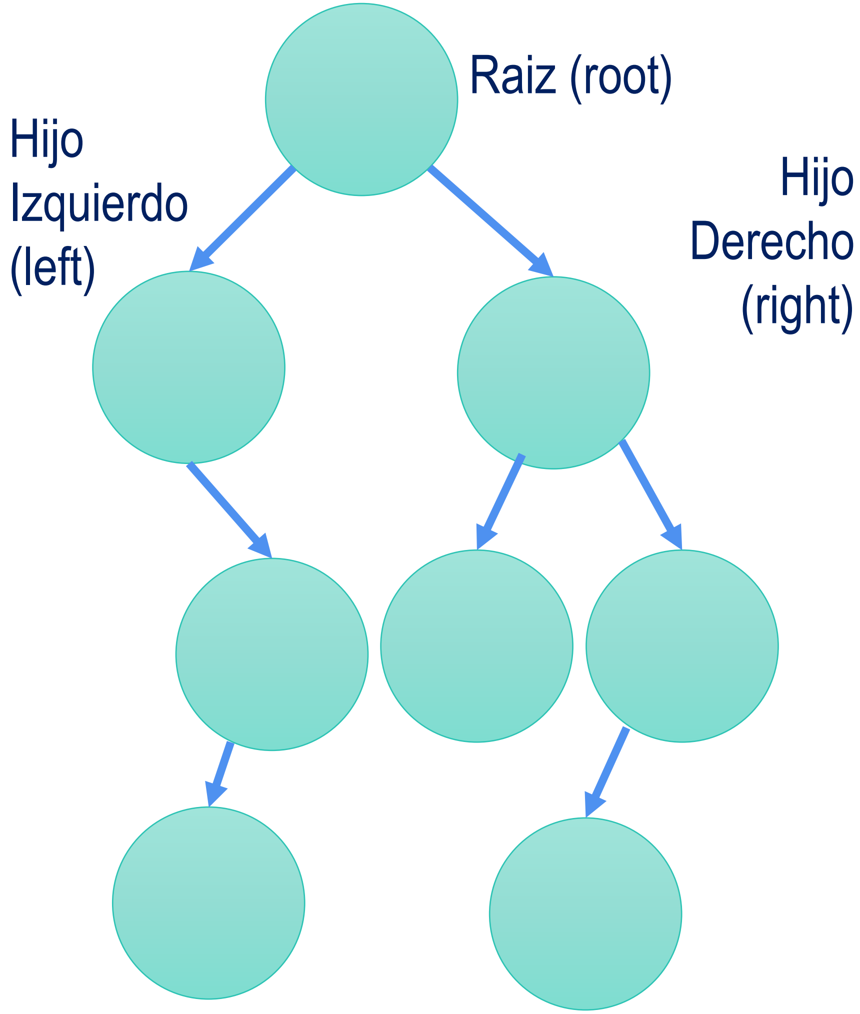
- ❑ Arboles
- ❑ **Arboles binarios**
- ❑ Arboles binarios de búsqueda
- ❑ Arboles balanceados

Arboles Binarios

Definición formal

Un árbol binario es un **árbol ordenado** con las siguientes propiedades:

- ❑Cada nodo tiene máximo 2 hijos
- ❑Cada nodo esta etiquetado como **hijo izquierdo** (left) o **hijo derecho** (right)
- ❑El hijo izquierdo precede al hijo derecho en el orden de los hijos

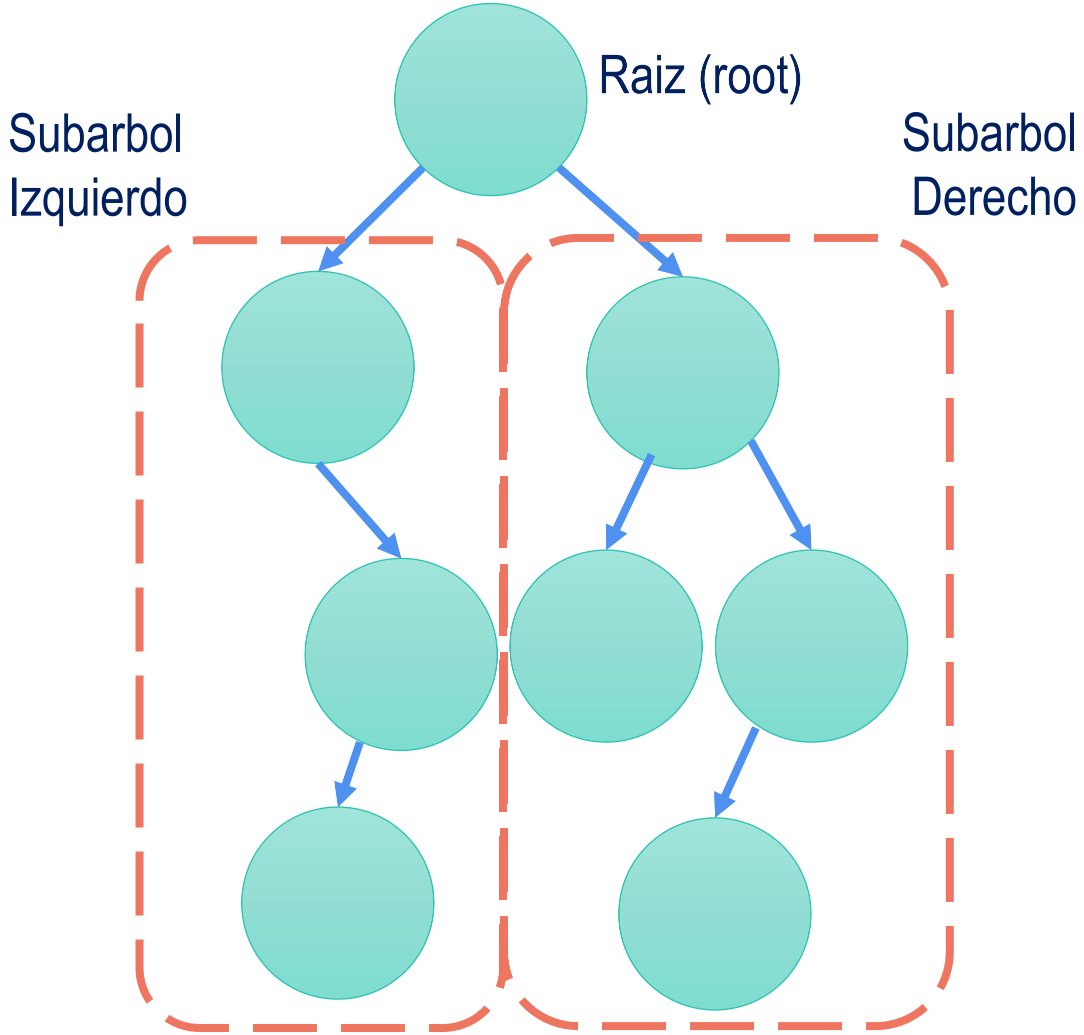


Arboles Binarios

Definición recursiva

Un árbol binario es un árbol vacío o un árbol T que consiste de:

