Árboles

Árboles

Definiciones

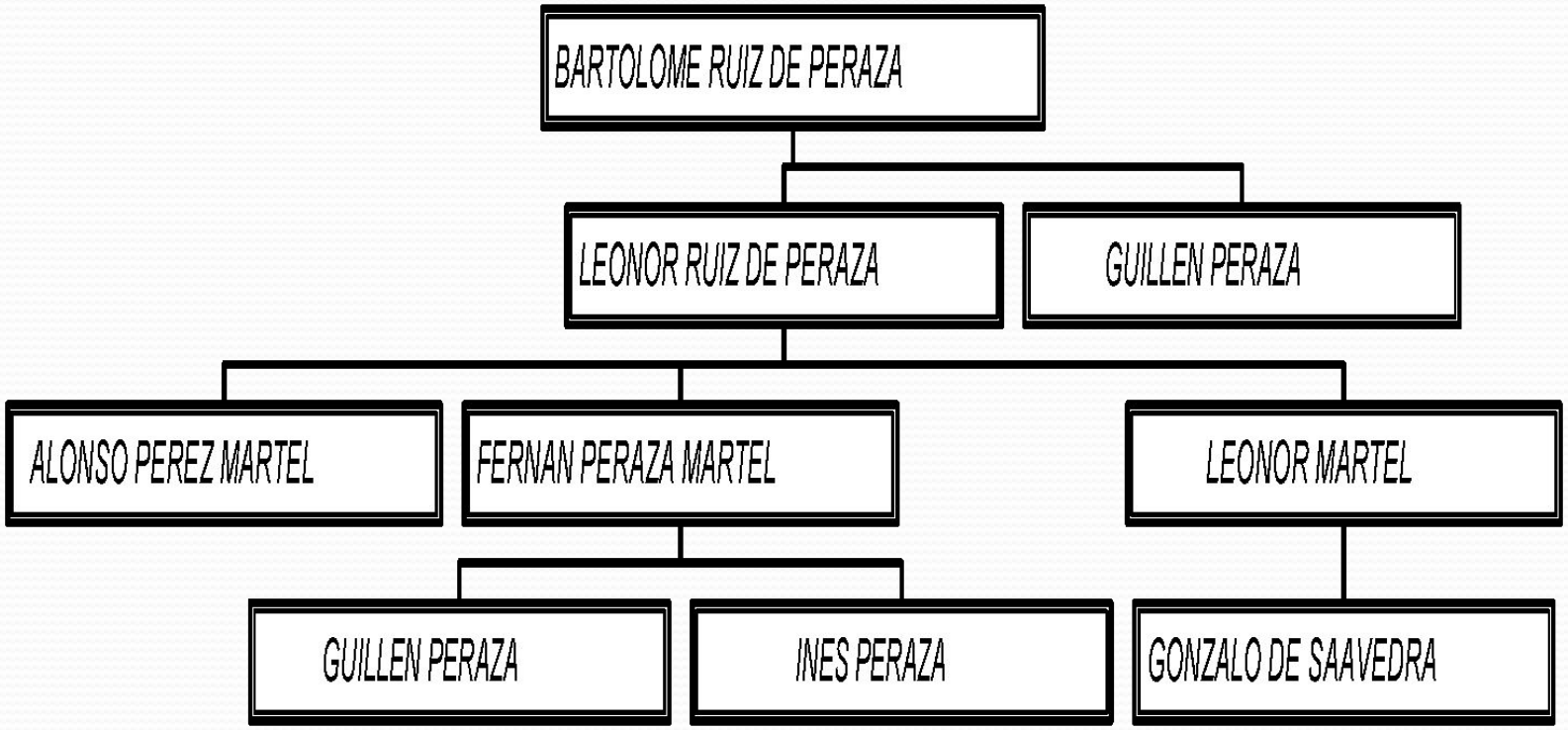
Inicio

Definiciones

- Más allá de las estructuras lineales.
  - Sólo expresan relaciones unidimensionales.
  - Aparecen las estructuras no lineales, que pueden presentar relaciones más complejas.
- Un árbol es una clase de estructura no lineal caracterizada por la relación jerárquica entre sus nodos.
- Ejemplos de estructuras jerárquicas son:
  - Los árboles familiares.
  - La clasificación decimal de libros en una biblioteca.
  - La jerarquía de posiciones en una organización.
  - El índice del temario de una asignatura.

Definiciones

- La figura muestra un árbol familiar.



Definiciones

- Formalmente un árbol puede definirse de manera recursiva como un conjunto finito de elementos tal que:
  - Si no está vacío —un árbol vacío es el que no tiene ningún nodo— se cumple que:
    - Hay un nodo especialmente designado que se llama *raíz*.
    - Los restantes nodos se distribuyen en una serie de subconjuntos disjuntos: $T_1, T_2, \ldots, T_n$ ($n \geq 0$), cada uno de los cuales es, a su vez, un árbol.
      - Cada uno de los T_i ($1 \leq i \leq n$) es un *subárbol* del árbol total.

Definiciones

- Los nodos pertenecientes a cada subárbol se dice que son descendientes de la raíz.
  - La raíz de cada uno de esos subárboles es un *hijo* de la raíz del árbol —su *padre*.
- La relación que conecta un padre con un hijo es una *rama* del árbol.
- Un conjunto de árboles separados —los que quedarán si un árbol perdiera su raíz— recibe el nombre de *bosque*.
- El *nodo raíz* de un árbol que no sea subárbol de otro no tiene ningún ascendiente y constituye el punto de partida de la jerarquía representada en la estructura.
- Los nodos que no tienen ningún descendiente se conocen como *hojas*, o terminales.

Definiciones

- Se denomina *grado de un nodo* al número de hijos que tiene.
  - Sin contar los subárboles vacíos.
  - Las hojas tienen grado cero.
- El *nivel de un nodo* es uno más la longitud de su camino desde la raíz.
  - Se entiende por *camino* de un nodo a otro a una secuencia de ramas que permite llegar al segundo nodo partiendo del primero.
  - El nivel de la raíz de un árbol es uno.
- La *altura de un nodo* se define como uno más la longitud del camino más largo desde ese nodo a sus hojas.
  - La *altura de un árbol* es la de su nodo raíz.

Definiciones

- *Árbol general* es aquel en el que no se considera ninguna limitación en el grado máximo de sus nodos.
- Los *árboles n-arios* son aquellos en los que el grado máximo de cualquiera de sus nodos es *n*
  - *n* es un entero positivo
- Desde el punto de vista de la interrelación entre las estructuras, cabe pensar que una lista lineal podría considerarse como un árbol 1-ario en el que la relación "sucesor" se expresa como "hijo".

Definiciones

especificación Árbol general **es**

usa Estructuras lineales recursivas

parámetro tipo TElemento

define TÁrbol<TElemento>

operaciones

{constructoras generadoras}

Vacío: () → (TÁrbol<TElemento>)

Crear: (TElemento,TListaRec<TÁrbol<TElemento>>) → (TÁrbol<TElemento>)

{observadoras}

Raíz : (TÁrbol<TElemento>) → (TElemento)

Hijos: (TÁrbol<TElemento>) → (TListaRec<TÁrbol<TElemento>>)

EsVacío: (TÁrbol<TElemento>) → (Lógico)

sea

E ∈ TElemento

LH ∈ TListaRec<TÁrbol<TElemento>>

A ∈ TÁrbol<TElemento>

definitud

def Raíz(A) si no EsVacío(A)

def Hijos(A) si no EsVacío(A)

axiomas

Raíz(Crear(E,LH)) = E

Hijos(Crear(E,LH)) = LH

EsVacío(Vacío) = Verdadero

EsVacío(Crear(E, LH)) = Falso

fin especificación

Definiciones

especificación Árbol N-ario **es**

usa Estructuras lineales recursivas

parámetros tipo TElemento

positivo N

define TÁrbol<TElemento,N>

operaciones

{constructoras generadoras}

Vacío: () → (TÁrbol<TElemento,N>)

Crear: (TElemento, TListaRec<TÁrbol<TElemento,N>>) →(TÁrbol<TElemento,N>)

{observadoras}

Raíz : (TÁrbol<TElemento,N>) → (TElemento)

Hijos: (TÁrbol<TElemento,N>) → (TListaRec<TÁrbol<TElemento,N>>)

EsVacío: (TÁrbol<TElemento,N>) → (Lógico)

sea

E ∈ TElemento

LH ∈ TListaRec<TÁrbol<TElemento,N>>

A ∈ TÁrbol<TElemento,N>

definitud

def Crear(E,LH) si Tamaño(LH) ≤ N

def Raíz(A) si no EsVacío(A)

def Hijos(A) si no EsVacío(A)

axiomas

Raíz(Crear(E,LH)) = E

Hijos(Crear(E,LH)) = LH

EsVacío(Vacío) = Verdadero

EsVacío(Crear(E,LH)) = Falso

fin especificación

Definiciones

- Cuando en la definición de árbol se dice que "...los restantes nodos se distribuyen en una serie de subconjuntos disjuntos...", significa que todo árbol tiene un *orden implícito*:
  - Viene dado por el orden lineal de los nodos en cada nivel.
  - La ordenación del árbol se pone de manifiesto en que los *hijos*:
    - Se constituyen en una lista lineal.
    - No simplemente como un conjunto.

Definiciones

- Cuando se piensa en los hijos como una lista de árboles, se nota que en una lista de árboles puede haber:
  - Árboles vacíos y no vacíos.
- Se define *árbol compacto* como aquel en el que no se permite la existencia de subárboles vacíos que sean hermanos por la izquierda de subárboles no vacíos.
  - En un árbol compacto los subárboles no vacíos, hijos de un mismo padre, están agrupados al principio de la lista.

Árboles

Recorrridos

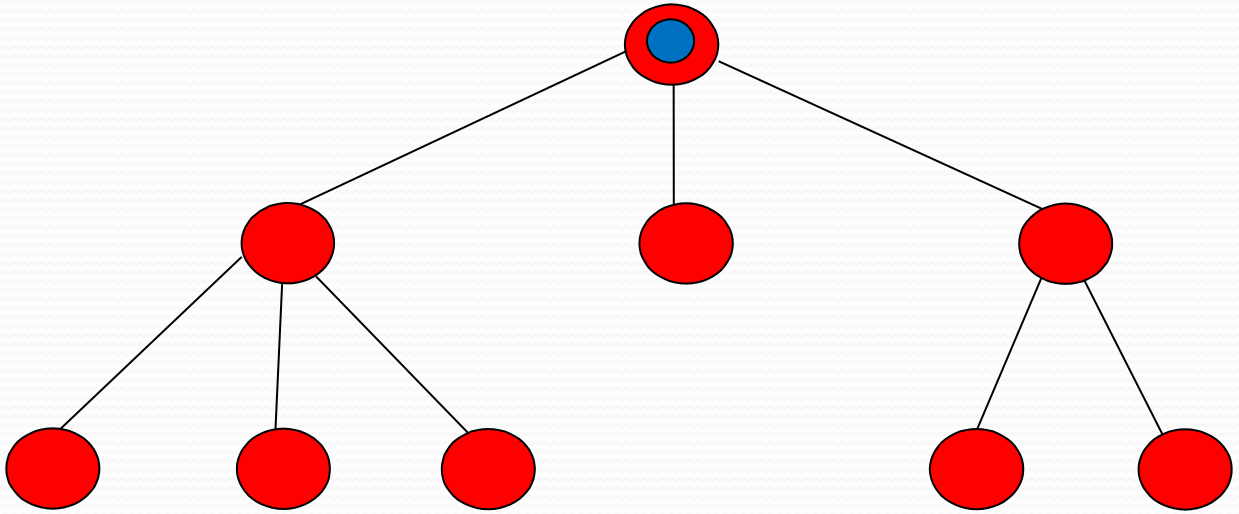
Inicio

Recorridos

- El acceso sistemático a cada uno de los miembros de una estructura resulta muy útil en cualquier estructura de datos.
- Un recorrido de una estructura lineal se hace según su ordenación natural de primero a último, según la relación de sucesión.
- En un recorrido de un árbol
  - Ya que la relación a seguir es padre/hijo.
  - Y como un padre puede tener varios hijos.
  - Por tanto, se multiplican las posibles formas de recorrido.
- Hay dos formas básicas de recorrer un árbol:
  - En profundidad
  - En amplitud.
    - El recorrido en profundidad admite diferentes variaciones.

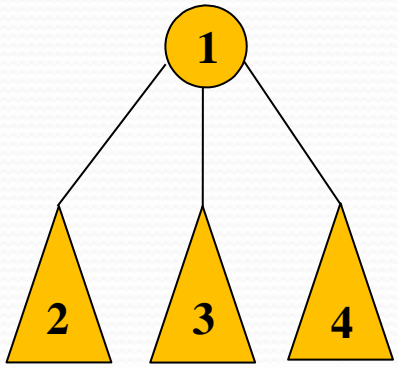
Recorrido en profundidad

- El recorrido siempre empieza por la raíz.
- Se trata de alejarse de la raíz hasta alcanzar un nodo hoja.
- Una vez alcanzado, se da un paso atrás para intentar alejarse por un camino alternativo.

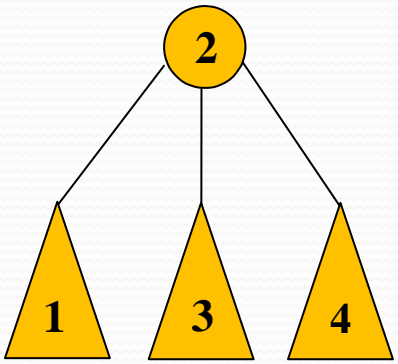


Recorrido en profundidad

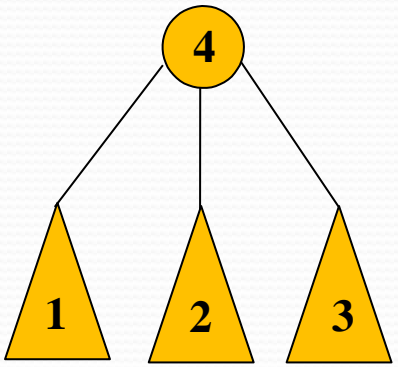
- Este esquema implica que por un nodo se pasa varias veces:
  - Cuando se accede por primera vez.
  - Cuando se regresa de cada uno de sus hijos para acceder al siguiente o volver al padre.
- El momento en que se trata el nodo da lugar a tres variantes:
  - Recorrido en preorden: en el primer acceso.
  - Recorrido en inorden o en orden simétrico: después de tratar al primer hijo.
  - Recorrido en postorden: después de tratar a los hijos.