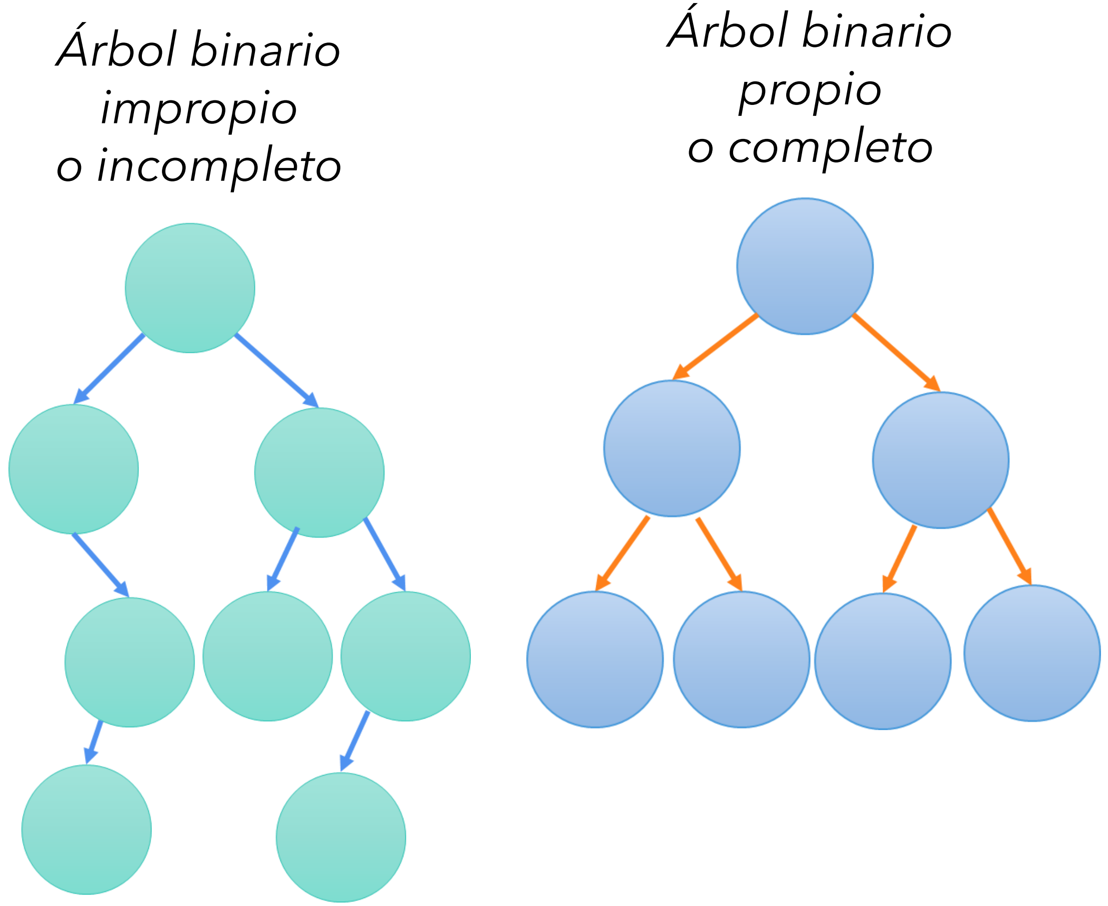
- ❑Un nodo raíz r
- ❑Un árbol binario denominado el subárbol izquierdo de T
- ❑Un árbol binario denominado el subárbol derecho de T



Arboles Binarios

Conceptos básicos

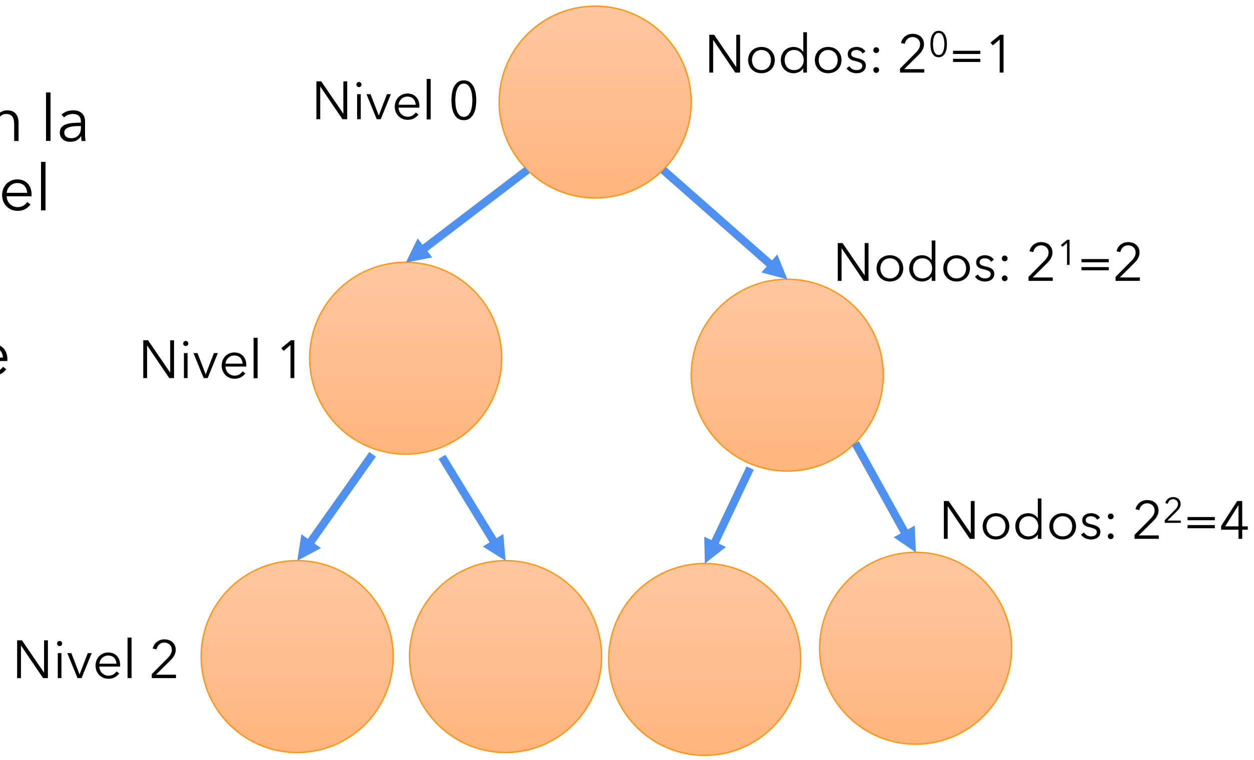
- ❑Un árbol binario propio o completo es aquel en el que cada nodo tiene 0 o 2 hijos
- ❑Es decir, cada nodo interno del árbol binario completo tiene 2 hijos



Arboles Binarios

Conceptos básicos

- El conjunto de nodos de un árbol T en la misma profundidad d conforma el nivel d de T
- En un árbol binario, cada nivel d tiene máximo 2d nodos



Arboles Binarios

Implementación

Nodo de un árbol binario:

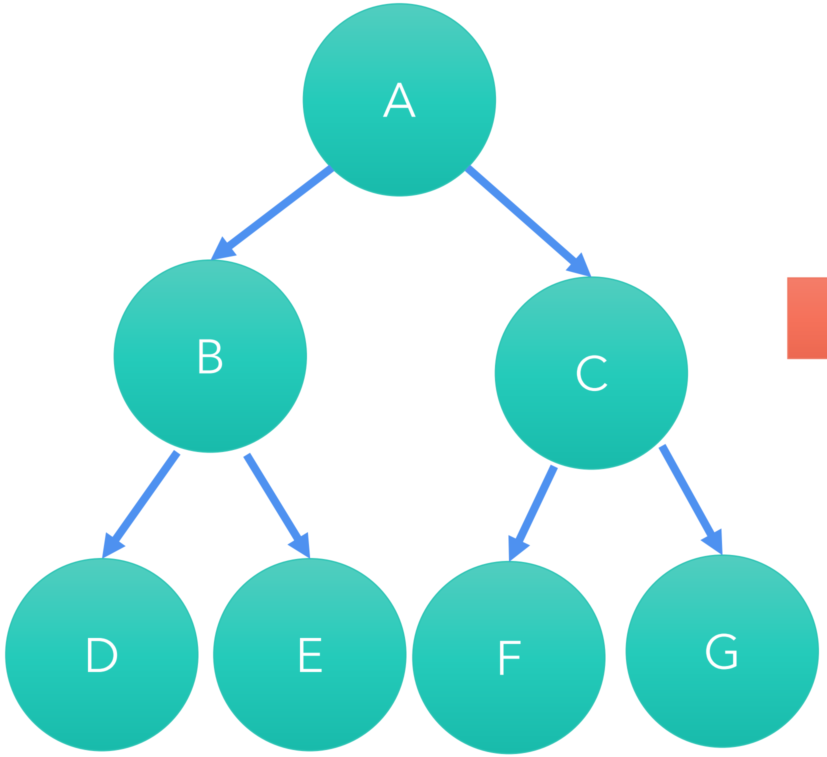
Representación gráfica de un nodo doble

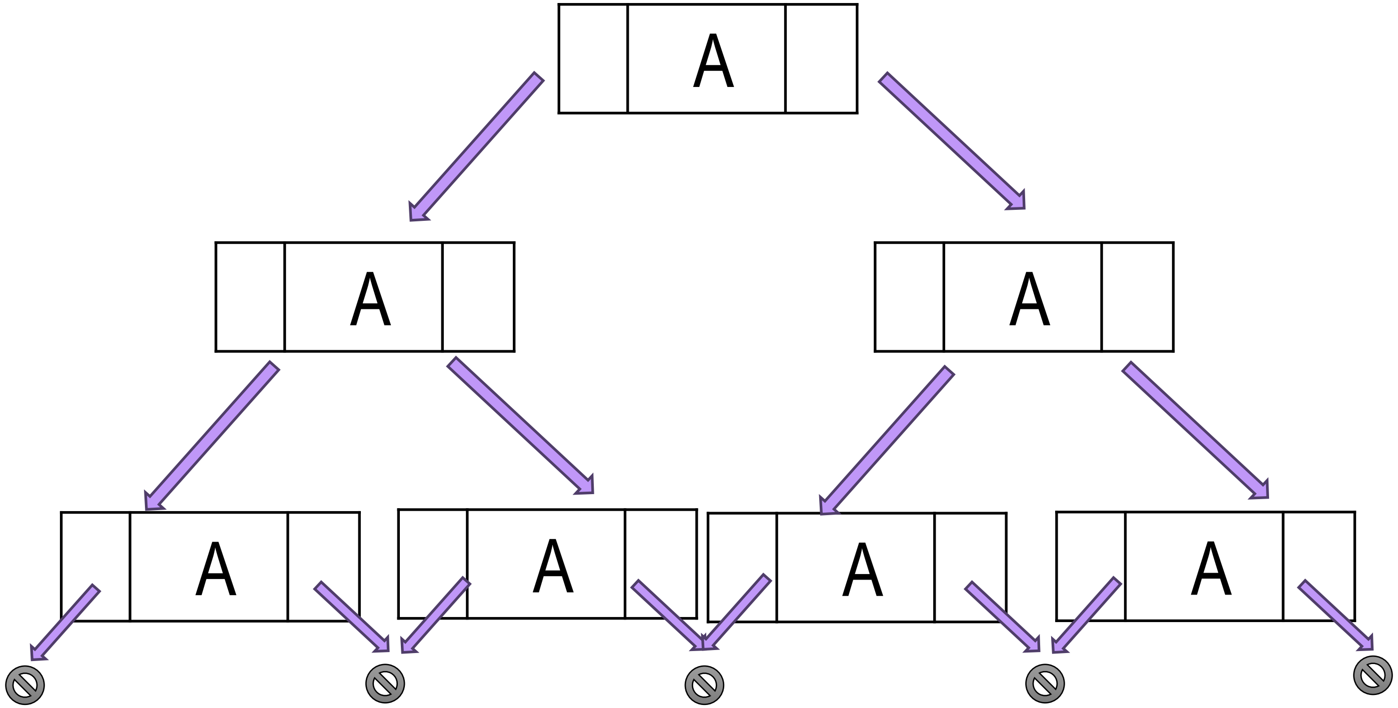
Node
- data: Object + left: Node + right: Node
+ Node() + Node(Object e) + getLeft(): Node + getRight(): Node + getData(): Object + setLeft(Node n) + setRight(Node n) + setFata(Object e)

- ❑ La implementación más sencilla es igual al nodo doble
- ❑ Esta no tiene un registro hacia el padre, sin embargo, se puede determinar mediante un algoritmo recursivo

Arboles Binarios

Implementación





Representación gráfica conexión nodos dobles en un árbol binario

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Atributos

- Se mantiene un apuntador a la raíz del árbol (**root**)
- Opcionalmente, se almacena el número de nodos en el árbol (**size**)

Métodos

- Estado del árbol:
  - isEmpty, isRoot, isInternal
  - hasLeft, hasRight
  - depth, height
- Acceso:
  - root
  - left, right
  - parent
- Modificación:
  - addRoot
  - insertLeft, insertRight
  - remove

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Atributos

- Se mantiene un apuntador a la raíz del árbol (**root**)
- Opcionalmente, se almacena el número de nodos en el árbol (**size**)

Constructor vacío:

```
BinaryTree()
1. root = null
2. size = 0
```

Tamaño del árbol:

```
size()
1. return size
```

$\Theta(1)$

```
isEmpty()
2. return size==0
```

$\Theta(1)$

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Estado del árbol:

- isEmpty, isRoot, isInternal
- hasLeft, hasRight
- depth, height

Métodos booleanos:

```
isRoot(Node v)
   return v==root
```
$\Theta(1)$

```
isInternal(Node v)
   return hasLeft(v)||hasRight(v)
```
$\Theta(1)$

```
hasLeft(Node v)
   return v.getLeft()!=null
```
$\Theta(1)$

```
hasRight(Node v)
   return v.getRight()!=null
```
$\Theta(1)$

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Estado del árbol:

- isEmpty, isRoot, isInternal
- hasLeft, hasRight
- depth, height

Métodos de altura y profundidad:

```
depth(DoubleNode v)
if isRoot(v) //caso base
    return 0
else //llamado recursivo
    return 1 + depth(parent(v))
```

$\Theta(1)$

$O(n^2)$

```
height(Node v)
if !isInternal(v) //caso base
    return 0
else //llamado recursivo
    int h = 0
    h = max(height(left(v)),height(right(v)))
    return 1+h
```

$\Theta(1)$

$O(n)$

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Acceso:

- root
- left, right
- parent

Métodos acceso a nodos:

```
root()
   return root
```

$\Theta(1)$

```
left(Node v)
   return v.getLeft()
```

$\Theta(1)$

```
right(Node v)
   return v.getRight()
```

$\Theta(1)$

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Acceso:

- root
- left, right
- parent

Métodos acceso a nodos:

`Parent(Node v):`

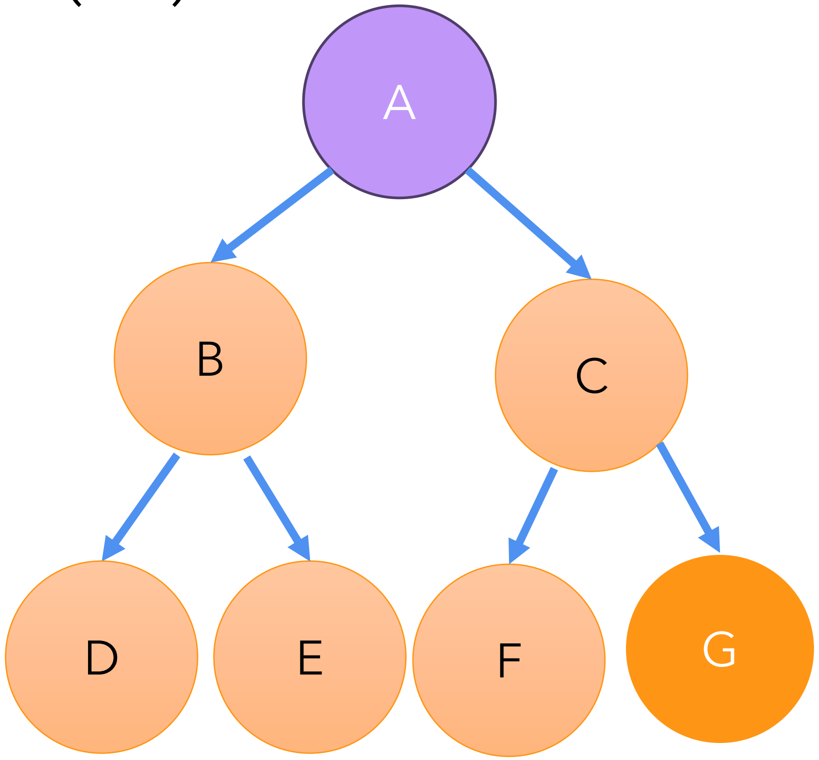
Dado que no tenemos un enlace directo al padre en esta implementación, podemos usar una cola (queue) para extraer el padre de un nodo desde la colección.

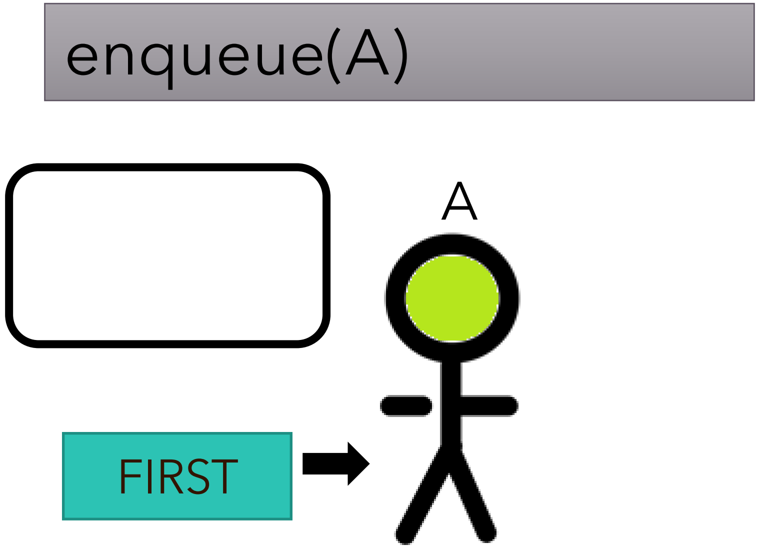
El algoritmo consiste en recorrer desde la raíz el árbol, nivel por nivel; los hijos del nodo explorado se insertan en una cola; si uno de los hijos es el nodo v detenemos el proceso.