Preorden



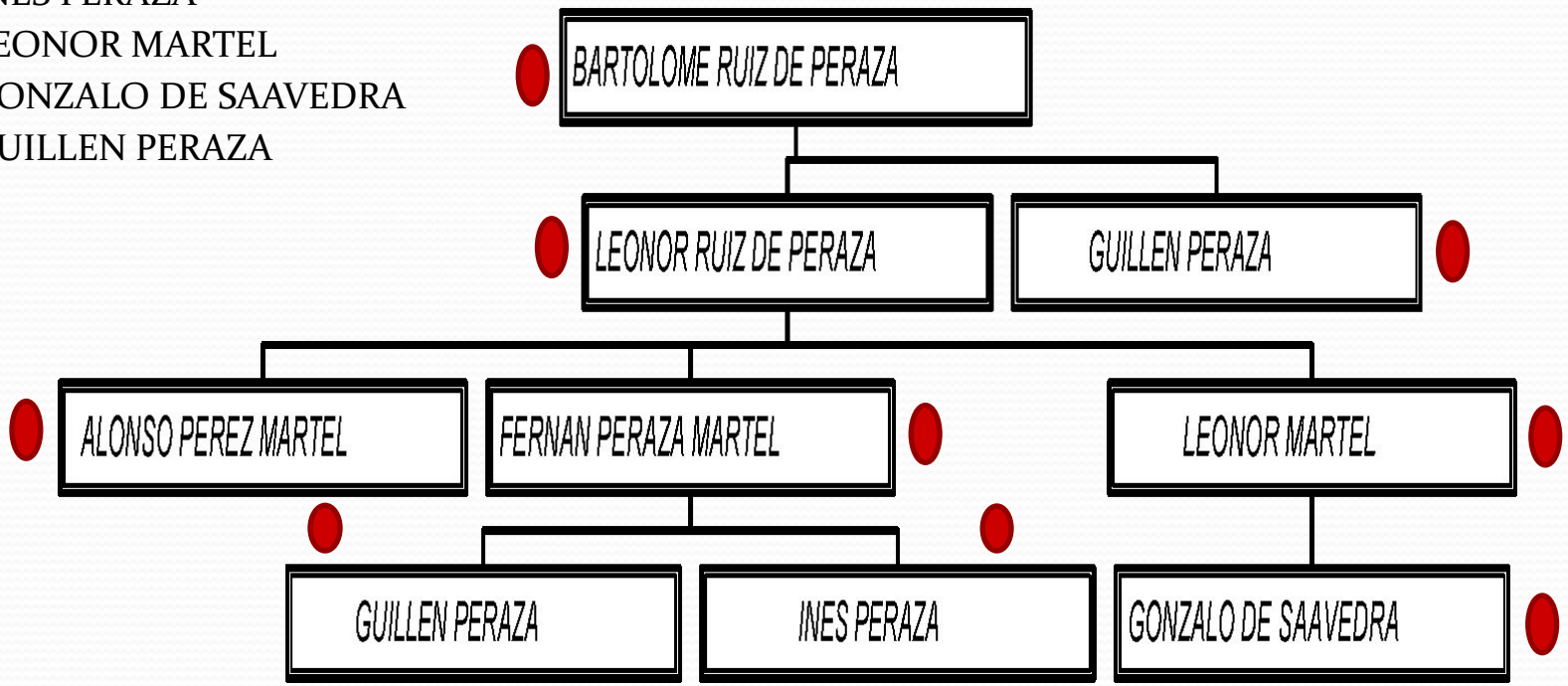
Inorden



Postorden

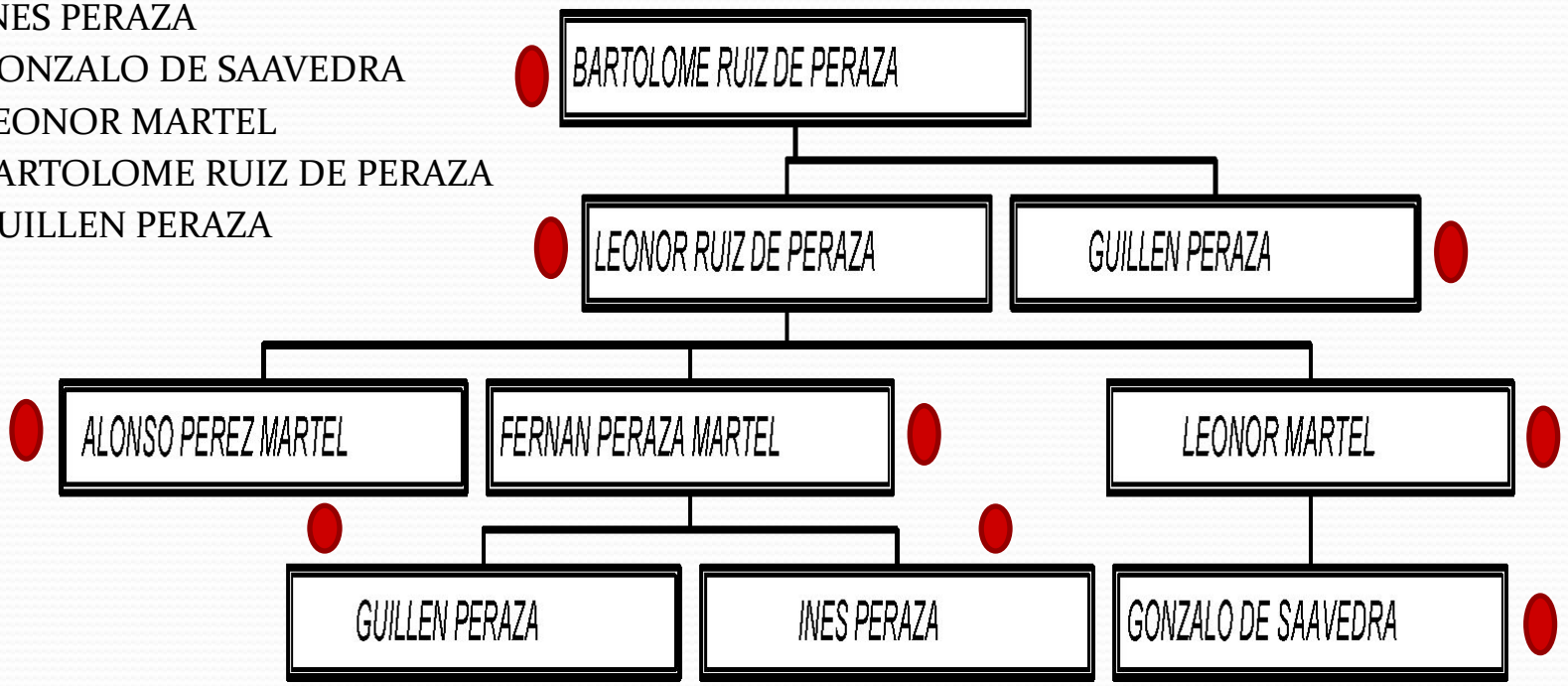
Recorrido en preorden

- Se trata primero la raíz del árbol y, a continuación, se recorren en preorden, sus subárboles de izquierda a derecha:
  - BARTOLOME RUIZ DE PERAZA
  - LEONOR RUIZ PERAZA
  - ALONSO PEREZ MARTEL
  - FERNAN PERAZA MARTEL
  - GUILLEN PERAZA
  - INES PERAZA
  - LEONOR MARTEL
  - GONZALO DE SAAVEDRA
  - GUILLEN PERAZA



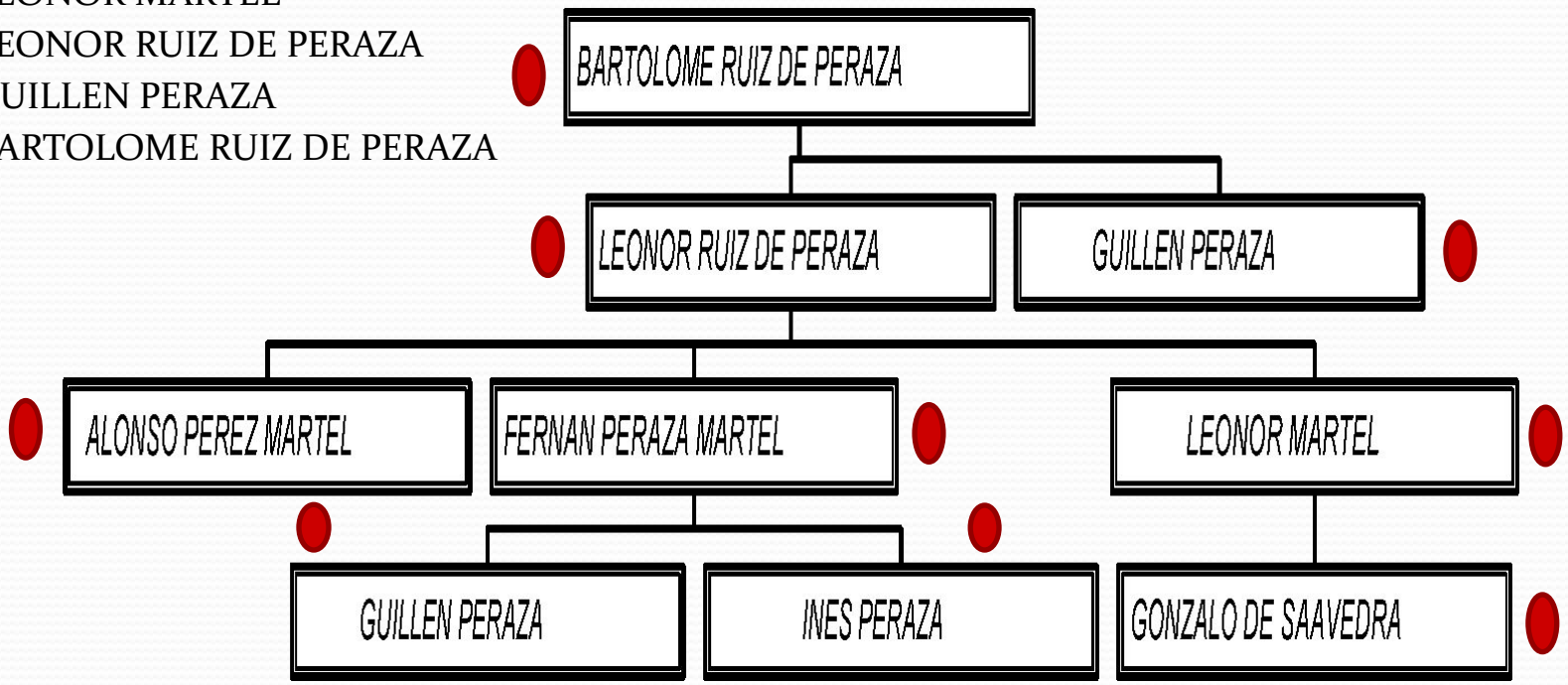
Recorrido en inorden o orden simétrico

- Se recorre en inorden el primer subárbol, luego se trata la raíz y, a continuación, se recorren en inorden el resto de los subárboles, de izquierda a derecha:
  - ALONSO PEREZ MARTEL
  - LEONOR RUIZ PERAZA
  - GUILLEN PERAZA
  - FERNAN PERAZA MARTEL
  - INES PERAZA
  - GONZALO DE SAAVEDRA
  - LEONOR MARTEL
  - BARTOLOME RUIZ DE PERAZA
  - GUILLEN PERAZA

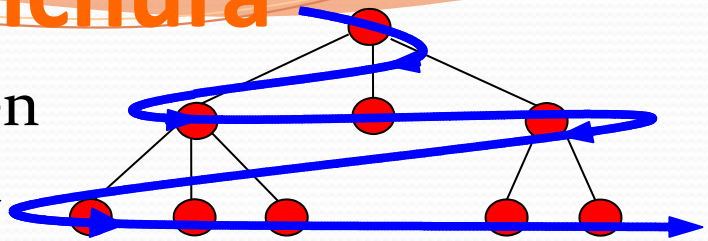


Recorrido en postorden

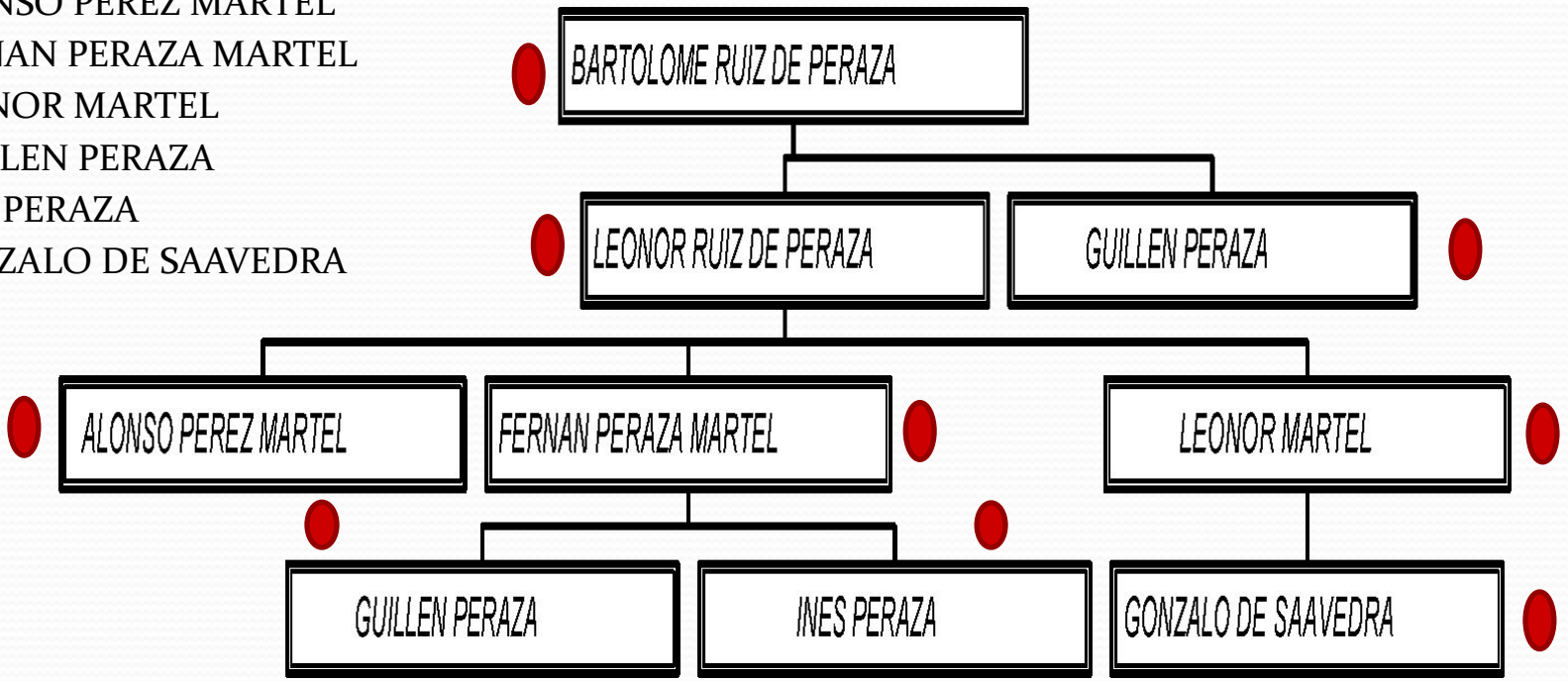
- *Postorden*: Se recorren primero en postorden todos los subárboles, de izquierda a derecha, y finalmente se trata la raíz:
  - ALONSO PEREZ MARTEL
  - GUILLEN PERAZA
  - INES PERAZA
  - FERNAN PERAZA MARTEL
  - GONZALO DE SAAVEDRA
  - LEONOR MARTEL
  - LEONOR RUIZ DE PERAZA
  - GUILLEN PERAZA
  - BARTOLOME RUIZ DE PERAZA



Recorrido en anchura



- Esta forma de recorrido se conoce también como recorrido por niveles o en amplitud
- Se explora el árbol nivel a nivel, de izquierda a derecha y del primero al último:
  - BARTOLOME RUIZ DE PERAZA
  - LEONOR RUIZ DE PERAZA
  - GUILLEN PERAZA
  - ALONSO PEREZ MARTEL
  - FERNAN PERAZA MARTEL
  - LEONOR MARTEL
  - GUILLEN PERAZA
  - INES PERAZA
  - GONZALO DE SAAVEDRA



Árboles

Árboles binarios

Inicio

Definición

- Si se restringe a dos el grado máximo de un nodo —el número de ramas que parten de dicho nodo—, surge:
  - el árbol 2-ario conocido como árbol binario.
- En cada nodo de los árboles binarios se distingue entre su subárbol izquierdo y su subárbol derecho.
- En definitiva, un árbol binario es un conjunto finito de *m* nodos ($m \geq 0$), tal que:
  - Si $m = 0$, el árbol está vacío.
  - Si $m > 0$:
    - Existe un nodo raíz.
    - El resto de los nodos se reparten entre dos árboles binarios:
      - Se conocen como subárbol izquierdo y subárbol derecho de la raíz.

Operaciones en un árbol binario

- *Vacío()*: Proporciona un árbol vacío.
- *Crear(E, Árbol$_I$, Árbol$_D$)*: Proporciona un árbol no vacío con:
  - *E* en su raíz.
  - *Árbol$_I$* como subárbol izquierdo.
  - *Árbol$_D$* como subárbol derecho.

- *Izquierdo(Árbol)*: Devuelve el subárbol izquierdo de *Árbol* (no vacío).
- *Derecho(Árbol)*: Devuelve el subárbol derecho de *Árbol* (no vacío).

- *Raíz(Árbol)*: Devuelve la raíz del *Árbol* (no vacío).
- *EsVacío(Árbol)*: Devuelve verdadero si *Árbol* está vacío.

Especificación

especificación Árboles binarios **es**

parámetro tipo TElemento

define TÁrbolBinario<TElemento>

operaciones

Vacío: () → (TÁrbolBinario<TElemento>)

Crear: (TElemento,
TÁrbolBinario<TElemento>,
TÁrbolBinario<TElemento>) → (TÁrbolBinario<TElemento>)

EsVacío: (TÁrbolBinario<TElemento>) → (Lógico)

Raíz: (TÁrbolBinario<TElemento>) → (TElemento)

Izquierdo: (TÁrbolBinario<TElemento>) → (TÁrbolBinario<TElemento>)

Derecho: (TÁrbolBinario<TElemento>) → (TÁrbolBinario<TElemento>)

sea

E ∈ TElemento; TI, TD ∈ TÁrbolBinario<TElemento>

A ∈ TÁrbolBinario<TElemento>

definitud

def Raíz(A) si no EsVacío(A)

def Izquierdo(A) si no EsVacío(A)

def Derecho(A) si no EsVacío(A)

axiomas

EsVacío(Vacío) = Verdadero

EsVacío(Crear(E,TI,TD)) = Falso

Raíz(Crear(E,TI,TD)) = E

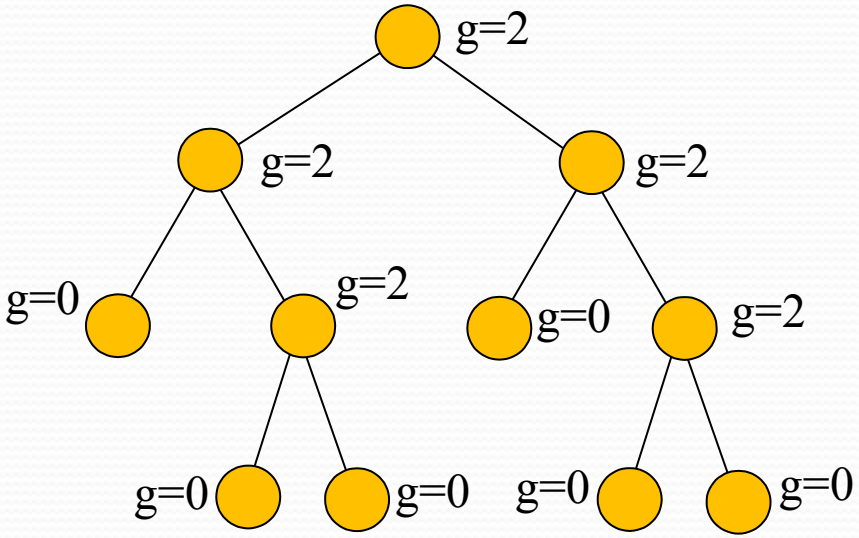
Izquierdo(Crear(E,TI,TD)) = TI

Derecho(Crear(E,TI,TD)) = TD

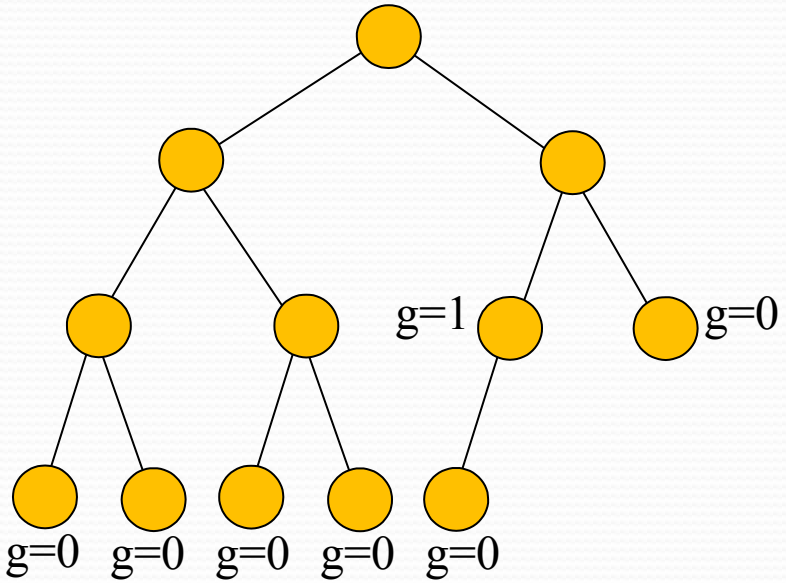
fin especificación

Definiciones

- Un *árbol binario lleno* es un árbol binario en el cual cada nodo es de grado cero o dos.
- Un *árbol binario completo* es aquel en el cual todos los nodos de grado cero o uno están en los dos últimos niveles, de forma que las hojas del último nivel ocupan las posiciones más a la izquierda de dicho nivel.



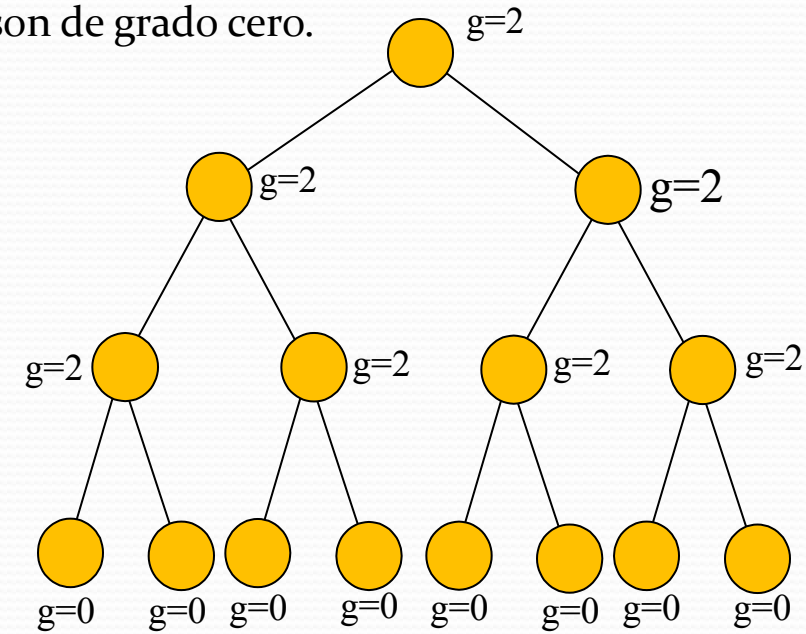
Árbol binario lleno



Árbol binario completo

Definiciones

- Un *árbol binario perfecto* es uno que tiene todos los nodos que puede tener en función de su altura.
  - En cada nivel de un árbol binario el número máximo de nodos se multiplica por dos respecto al del nivel anterior
    - Uno en el primer nivel
    - Con *k* niveles puede tener como máximo 2^k-1 nodos.
      - Todos los nodos son de grado dos.
      - Excepto los del último nivel que son de grado cero.



Árbol binario perfecto

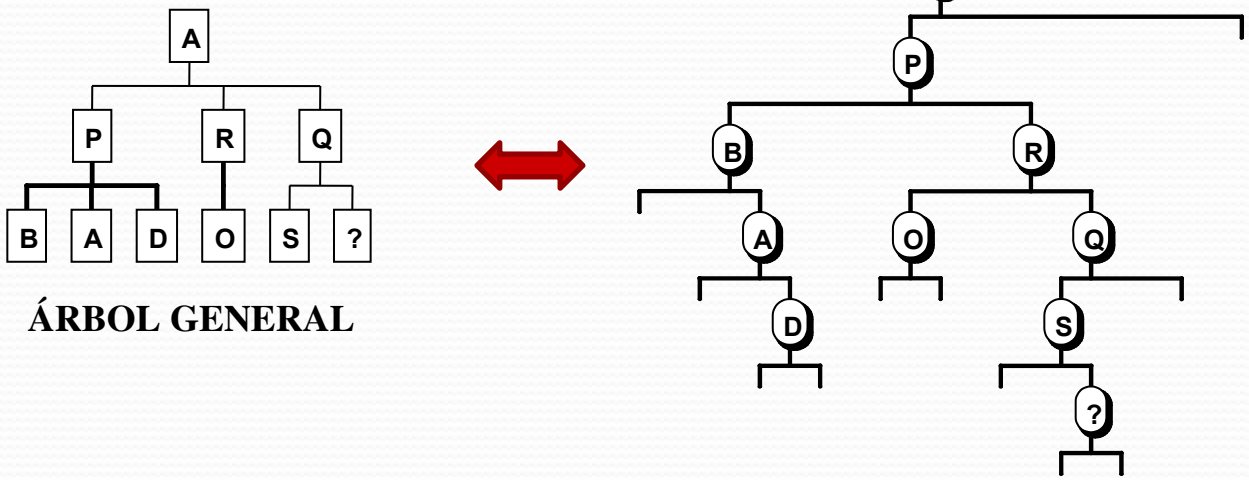
Árboles

Aplicaciones de los árboles binarios

Inicio

Aplicaciones de los árboles binarios

- La representación de otros tipos de árboles.
  - Es posible establecer una relación de equivalencia entre cualquier árbol no binario y un árbol binario.
    - Se trata de su *árbol binario equivalente*.
  - A cada nodo del árbol no binario le corresponde un nodo del binario y viceversa.
  - El hijo izquierdo de un nodo en el árbol binario será el correspondiente al primer hijo del nodo equivalente del árbol no binario.
  - El hijo derecho de un nodo en el árbol binario será el correspondiente al primer hermano por la derecha del nodo equivalente del árbol no binario.



ÁRBOL GENERAL

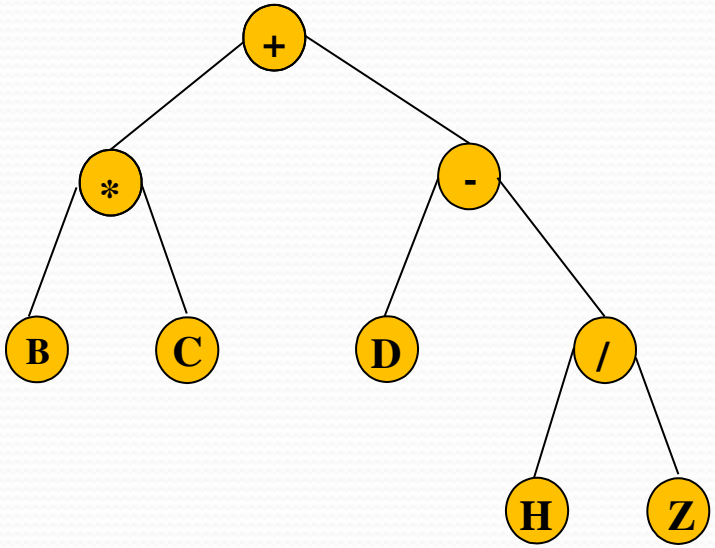
ÁRBOL BINARIO EQUIVALENTE

Aplicaciones de los árboles binarios

- Este esquema de equivalencia puede ser extendido fácilmente para encontrar la forma binaria equivalente de un bosque.
  - Basta con considerar a las raíces de los distintos árboles como hermanas.
- Una vez establecida la equivalencia, suele resultar más fácil implementar un árbol binario.
  - Sólo tienen dos hijos.

Aplicaciones de los árboles binarios

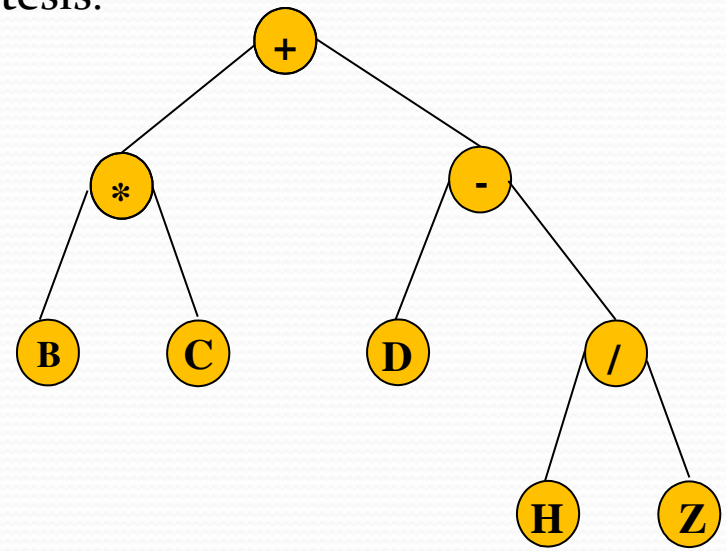
- Otro campo de aplicación de los árboles binarios se halla en aplicaciones que dan solución a problemas cuya estructura es esencialmente binaria.
  - Las expresiones aritméticas y lógicas.
    - O son átomos: operadores unarios.
      - Un valor sólo o expresión acompañado de un operador unario.
    - O están formadas por dos operandos: operadores binarios.
      - Dos valores o expresiones conectados mediante un operador binario.
    - Implica que cualquier expresión puede representarse por su árbol implícito en el que los nodos internos serían operadores y los nodos hoja operandos.
      - Los operadores unarios quedan representados por un solo subárbol.



(B * C) + (D – H / Z)

Aplicaciones de los árboles binarios

- El que cualquier expresión pueda representarse por su árbol implícito conduce al resultado de que se puedan utilizar estructuras arbóreas para llevar a cabo la manipulación de expresiones en un ordenador.
  - Como ventaja adicional, en el árbol las precedencias entre operadores quedan recogidas en la estructura, por lo que no es necesario representar paréntesis.



(B * C) + (D – H / Z)

- Otra gran área de aplicación de los árboles binarios es como estructura de representación para contenedores de claves ordenadas.
  - Este aspecto se estudiará en detalle más adelante.