Arboles Binarios

Implementación

`Parent("G")`

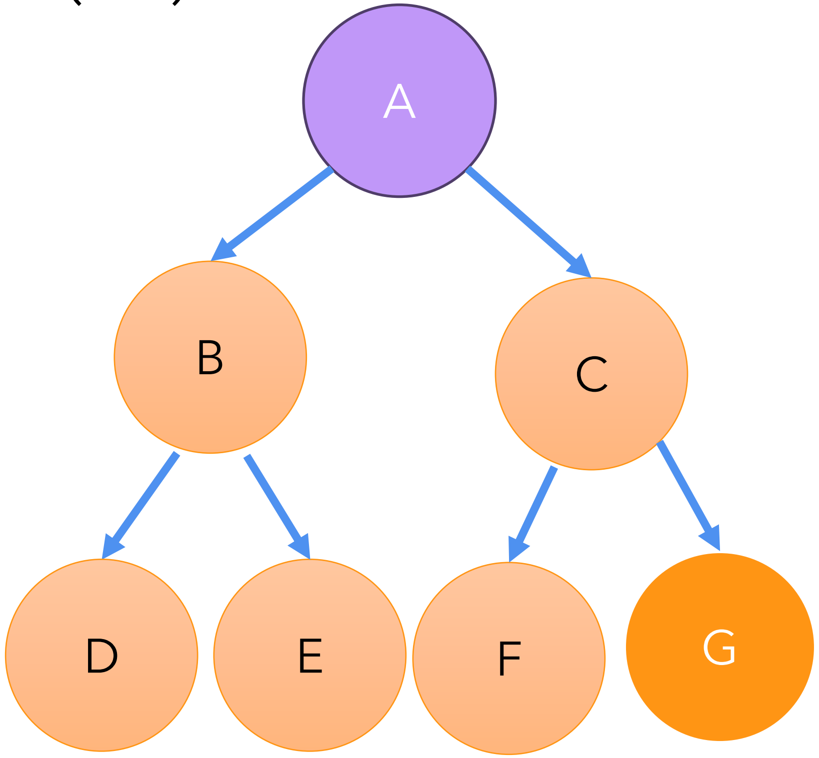




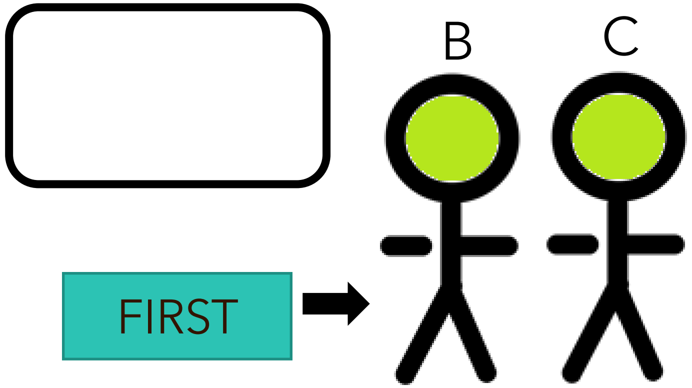
Arboles Binarios

Implementación

```
Parent("G")
```



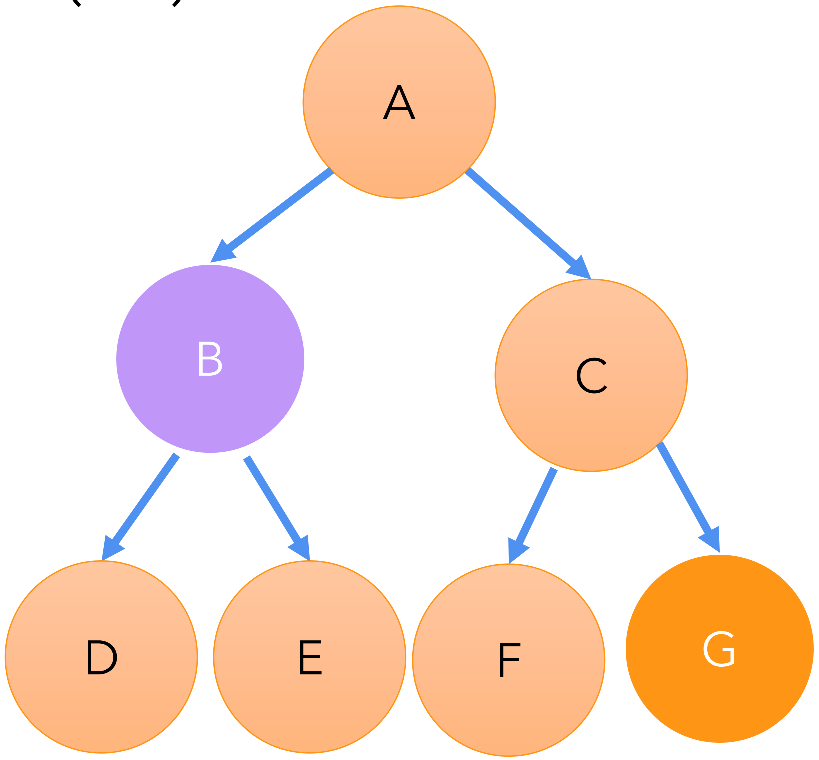

```
N = dequeue()
If N.left()!="G"&& N.right()!="G"
   enqueue(n.left())
   enqueue(n.right())
```



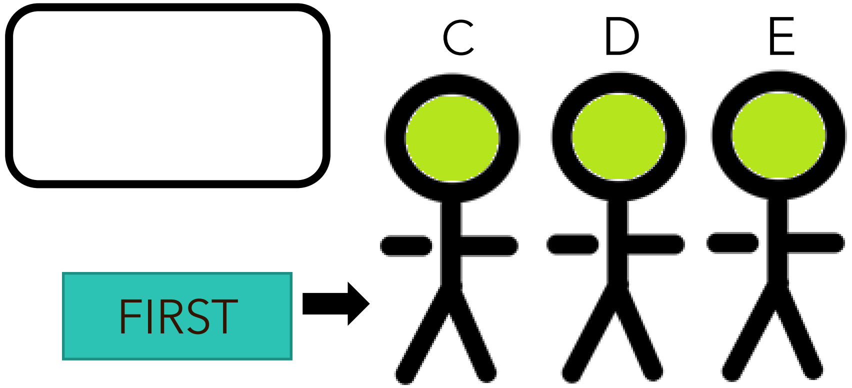
Arboles Binarios

Implementación

```
Parent("G")
```



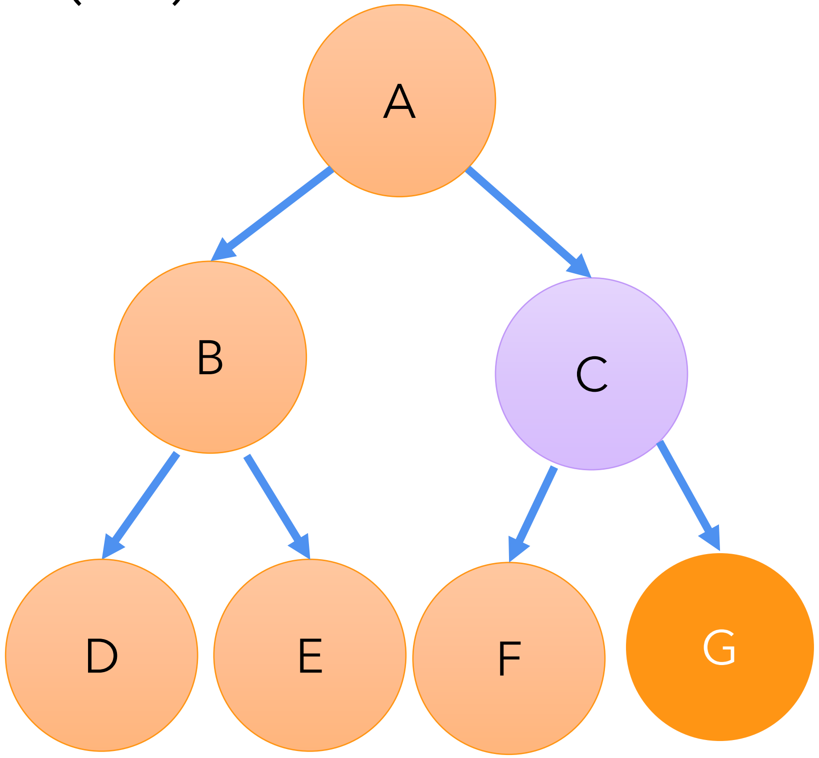

```
N = dequeue()
If N.left()!="G"&& N.right()!="G"
   enqueue(n.left())
   enqueue(n.right())
```