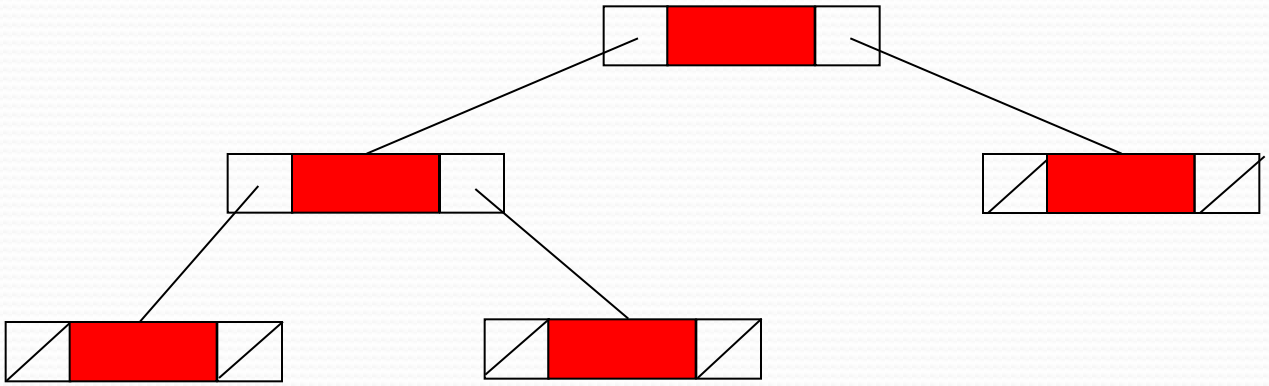
Árboles

Implementación de árboles binarios

Inicio

Representación encadenada

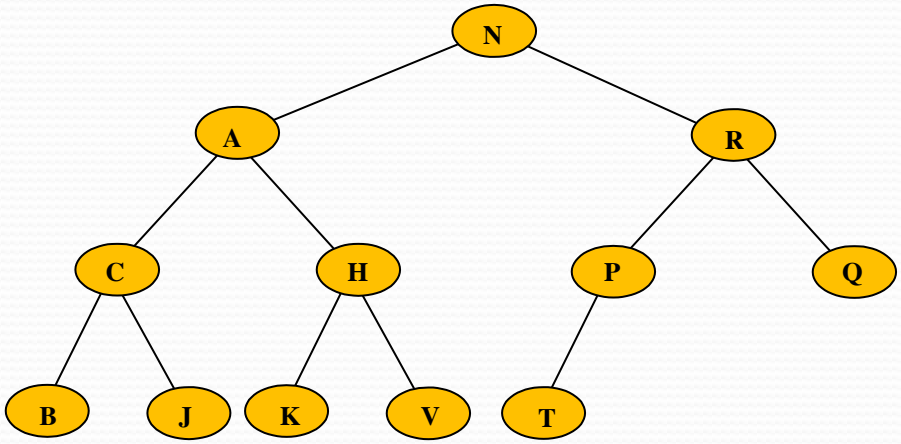
- La implementación más común para los árboles binarios es una estructura encadenada en memoria dinámica.
  - Cada nodo almacena:
    - La información propia.
    - Dos enlaces que referencian a sus hijos: izquierdo y derecho
      - Un enlace nulo indica un hijo vacío.

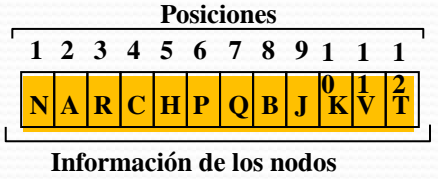


También es posible representar un árbol en un espacio secuencial; en ocasiones puede ser lo más adecuado.

Representación contigua para árboles binarios completos

- Un espacio secuencial homogéneo y aleatoriamente direccionable (un vector, un fichero, una lista directa,...) en el que:
  - La raíz ocupa la primera posición.
  - Los hijos izquierdo y derecho del nodo que ocupe la posición *i*, tienen reservadas las posiciones *2*i* y *2*i+1*.

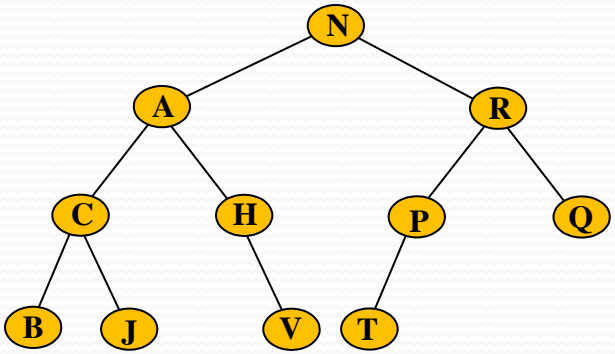




Los árboles binarios no completos también pueden representarse de esta forma, pero podría desperdiciarse una cantidad significativa de espacio de almacenamiento, ya que habría que respetar las posiciones correspondientes a nodos no existentes

Enumeraciones secuenciales

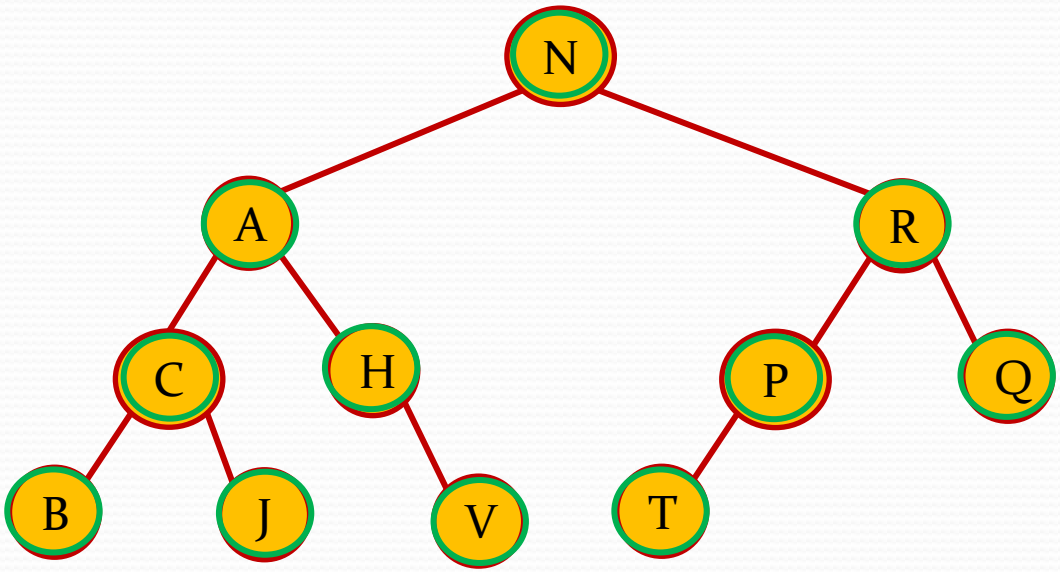
- Consiste en enumerar los elementos del árbol según alguno de los posibles órdenes de recorrido
  - Para poder conocer la estructura exacta del árbol debe incluirse información adicional sobre la misma.
  - Como ejemplo de esta técnica se presenta una opción de representación secuencial en preorden de árboles binarios.
    - Para cada elemento, el rango representa la posición donde comienza en la representación secuencial el subárbol que sigue en preorden al recorrido del subárbol izquierdo del elemento.
      - El subárbol izquierdo de un elemento se representa desde la posición siguiente a la del elemento hasta la posición anterior a su rango.



	Posiciones										
	1	2	3	4	5	6	7	8	9	10	11
Info de los nodos	N	A	C	B	J	H	V	R	P	T	Q
"Rango" de los nodos	8	6	5	5	6	7	8	11	11	11	12

Se pueden hacer representaciones en otros órdenes, e incluso algunas de ellas pueden ser válidas para árboles generales.

Enumeración secuencial en preorden con rango



Posiciones	1	2	3	4	5	6	7	8	9	10	11
Info de los nodos											
Rango de los nodos											

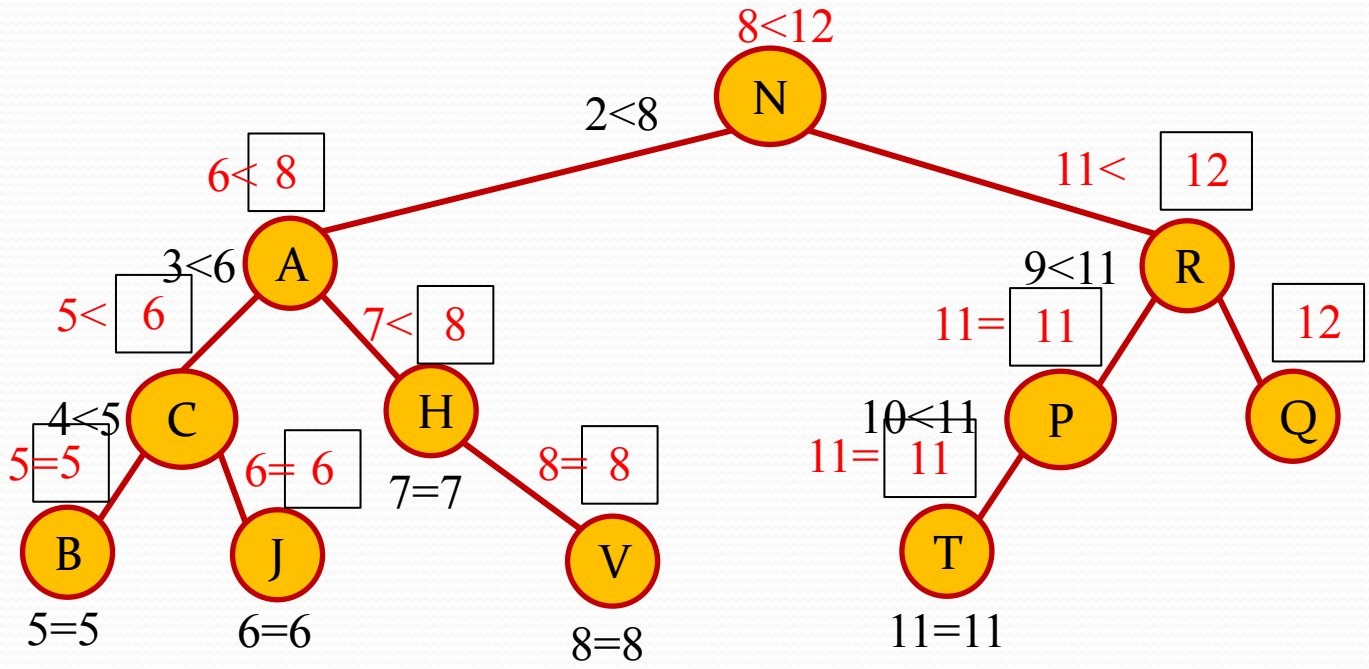
Algoritmo para obtener la enumeración secuencial en preorden con rango de un árbol binario

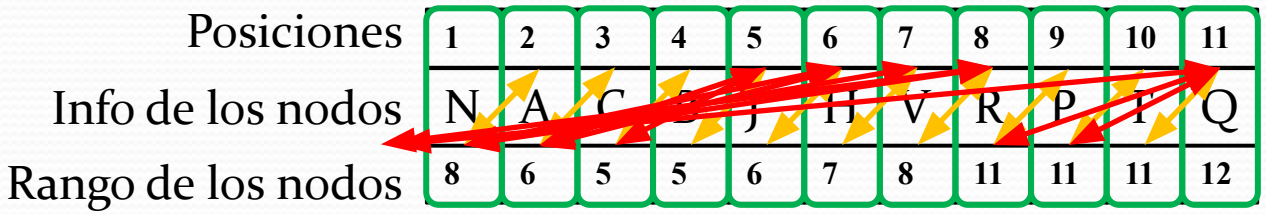
Vector de parejas (info, rango) con la información de la enumeración secuencial en preorden

```
void construyeEnumeración(ÁrbolBinario árbol, ParejasInfoRango enumer[], int [] pos) {
   int posActual = pos[0];
   if ( !árbol.esVacío() ){
      enumer[pos[0]++].info = árbol.raíz();
      construyeEnumeración(árbol.izquierdo(), enumer, pos);
      enumer[posActual].rango = pos[0];
      construyeEnumeración(árbol.derecho(), enumer, pos);
   }
}
```

Posición del vector enumer donde se inserta el próximo elemento

Enumeración secuencial en preorden con rango: Construcción del árbol





Algoritmo para la reconstrucción de un árbol binario a partir de la enumeración secuencial en preorden con rango

Posición en la enumeración a partir de la cual no hay elementos del árbol que se construye

Vector de parejas (info, rango) con la información de la enumeración secuencial en preorden

¿Hay elementos en el subárbol izquierdo?

¿Quedan elementos por tratar?

¿Hay elementos en el subárbol derecho?

```
ÁrbolBinario reconstruyeDeEnumeración(ParejasInfoRango enumer[], int [] pos, int rango) {
   ÁrbolBinario izq = new ÁrbolBinario();
   ÁrbolBinario der = new ÁrbolBinario();
   Object raíz = enumer[pos[0]].info;
   if ( pos[0] < enumer.length ){
      if ( pos[0]+1 < enumer[pos].rango ){
         pos[0]++;
         izq = reconstruyeDeEnumeración(enumer, pos, enumer[pos[0]-1].rango);
      }
      if ( pos[0]+1 < rango ){
         pos[0]++;
         der = reconstruyeDeEnumeración(enumer, pos, rango);
      }
      return new ÁrbolBinario(raíz, izq, der);
   }
}
```

Realización de un árbol binario en Java

```
public class ÁrbolBinario {

   class Nodo {
      Object info;
      ÁrbolBinario hijos[] = new ÁrbolBinario[2];
   }

   private Nodo árbol;

   public ÁrbolBinario() {
      árbol = null;
   }

   public ÁrbolBinario(Object info, ÁrbolBinario izqdo, ÁrbolBinario drcho) {
      árbol = new Nodo();
      árbol.info =  info;
      árbol.hijos[0] = izqdo;
      árbol.hijos[1] = drcho;
   }
```

Componente básico del árbol

Atributo que representa la raíz del árbol

Construye un árbol vacío

Construye un árbol con info en la raíz e izqdo y drcho como subárboles

- Las operaciones *Vacío* y *Crear* del árbol binario se convierten en los constructores respectivos.

Realización de un árbol binario en Java

```
public void modificar(Object info, ÁrbolBinario izqdo, ÁrbolBinario drcho) {
   árbol.info =  info;
   árbol.hijos[0] = izqdo;
   árbol.hijos[1] = drcho;
}

public Object raíz() {
   return árbol.info;
}

public ÁrbolBinario izquierdo() {
   return árbol.hijos[0];
}

public ÁrbolBinario derecho() {
   return árbol.hijos[1];
}

public boolean esVacío() {
   return árbol == null;
}
}
```

Modifica un árbol poniendo info como raíz e izqdo y drcho como subárboles

Devuelve la información de la raíz

Si el árbol está vacío, se produce un NullPointerException

Devuelve el subárbol izquierdo

Devuelve el subárbol derecho

Devuelve verdadero si árbol está vacío

- El método *modificar* altera los valores almacenados en un nodo ya creado.
  - Se podría prescindir de *modificar* a costa de un uso más intensivo de la memoria.

Árboles

Árbol binario de búsqueda

Inicio

Búsqueda binaria o dicotómica

- Si se tiene un conjunto de datos y se quiere buscar un elemento, habrá que empezar a compararlo con los del conjunto.
  - Hasta encontrar uno que sea igual o
  - Poder determinar que no se encuentra.
    - Si los elementos no están ordenados, para determinar que un elemento no se encuentra habrá que compararlo con todos los miembros del conjunto.
    - Si los elementos están almacenados en orden, se puede buscar linealmente desde el primero y se sabe que el buscado no se encuentra cuando figura uno mayor que él —o menor, si el orden es descendente.
    - Si los elementos se hallan ordenados en un espacio de almacenamiento contiguo y direccionable aleatoriamente —como un vector— una opción mejor es la búsqueda binaria o dicotómica.

Búsqueda binaria o dicotómica

Si se tiene una secuencia contigua de claves ordenadas con acceso directo, se puede localizar una clave mediante una búsqueda dicotómica.