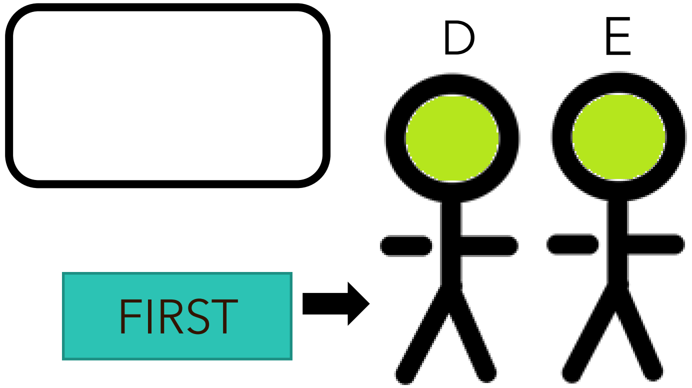
Arboles Binarios

Implementación

`Parent(“G”)`




```
N = dequeue()
If N.left()!=“G”&& N.right()!=“G”
   enqueue(n.left())
   enqueue(n.right())
```



Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Acceso:

- root
- left, right
- parent

Métodos acceso a nodos:

```
Parent(Node v)
1. if isRoot(v)
2.    return null
3. else
4.      Queue Q = new Queue()
5.      Q.enqueue(root)
6.      Node temp = root
7.   while (!Q.isEmpty() &
             left(Q.first())!=v
             & right(Q.first())!=v)
8.      temp = Q.dequeue()
9.      if hasLeft(temp)
10.        Q.enqueue(left(temp))
11.     if hasRight(temp)
12.        Q.enqueue(right(temp))
13.     return temp
```

O(n)

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Modificación:

- addRoot
- insertLeft, insertRight
- remove

Métodos para agregar datos:

```
addRoot(Object e)
1. root = new Node(e)
2. size = 1
```

Θ(1)

```
insertLeft(Node v, Object e)
1. Node left = new Node(e)
2. v.setLeft(left)
3. size++
```

Θ(1)

```
insertRight(Node v, Object e)
1. Node right = new Node(e)
2. v.setRight(right)
3. size++
```

Θ(1)

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Modificación:

- addRoot
- insertLeft, insertRight
- remove

Métodos para eliminar un nodo:

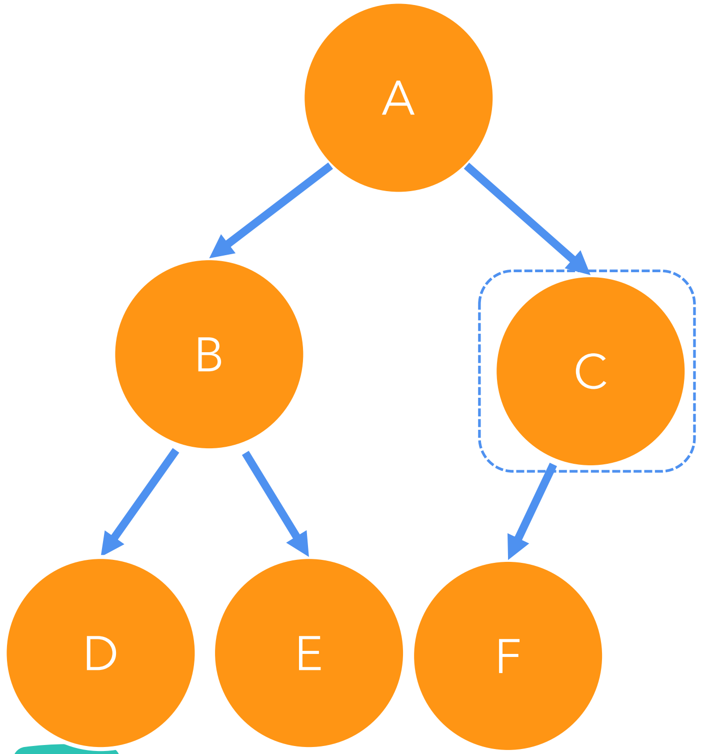
`remove(Node v):` elimina el nodo v del árbol. Importante: si el nodo tiene dos hijos, se elimina todo el subárbol.

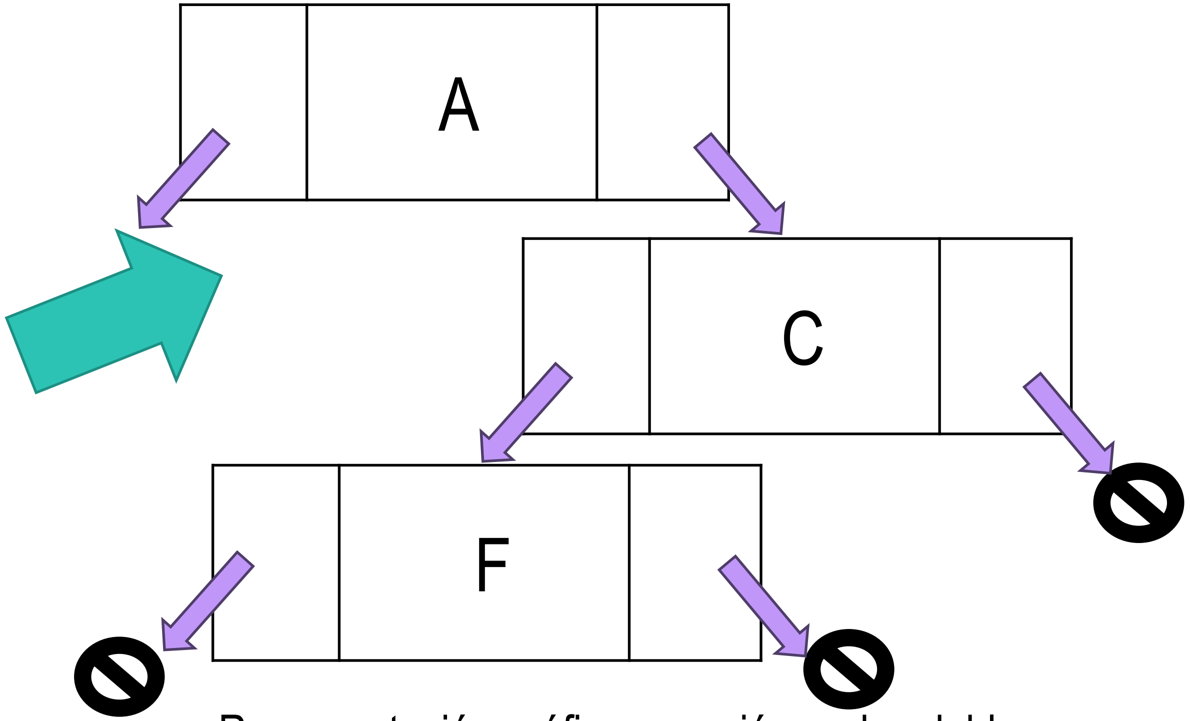
Arboles Binarios

Implementación

Métodos para eliminar un nodo:

Caso 1: El nodo a eliminar tiene un hijo





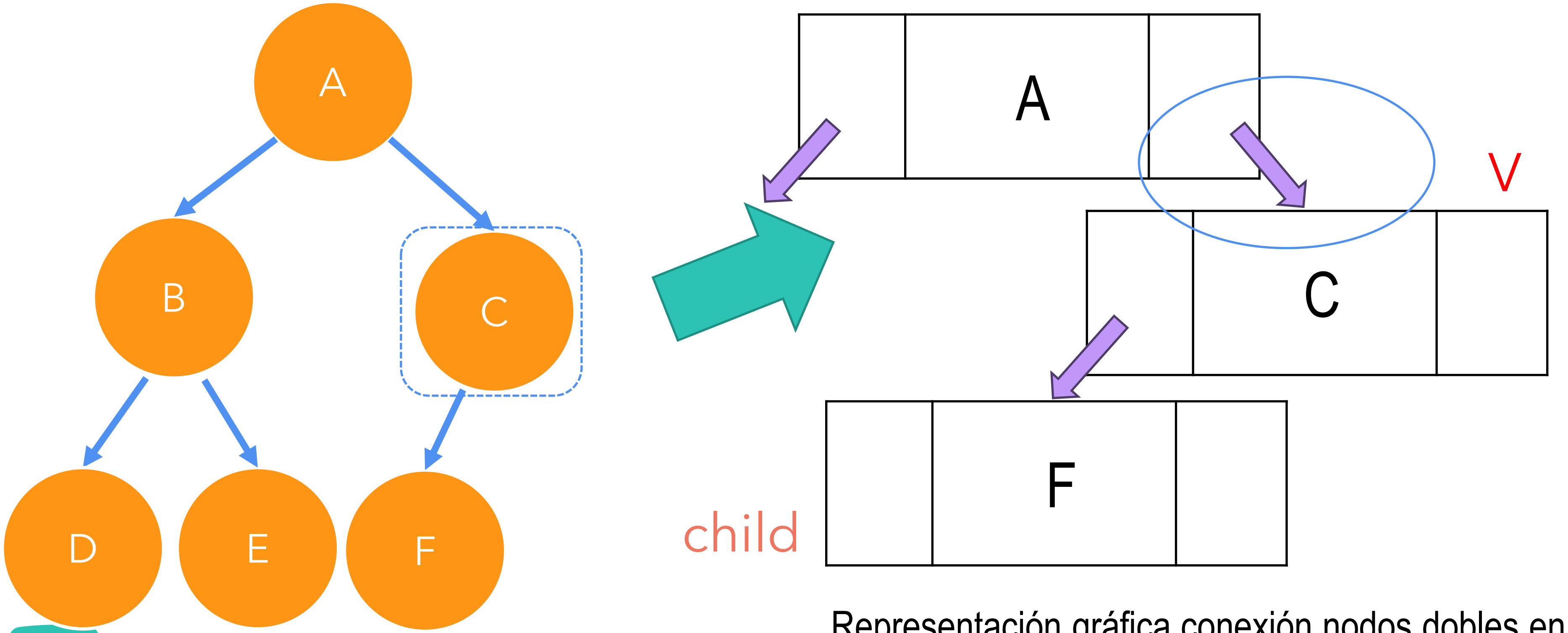
Representación gráfica conexión nodos dobles en un árbol binario

Arboles Binarios

Implementación

Métodos para eliminar un nodo:

- Si el hijo es el izquierdo child = left(v)
- Si el hijo es el derecho child = right(v)



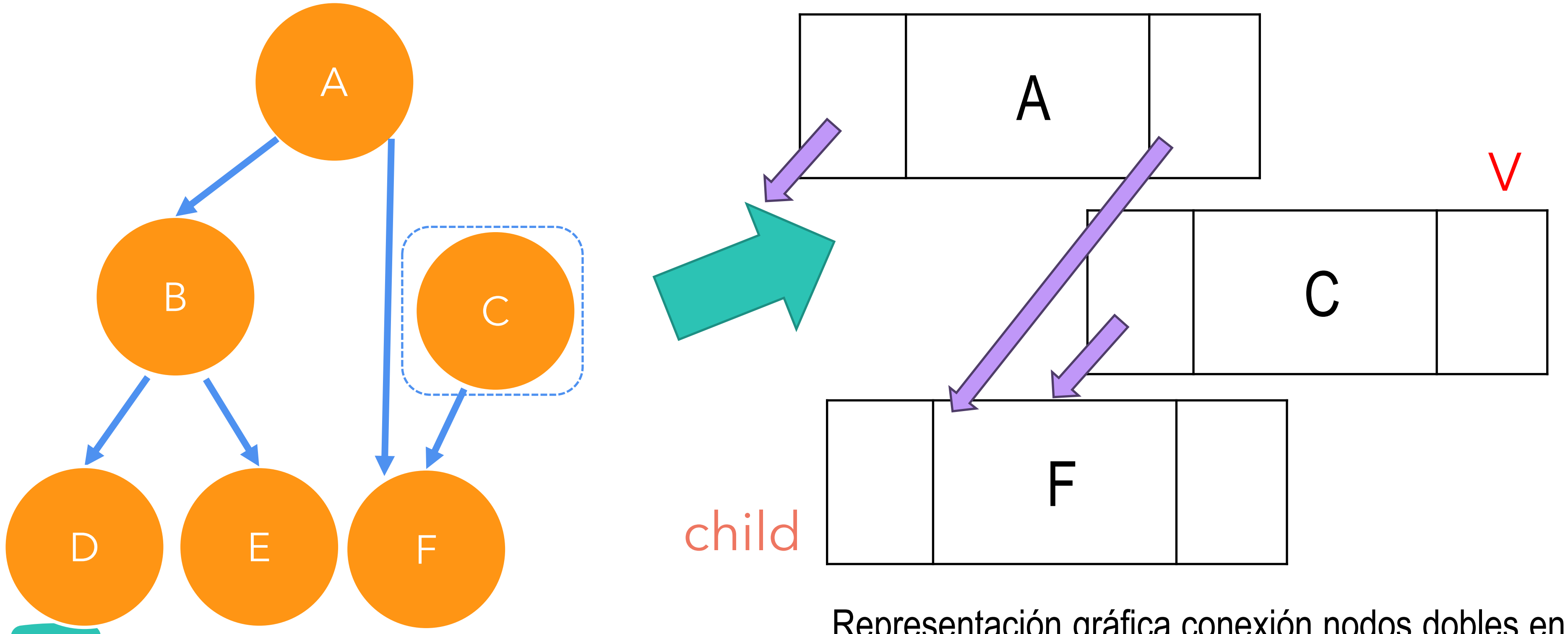
Representación gráfica conexión nodos dobles en un árbol binario

Arboles Binarios

Implementación

Métodos para eliminar un nodo:

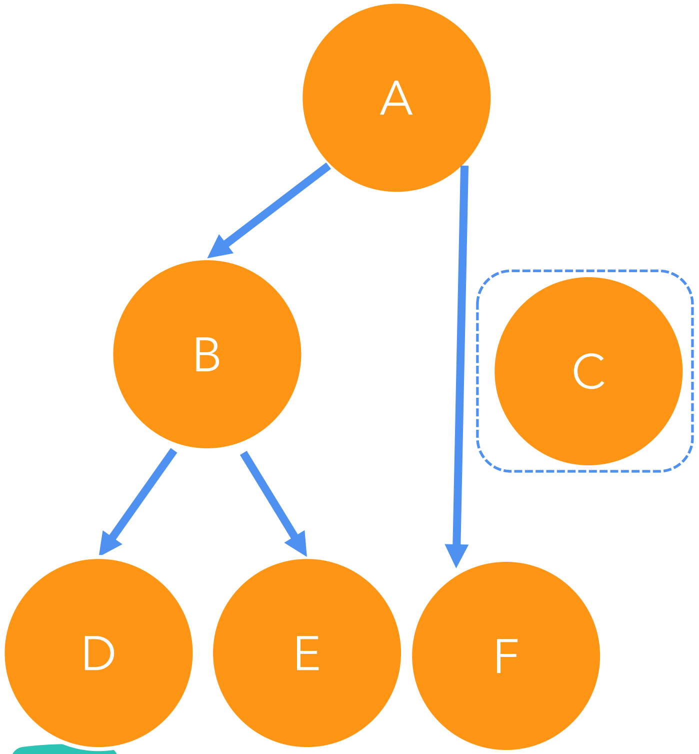
- Si v es el hijo izquierdo: parent(v).setLeft(child)
- Si v es el hijo derecho: parent(v).setRight(Child)