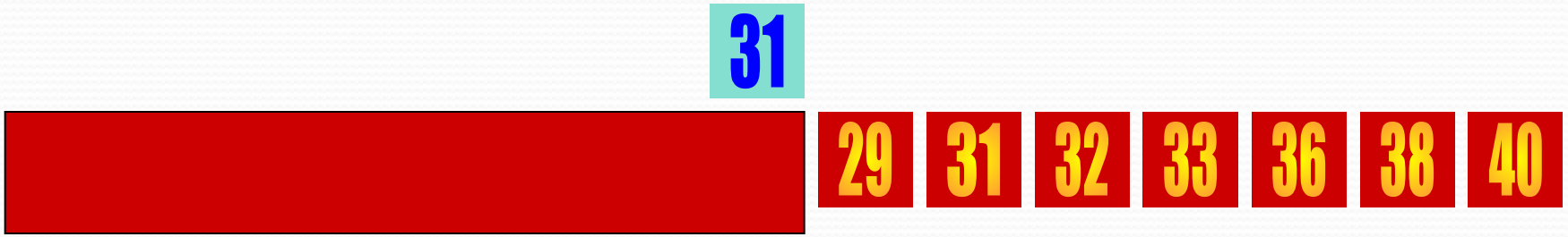
Búsqueda binaria o dicotómica

Clave a buscar

31

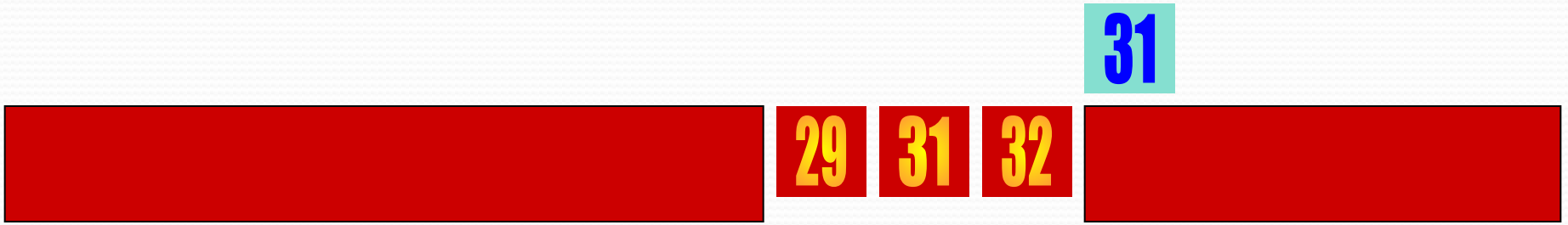
- Comienza por comparar el valor que se busca con el elemento que está en el centro del espacio de almacenamiento.
- Si el elemento buscado es mayor que el central quedan descartados todos los anteriores.

Búsqueda binaria o dicotómica



- La búsqueda continúa comparando con el elemento central del subespacio no descartado.
- Si el elemento buscado es menor que el central quedan descartados todos los siguientes.

Búsqueda binaria o dicotómica



- La búsqueda continúa comparando con el elemento central del subespacio no descartado hasta localizar el elemento o determinar que no se encuentra.

Búsqueda binaria

- La exigencia de la búsqueda binaria de que los elementos deban estar ordenados complica la inserción y la extracción de los elementos en el espacio de búsqueda.
  - Una inserción requiere el desplazamiento de todos los que sean mayores que el insertado para abrir hueco.
  - Una extracción requiere que los elementos mayores que el extraído se desplacen para cubrir el hueco que aquél deja.

Búsqueda binaria o logarítmica

- Como en cada iteración el espacio de búsqueda se reduce a la mitad, se puede encontrar un elemento o determinar que no está con un número de comparaciones del orden de $log_2 N$, donde N es el número de elementos del conjunto.
  - Por tal razón este algoritmo también se conoce como **búsqueda logarítmica**.
- Una búsqueda lineal necesita:
  - Del orden de N comparaciones en el caso peor, o
  - $N/2$ en el caso promedio, si los elementos son equiprobables como candidatos de búsqueda.

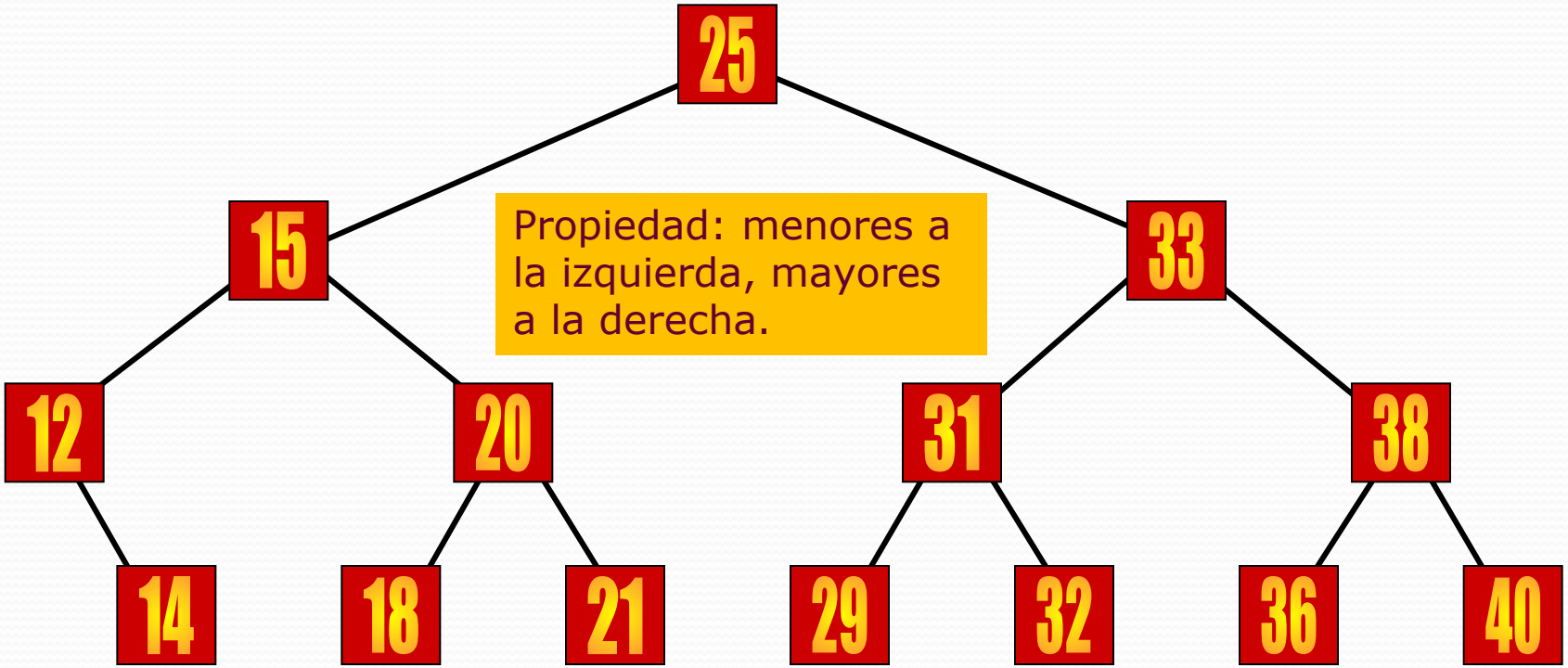
Árbol binario de búsqueda

12	14	15	18	20	21	25	29	31	32	33	36	38	40

Las claves se pueden jerarquizar en función de su probabilidad de ser accedidas durante una búsqueda.

- El elemento central tiene una probabilidad de comparación del 100%
- Los centrales de los subespacios en que éste divide el espacio de búsqueda la tienen del 50%, y así sucesivamente

Árbol binario de búsqueda



Esta jerarquización da lugar al árbol de la búsqueda binaria o *árbol binario de búsqueda*.

Árbol binario de búsqueda

- Formalmente, consiste en un árbol binario en el que, si no está vacío:
  - El subárbol izquierdo es un árbol binario de búsqueda que sólo contiene elementos menores que la raíz.
  - El subárbol derecho es un árbol binario de búsqueda que sólo contiene elementos mayores que la raíz.
- ✔ Si un árbol binario de búsqueda se implementa con una estructura encadenada, se pueden tener las ventajas de la búsqueda dicotómica sin los inconvenientes de la ubicación contigua a la hora de insertar o extraer.

Árboles

El árbol binario de búsqueda como un contenedor

Inicio

Operaciones del árbol binario de búsqueda

- *Insertar(Árbol, Elemento)*: Devuelve el árbol resultante de insertar *Elemento* en *Árbol*.
- *Extraer(Árbol, Elemento)*: Devuelve el árbol resultante de extraer *Elemento* de *Árbol*.
- *Buscar(Árbol, Elemento)*: Devuelve el subárbol de *Árbol* cuya raíz contiene a *Elemento*.

Especificación del árbol binario de búsqueda

Una extensión de la especificación expuesta para los árboles binarios.

especificación Árboles binarios de búsqueda **es**

usa Árboles binarios

parámetro tipo ordenable TElemento

define

TÁrbolBinBus<TElemento> es un TÁrbolBinario<TElemento>

operaciones

Insertar: (TÁrbolBinBus<TElemento>,TElemento) → (TÁrbolBinBus<TElemento>)

Extraer : (TÁrbolBinBus<TElemento>,TElemento) → (TÁrbolBinBus<TElemento>)

Buscar : (TÁrbolBinBus<TElemento>,TElemento) → (TÁrbolBinBus<TElemento>)

sea

E, E1, E2 ∈ TElemento

T, TI, TD ∈ TÁrbolBinBus<TElemento>

E ∈ T ⇔ (E = Raíz(T)) ∨ (E ∈ Izquierdo(T)) ∨ (E ∈ Derecho(T))

axiomas

Crear(E,TI,TD) = ERROR
si ($\exists$ E1 $\in$ TI / E1 $\geq$ E) $\vee$ ($\exists$ E1 $\in$ TD / E1 $\leq$ E)

Insertar(Vacío,E) = Crear(E,Vacío,Vacío)
Insertar(Crear(E1,TI,TD),E) =
Crear(E1,Insertar(TI,E),TD) si E < E1
Crear(E1,TI,Insertar(TD,E)) si E > E1
Crear(E1,TI,TD) si E = E1

Extraer(Vacío,E) = Vacío
Extraer(Crear(E1,TI,TD),E) =
Crear(E1,Extraer(TI,E),TD) si E < E1
Crear(E1,TI,Extraer(TD,E)) si E > E1
TD si E = E1 $\wedge$ EsVacío(TI)
TI si E = E1 $\wedge$ EsVacío(TD)
Crear(E2,TI,Extraer(TD,E2)),
donde E2 $\in$ TD $\wedge$ $\forall$ E3 $\in$ TD / E3 $\neq$ E2: E2 < E3
si (E = E1) $\wedge$
no EsVacío(TI) $\wedge$
no EsVacío(TD)

Buscar(Vacío,E) = Vacío
Buscar(Crear(E1,TI,TD),E) =
Crear(E1,TI,TD) si E = E1
Buscar(TI,E) si E < E1
Buscar(TD,E) si E > E1

fin especificación

Del árbol binario de búsqueda

- Es un árbol binario con la restricción de "menores a la izquierda, mayores a la derecha".
- Se añaden tres operaciones —*Insertar*, *Extraer* y *Buscar*— que reflejan el carácter del árbol binario de búsqueda como estructura de representación de contenedores asociativos de **claves ordenables**.
- *Buscar* permite saber si un elemento está, o no, en el árbol.
  - Devuelve el subárbol del que es raíz el elemento buscado.
  - Vacío si no está.
  - Facilita la realización de ulteriores operaciones.

Operaciones del árbol binario de búsqueda

- Tanto *Buscar* como *Insertar* y *Extraer* siguen un esquema que responde a un "algoritmo básico de búsqueda binaria", donde la decisión está entre:
  - Actuar en la raíz o
  - Continuar la operación en uno de los subárboles.

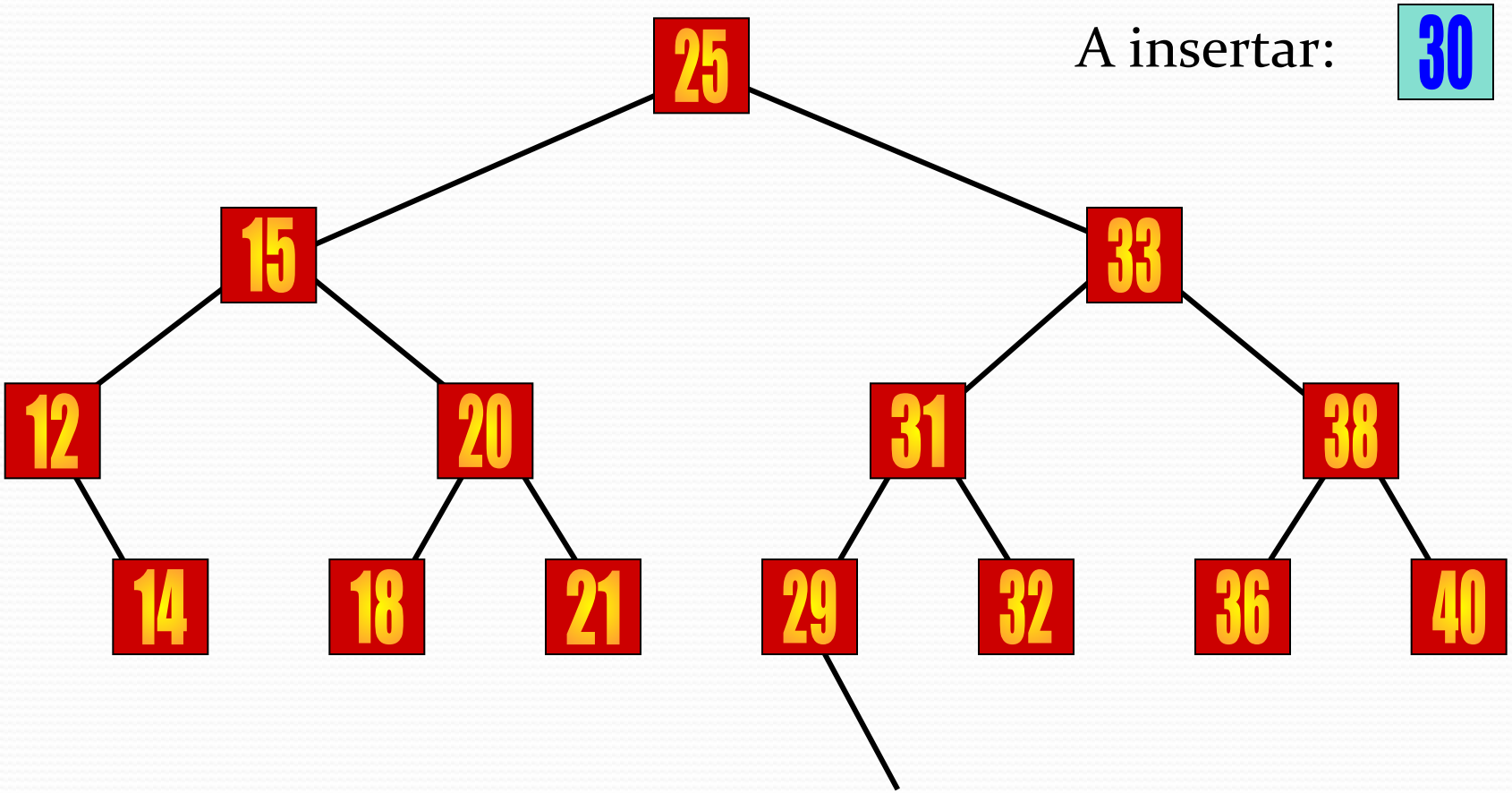
```
sea E el elemento a buscar
si el árbol está vacío entonces
   E no se encuentra en el árbol, se actúa en consecuencia
si no
   si E coincide con la raíz del árbol entonces
      se encontró E, se actúa en consecuencia
   si no
      si E es mayor que la raíz del árbol entonces
         buscar E en el subárbol derecho
      si no
         buscar E en el subárbol izquierdo
      fin si
   fin si
fin si
```

Se denomina camino de búsqueda al que se sigue, según este algoritmo, cuando se intenta localizar un valor determinado.