Representación gráfica conexión nodos dobles en un árbol binario

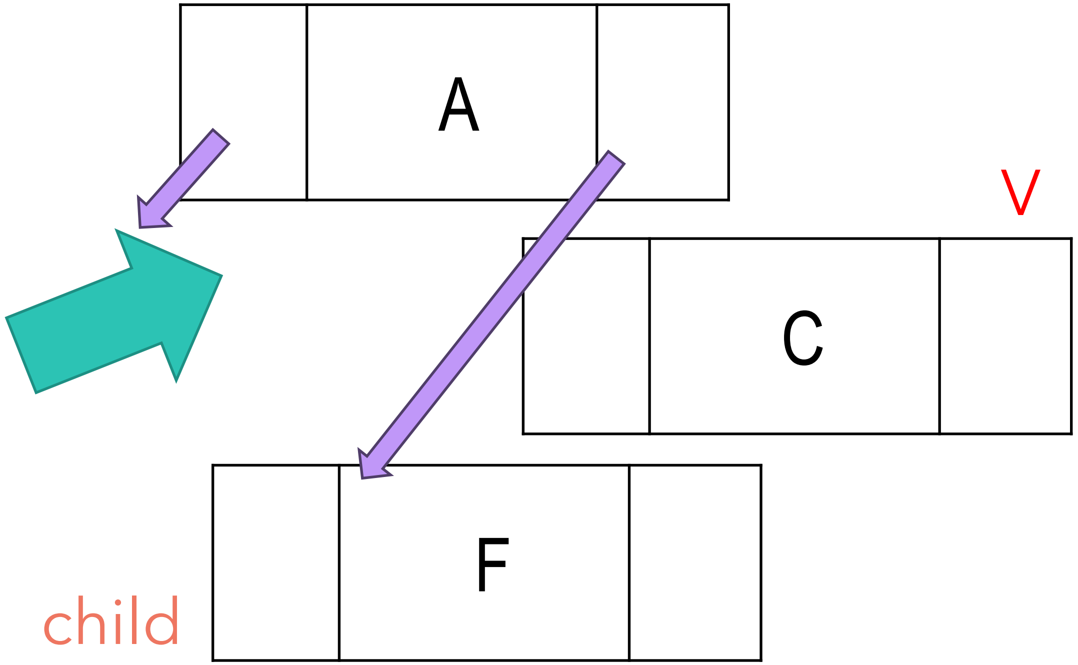
Arboles Binarios

Implementación



Métodos para eliminar un nodo:

- v.setLeft(null)
- v.setRight(null)



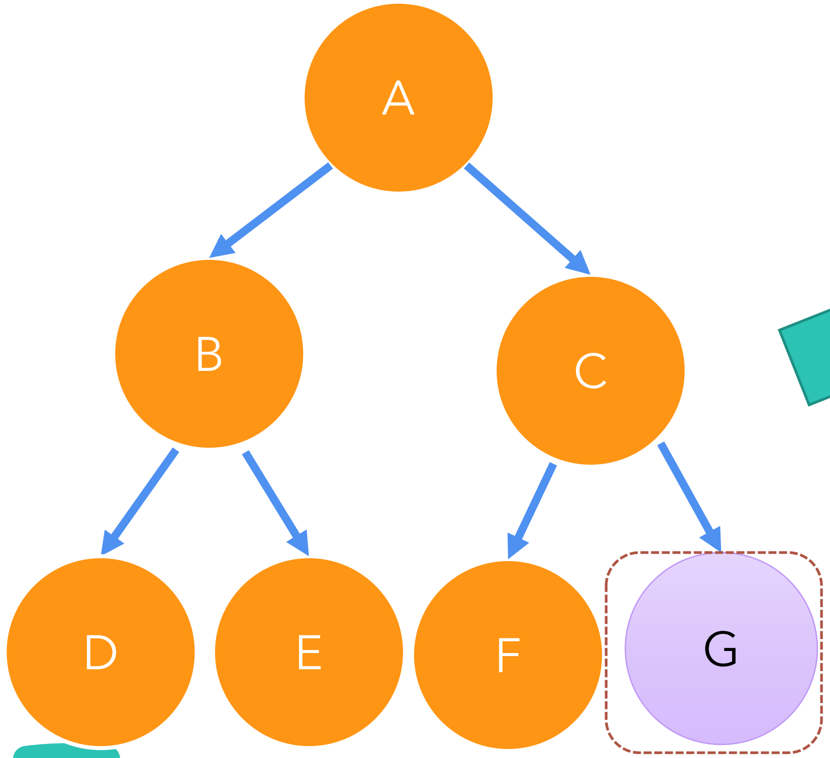
Representación gráfica conexión nodos dobles en un árbol binario

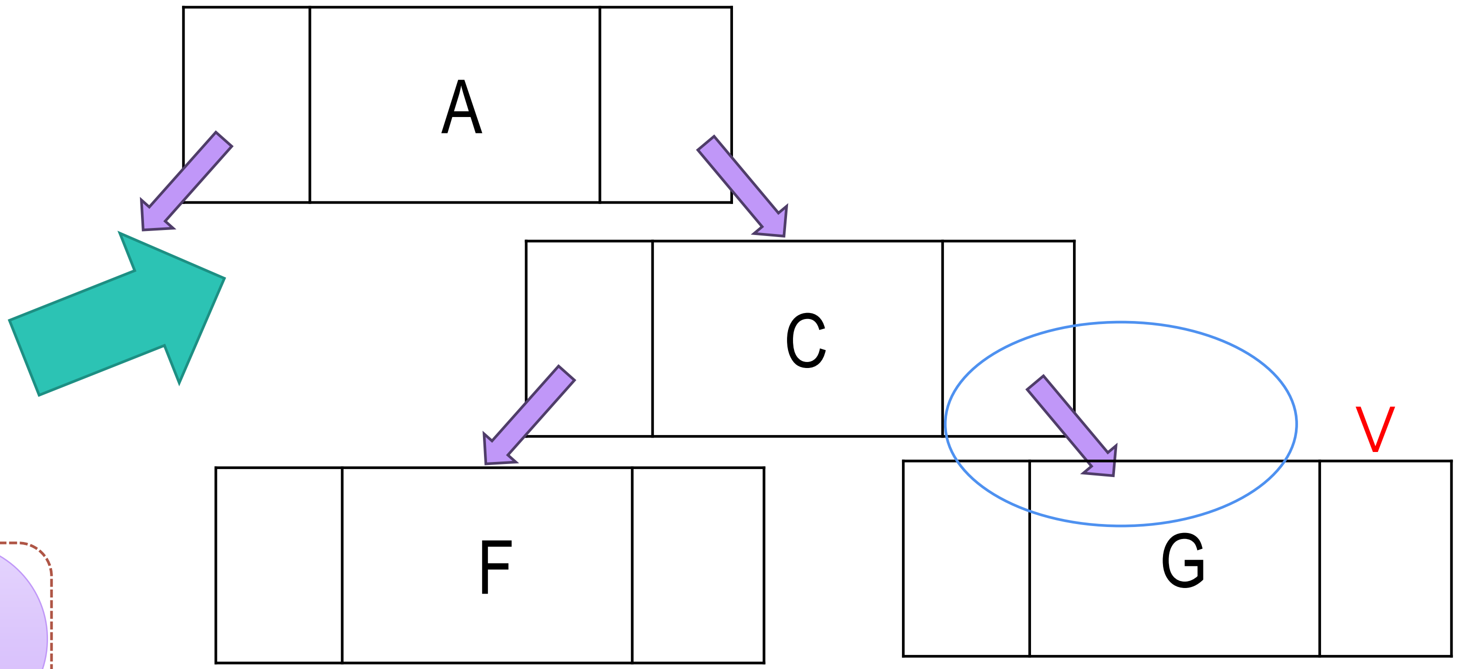
Arboles Binarios

Implementación

Métodos para eliminar un nodo:

Caso 2: El nodo a eliminar no tiene hijos

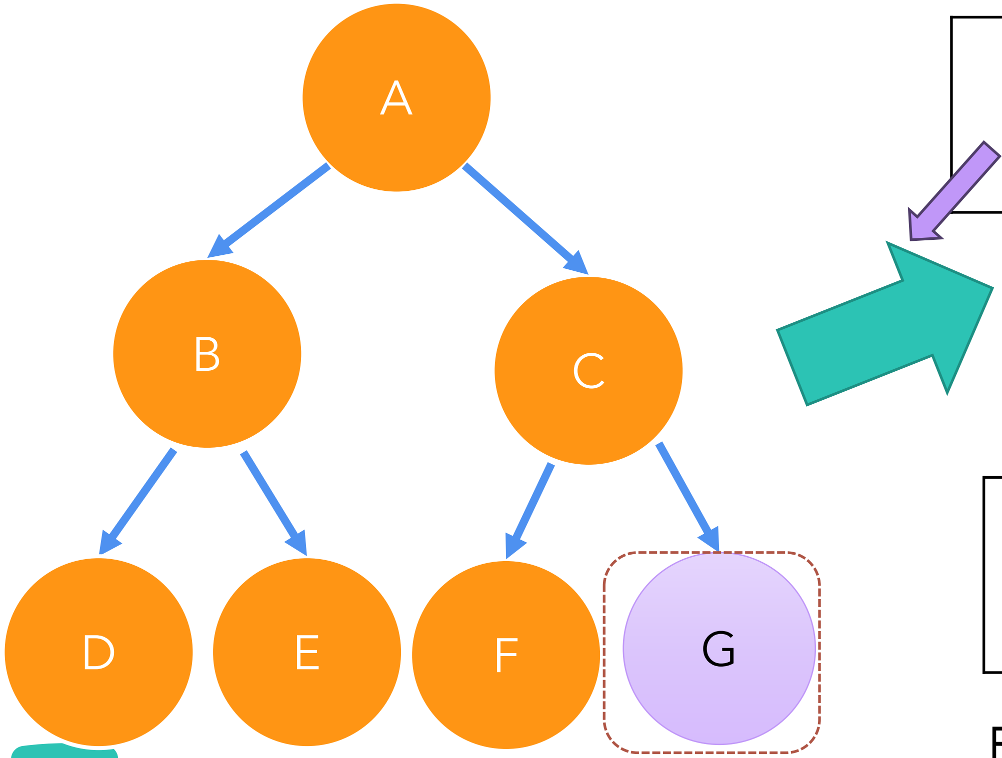




Representación gráfica conexión nodos dobles en un árbol binario

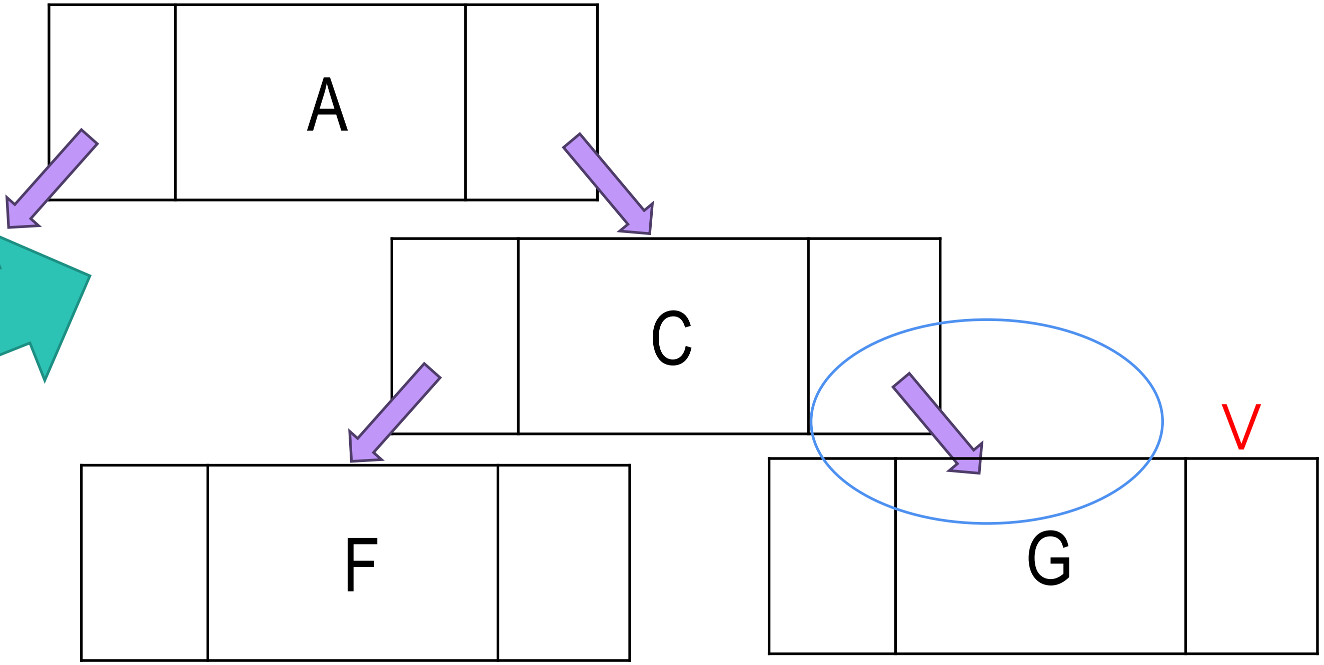
Arboles Binarios

Implementación



Métodos para eliminar un nodo:

Si es el hijo izquierdo: parent(v).setPrev(null)
Si es el hijo derecho: parent(v).setNext(null)



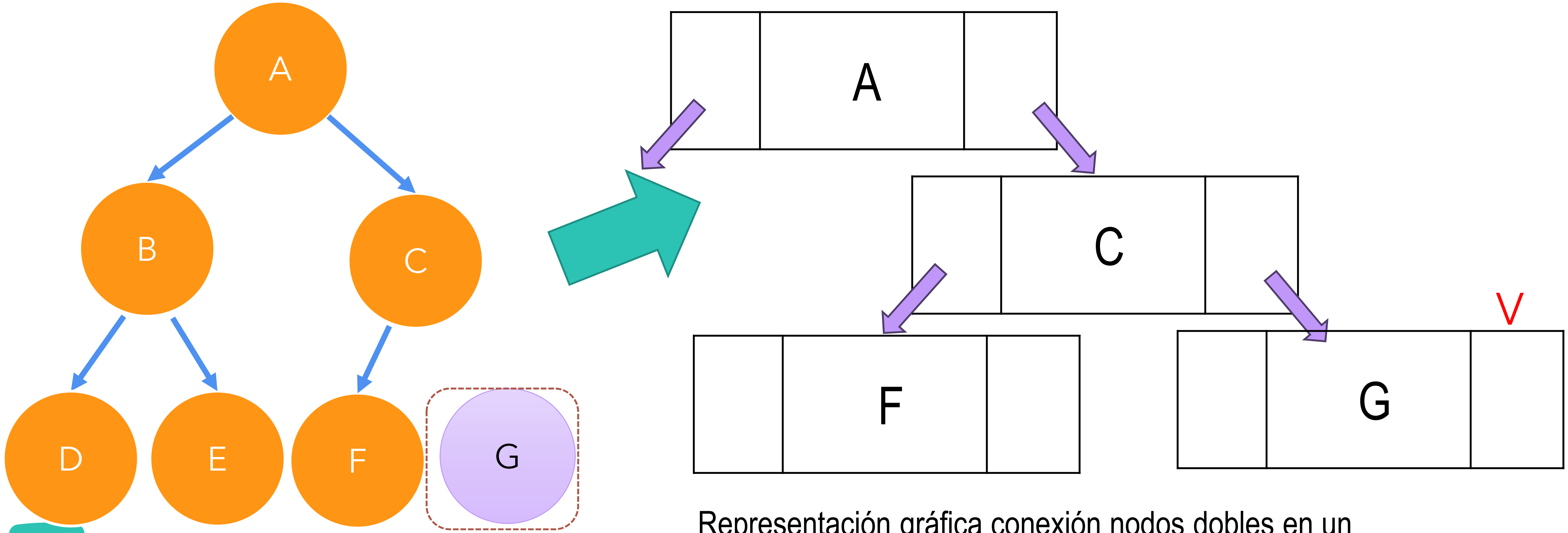
Representación gráfica conexión nodos dobles en un árbol binario

Arboles Binarios

Implementación

Métodos para eliminar un nodo:

Si es el hijo izquierdo: parent(v).setPrev(null)
Si es el hijo derecho: parent(v).setNext(null)



Representación gráfica conexión nodos dobles en un árbol binario

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Métodos

❑Modificación:

- addRoot
- insertLeft, insertRight
- remove

Métodos para eliminar un nodo:

```
remove(DoubleNode v)
1. Node p = parent(v)
    //v tiene al menos un hijo - caso 1
2. if hasLeft(v)||hasRight(v)
3.    if hasLeft(v)
4.       Node child = left(v)
5.    else
6.       Node child = right(v)
    //se conecta el hijo de v al padre
7.    if left(p)==v
8.       p.setPrev(child)
9.    else
10.      p.setNext(child)
    //se desconecta el nodo v
11.   v.setPrev(null)
12.   v.setNext(null)
13.else
    //v no tiene hijos - caso 2
14.   if left(p)==v
15.      p.setPrev(null)
16.   else
17.      p.setNext(null)
18.size--
```

O(n)

O(1)

Arboles Binarios

Implementación

BinaryTree	
-root: Node -size: int	
+BinaryTree() +size(): int +isEmpty():Boolean +isRoot(Node v):Boolean +isInternal(Node v):Boolean +hasLeft(Node v): Boolean +hasRight(Node v): Boolean +root():Node +left(Node v):Node	+right(Node v):Node +parent(Node v):Node +depth(Node v): int +height(Node v):int +addRoot(Object e) +insertLeft(Node v, Object e) +insertRight(Node v, Object e) +remove(Node v)

Operación	Complejidad
parent()	O(n)
depth()	$O(n^2)$ Árbol completo: O(n lg n)
height()	O(n) Árbol completo: O(lg n)
addRoot()	Θ(1)
insertLeft()	Θ(1)
insertRight()	Θ(1)
remove()	O(n)