Operaciones del árbol binario de búsqueda

- *Extraer* se lleva a cabo sólo cuando el elemento se encuentra en el árbol.
- *Insertar* sólo se completa cuando el elemento no se encuentra en el árbol.
  - Recursivamente se alcanza un árbol vacío, se sustituye (en su padre) por un árbol cuya raíz contiene la nueva información.
    - Se debe a que se ha elegido una variante común del árbol binario de búsqueda que no admite elementos repetidos.
    - Se puede permitir la inserción de repetidos modificando la regla básica de distribución: “menores o iguales a la izquierda, mayores a la derecha”.
  - El esquema recursivo especificado implica que el nuevo elemento siempre se inserta como hoja.

Árbol binario de búsqueda: Inserción.



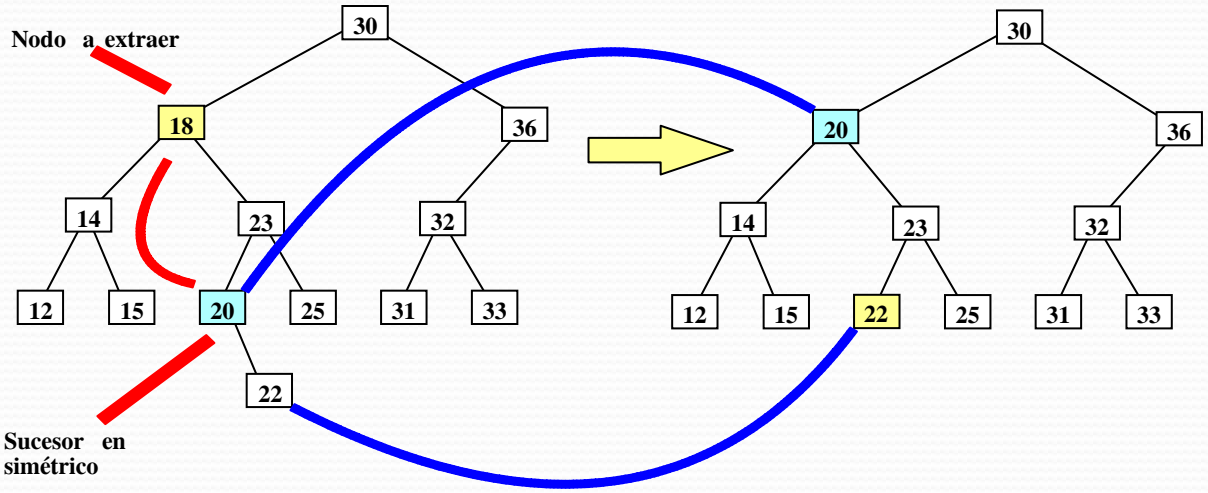
La nueva clave siempre se inserta en un nodo hoja

Operaciones del árbol binario de búsqueda

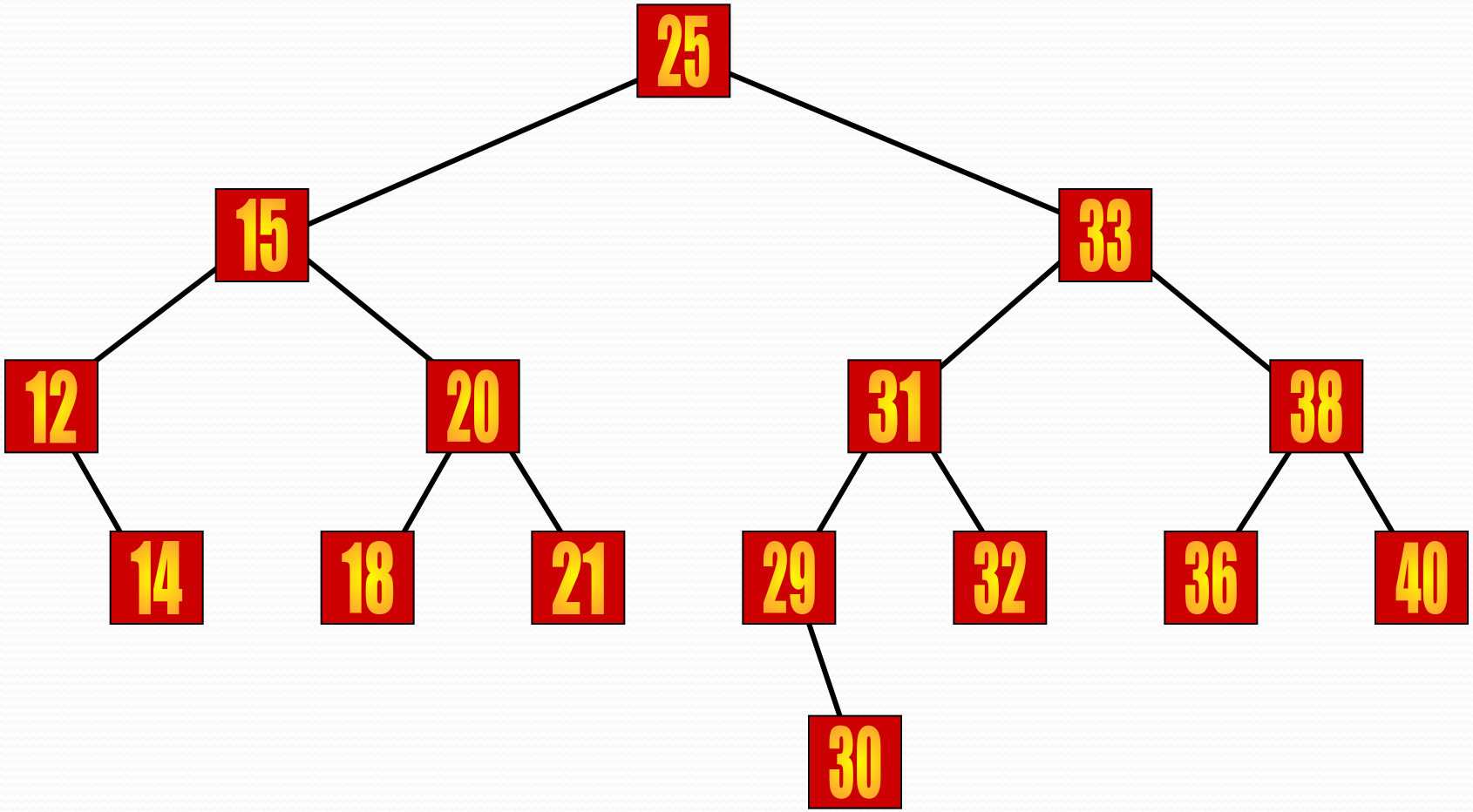
- *Extraer* comienza por localizar el subárbol que contiene en su raíz el valor de clave que se desea extraer.
  - Si no se encuentra no hay nada que hacer.
  - Si el subárbol tiene uno de sus hijos vacío:
    - Basta con que el otro hijo pase a ocupar el lugar de su padre.
  - Cuando ninguno de los subárboles del nodo a extraer está vacío la situación se complica:
    - Si uno de los hijos sustituye al padre ¿qué se hace con el otro?

Operaciones del árbol binario de búsqueda

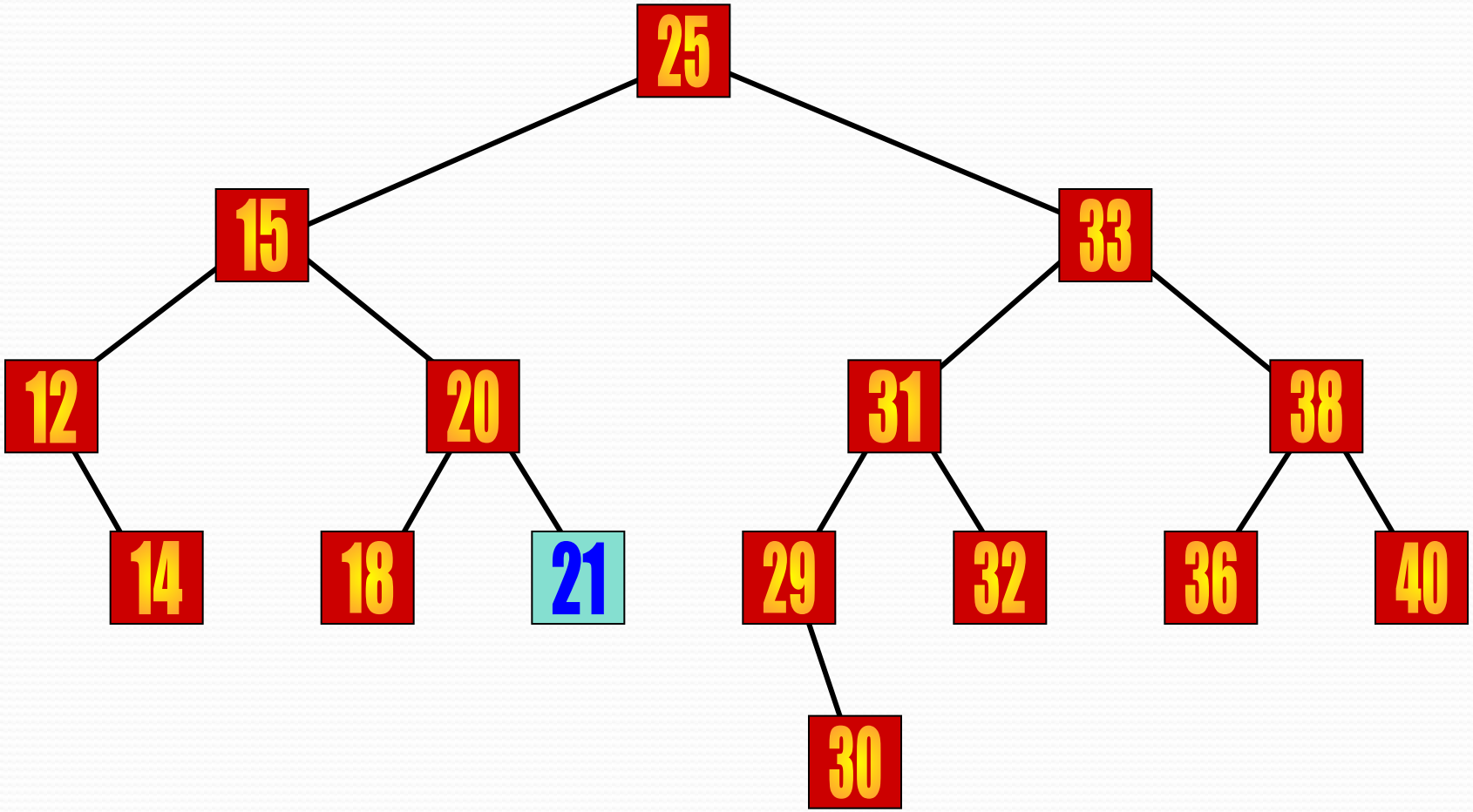
- Cuando ninguno de los subárboles del nodo a extraer esté vacío lo más adecuado es sustituir el valor del nodo a extraer por el valor más cercano y así alterar el árbol lo menos posible.
  - Los candidatos son el sucesor y el predecesor en orden simétrico.
    - Si se elige la opción del sucesor, se extrae del lugar en el que se encuentra y se deposita en el lugar ocupado por el nodo a extraer,
      - Se hace cargo de sus dos hijos.
      - El hijo derecho del sucesor del nodo a extraer, si es que existe, ocupa el lugar del sucesor.
        - Nótese que, por definición de recorrido simétrico, el nodo sucesor de un nodo de grado 2, siempre tiene el subárbol izquierdo vacío.



Árbol binario de búsqueda: Extracción.

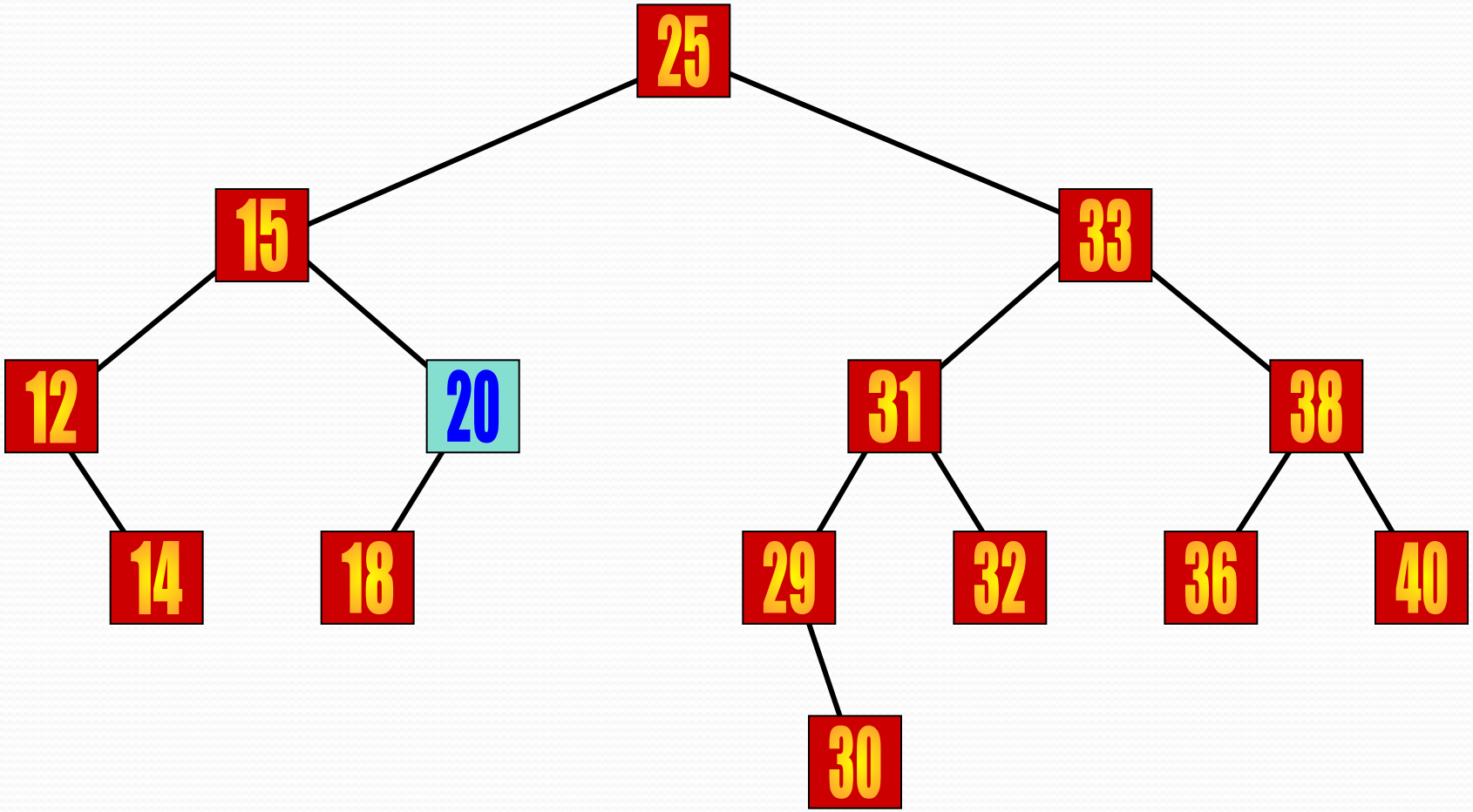


Árbol binario de búsqueda: Extracción.



Quitar un hoja no entraña dificultad

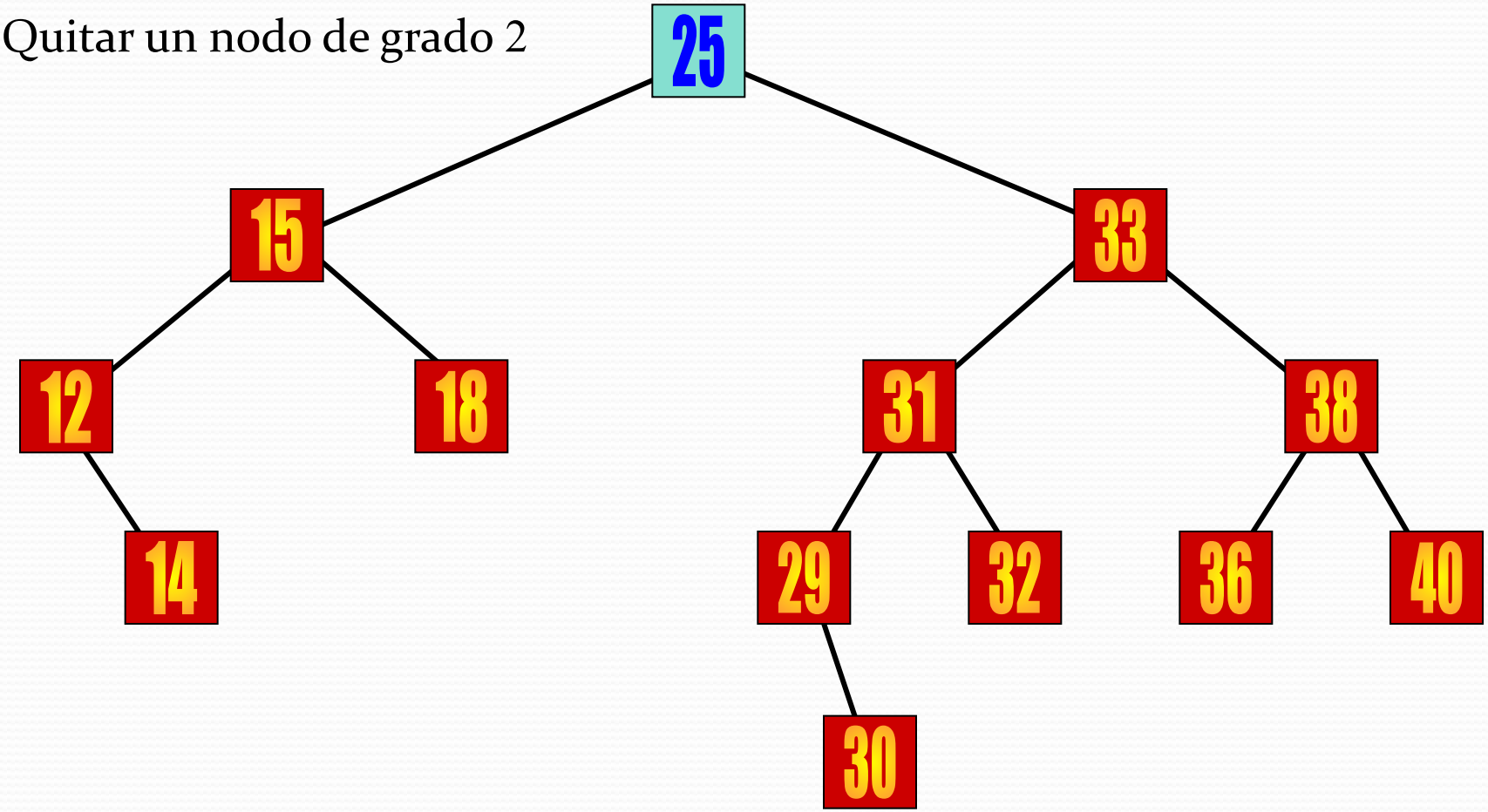
Árbol binario de búsqueda: Extracción.



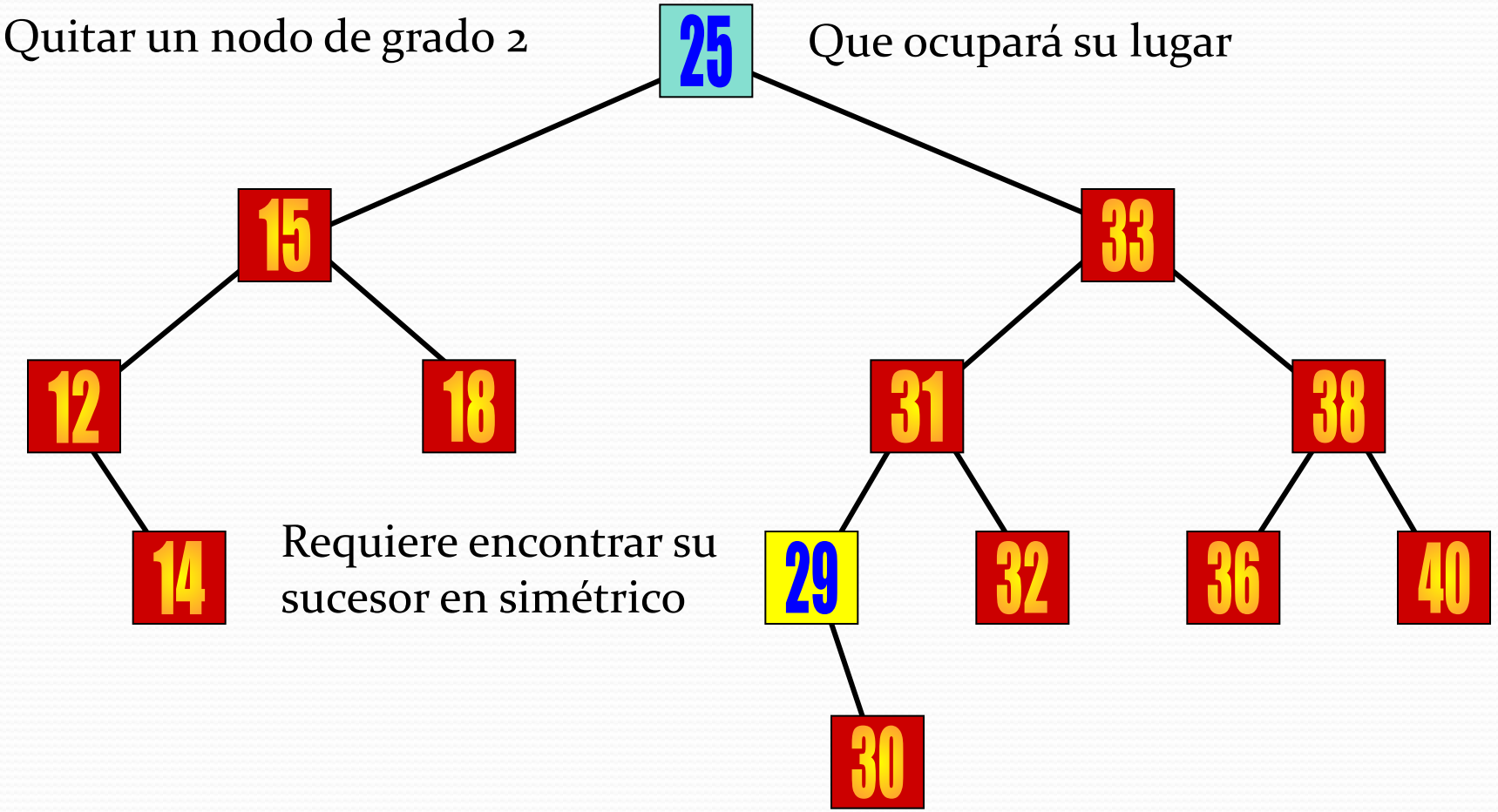
Tampoco la tiene quitar un nodo de grado 1

Árbol binario de búsqueda: Extracción.

Quitar un nodo de grado 2



Árbol binario de búsqueda: Extracción.



Árboles

Implementación de los algoritmos de inserción y extracción

Inicio

Implementación de los algoritmos de inserción y extracción

- Se desarrolla un contenedor asociativo que utiliza el árbol binario de búsqueda como estructura interna.
- Aunque podría haberse hecho, no se usa como base el árbol binario implementado previamente.
  - Se define una estructura nodal y se desarrollan las operaciones necesarias como mecanismos internos del contenedor asociativo.
  - Tal contenedor tiene la necesaria restricción de que sólo puede admitir claves que sean ordenables.

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

- Se parte de una interfaz —*ContenedorOrdenado*—
  - Define un contenedor en el que los elementos responden a la interfaz *Comparable* de *java.lang*.
  - La interfaz *Comparable* especifica un método llamado *compareTo*
    - *D*evuelve -1, 0, ó 1 según el objeto *this* sea:
      - Menor
      - Igual
      - Mayor que aquel con el que se compara.

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
package contenedores;

public interface ContenedorOrdenado {

   public void insertar(Comparable e);

   public void extraer(Comparable e);

   public void vaciar();

   public boolean está(Comparable e);

   public boolean esVacío();

   public int tamaño();
}
```

Interfaz para un contenedor ordenado

Inserta e en el contenedor. Si está no hace nada

Extrae e en el contenedor. Si no está no hace nada

Vacía el contenedor

Devuelve verdadero si e se encuentra en el contenedor

Devuelve verdadero si el contenedor está vacío

Devuelve el número de elementos del contenedor

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
package árbolesBinariosBusqueda;
import contenedores.ContenedorOrdenado;

public class ÁrbolBinarioBúsqueda implements ContenedorOrdenado {

   class NodoBinario {
      Comparable clave;
      NodoBinario[] enlaces;
      public NodoBinario(Comparable info) {
         clave   = info;
         enlaces = new NodoBinario[2];
         enlaces[0] = null;
         enlaces[1] = null;
      }
   }

   protected int numElem;
   private NodoBinario raíz;
   public ÁrbolBinarioBúsqueda() {
      raíz = null;
      numElem = 0;
   }
...
```

Implementación de un contenedor con un árbol binario de búsqueda. Implementa la interfaz ContenedorOrdenado

Clase auxiliar: Estructura nodal del árbol binario

Constructor de NodoBinario

Constructor de ÁrbolBinarioBúsqueda

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
...
   public boolean está(Comparable valor) {
      return está(raíz, valor);
   }
   protected boolean está(NodoBinario nodo, Comparable valor) {
      while (nodo != null)
         if (valor.compareTo(nodo.clave) == 0)
            return true;
         else if (valor.compareTo(nodo.clave) < 0)
            nodo = nodo.enlaces[0];
         else
            nodo = nodo.enlaces[1];
      return false;
     }

   public boolean esVacío() {
      return tamaño() == 0;
   }

   public void extraer(Comparable valor) {
      raíz = extraer(raíz, valor);
   }
...
```

Implementación del método está de la interfaz contenedores.ContenedorOrdenado

Método interno que recorre el árbol para tratar de encontrar la clave

Implementación del método esVacío de la interfaz contenedores.ContenedorOrdenado

Implementación del método extraer de la interfaz contenedores.ContenedorOrdenado

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
...
   private NodoBinario extraer(NodoBinario nodo, Comparable valor){
      if (nodo != null) {
         if (valor.compareTo(nodo.clave) == 0)
            if ((nodo.enlaces[0] == null) || (nodo.enlaces[1] == null)) {
               numElem--;
               if (nodo.enlaces[0] == null)
                  return nodo.enlaces[1];
               else
                  return nodo.enlaces[0];
            }
            else
               nodo.enlaces[1] = extraerSucesor(nodo, nodo.enlaces[1]);
         else {
            if (valor.compareTo(nodo.clave) < 0)
               nodo.enlaces[0] = extraer(nodo.enlaces[0], valor);
            else
               nodo.enlaces[1] = extraer(nodo.enlaces[1], valor);
         }
      }
      return nodo;
   }
...
```

Método interno que extrae una clave del árbol

Se encontró la clave

¿El nodo tiene grado 0 ó 1?

El nodo tiene grado 2; hay que buscar su sucesor en simétrico

Si la clave no se encuentra en el nodo, la extracción continúa en el subárbol adecuado

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
...
   private NodoBinario extraerSucesor(NodoBinario nodoExtraer, NodoBinario nodo) {
      if (nodo.enlaces[0] == null) {
         nodoExtraer.clave = nodo.clave;
         numElem--;
         nodo = nodo.enlaces[1];
      }
      else
         nodo.enlaces[0] = extraerSucesor(nodoExtraer, nodo.enlaces[0]);
      return nodo;
   }

   public void insertar(Comparable valor) {
      raíz = insertar(raíz, valor);
   }
...
```

Implementación del método insertar de la interfaz contenedores.ContenedorOrdenado

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
...
   private NodoBinario insertar(NodoBinario nodo, Comparable valor) {
      if (nodo == null) {
         numElem++;
         nodo = new NodoBinario(valor);
      }
      else {
         if (valor.compareTo(nodo.clave) != 0) {
            if (valor.compareTo(nodo.clave) < 0)
               nodo.enlaces[0] = insertar(nodo.enlaces[0], valor);
            else
               nodo.enlaces[1] = insertar(nodo.enlaces[1], valor);
         }
      }
      return nodo;
   }
...
```

Método interno que inserta una clave en el árbol

El nuevo valor se inserta como hoja

Si valor no se encuentra en nodo, se continúa por el subárbol adecuado

Realización en Java de un contenedor asociativo usando una estructura interna de árbol binario de búsqueda

```
...
   public int tamaño() {
      return numElem;
   }

   public void vaciar() {
      raíz = null;
      numElem = 0;
   }
}
```

Implementación del método tamaño de la interfaz contenedores.ContenedorOrdenado

Implementación del método vaciar de la interfaz contenedores.ContenedorOrdenado

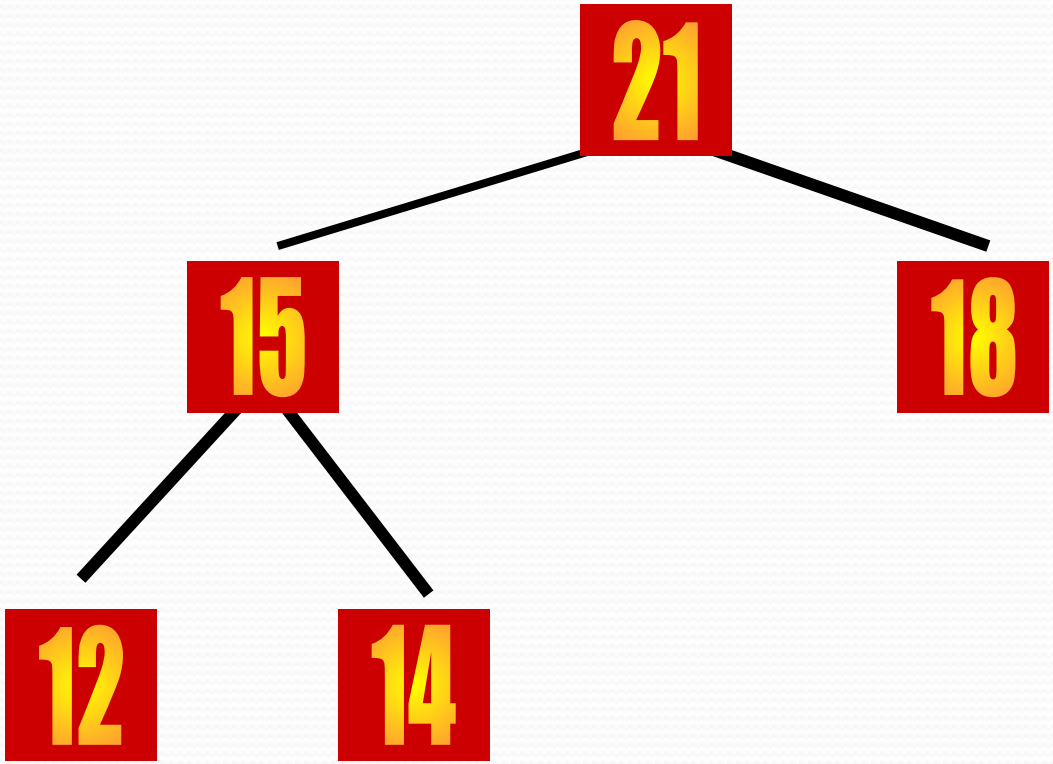
Árboles

Montículos

Inicio

Definición

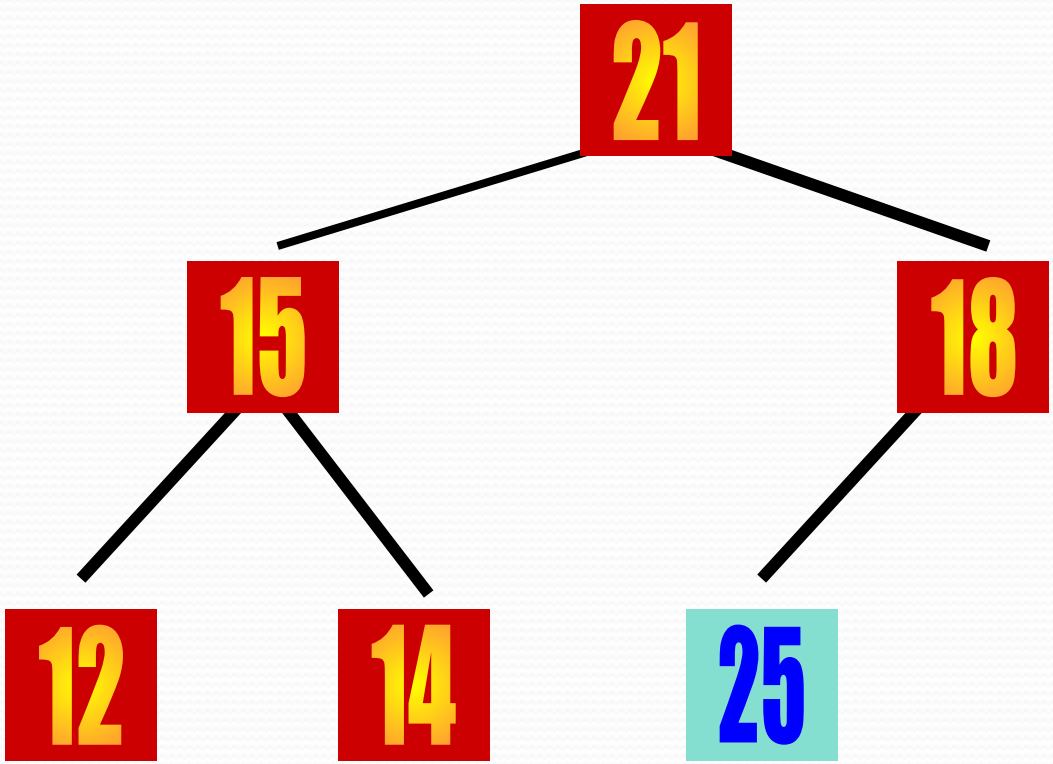
- Es un árbol binario completo en el que la información contenida en cualquier nodo es mayor que la de cualquiera de sus hijos.
  - Significa:
    - Que en la raíz está el valor más grande de los almacenados en el árbol.
    - Que los nodos no tienen que estar totalmente ordenados entre sí, a diferencia del árbol binario de búsqueda en que sí lo están.



Definición

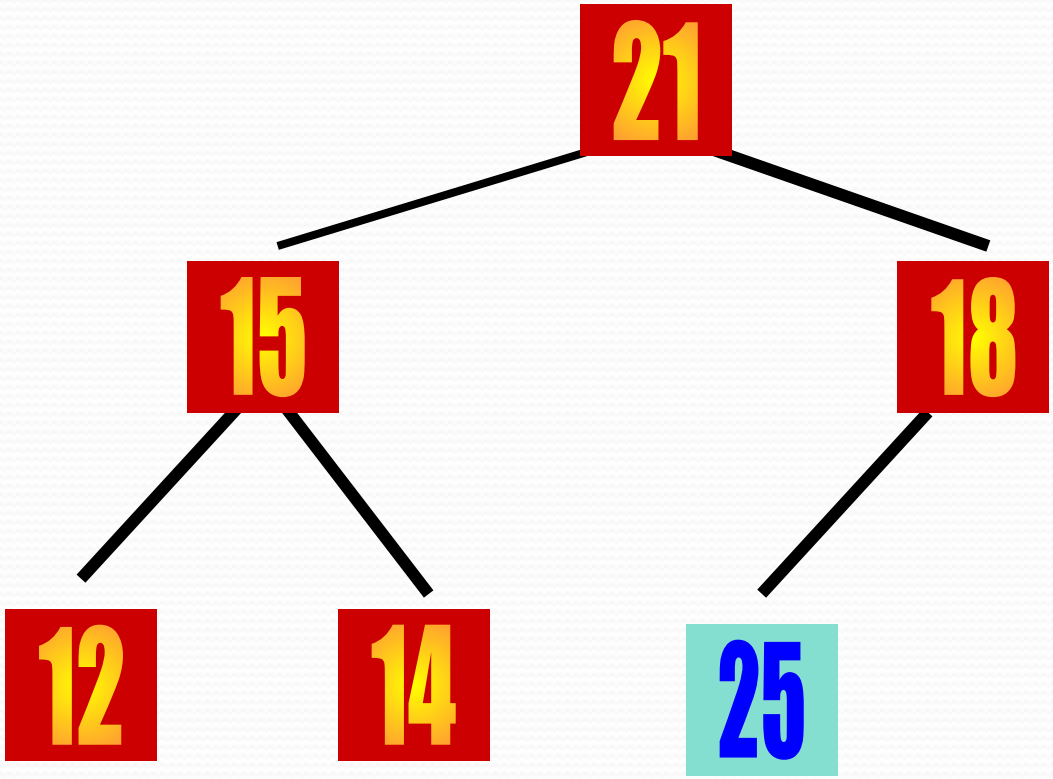
- Por sus características, el montículo es una buena estructura para la representación de colas de prioridad
  - El elemento de mayor prioridad siempre es la raíz.
- Para realizar actualizaciones, hay que tener presente las dos propiedades que definen el montículo:
  - Ser un árbol binario completo
  - Mantener el orden padre/hijos.

Montículo: Inserción.



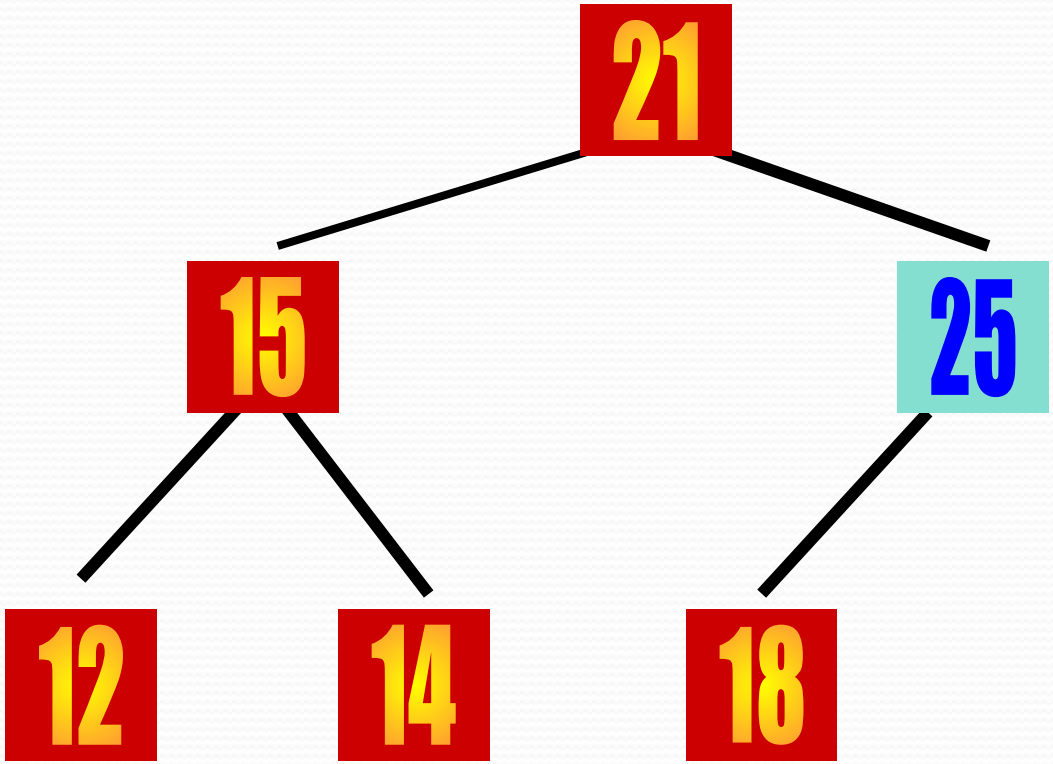
El mantenimiento de la propiedad de árbol binario completo se consigue al insertar el nuevo nodo en el primer hueco libre en el orden del recorrido en anchura.

Montículo: Inserción.



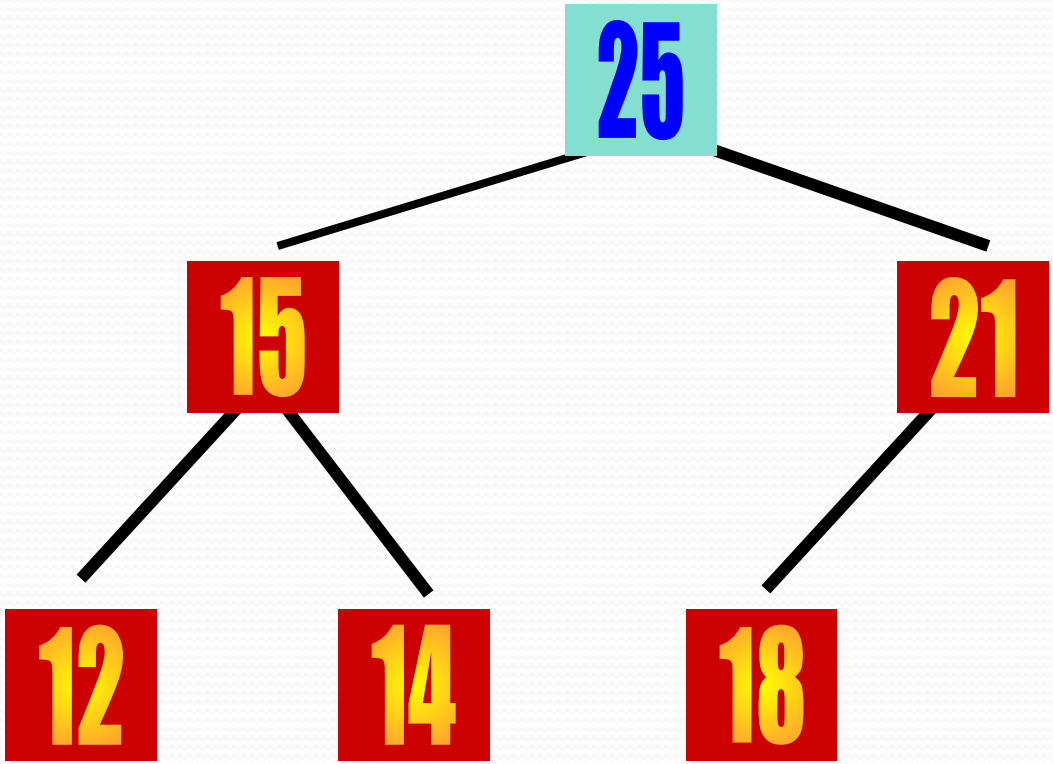
El valor contenido en dicho nodo debe ascender, intercambiándose repetidamente con su padre, hasta que la propiedad de orden haya sido restablecida.

Montículo: Inserción.



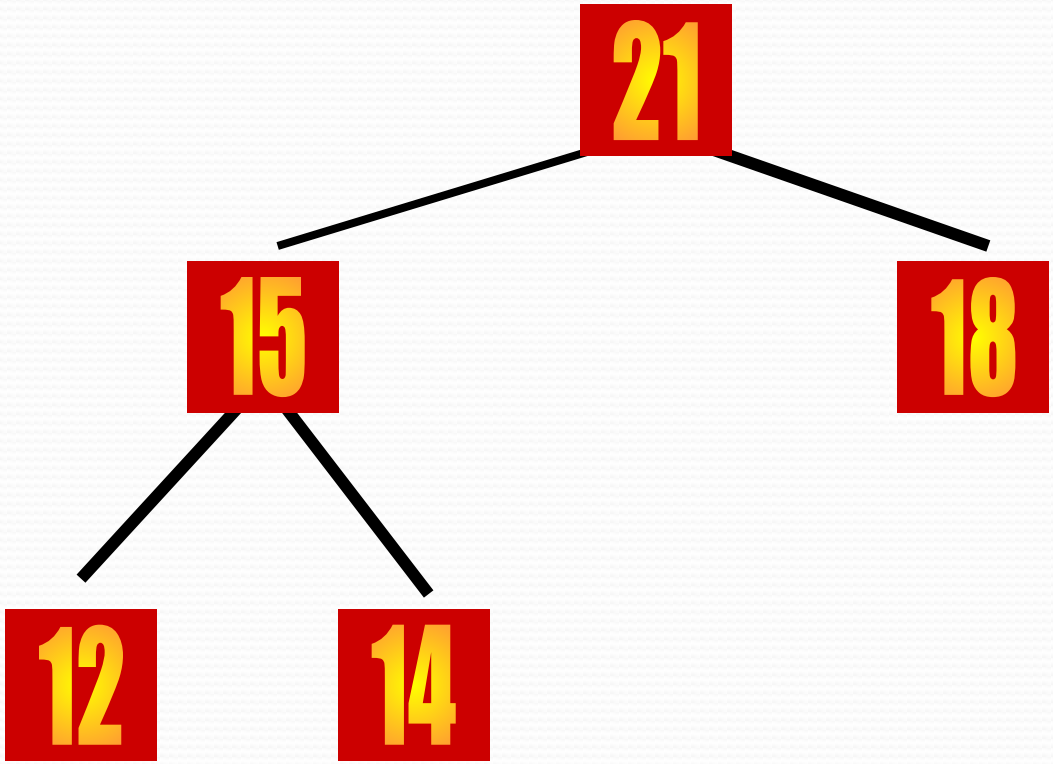
El valor contenido en dicho nodo debe ascender, intercambiándose repetidamente con su padre, hasta que la propiedad de orden haya sido restablecida.

Montículo: Inserción.



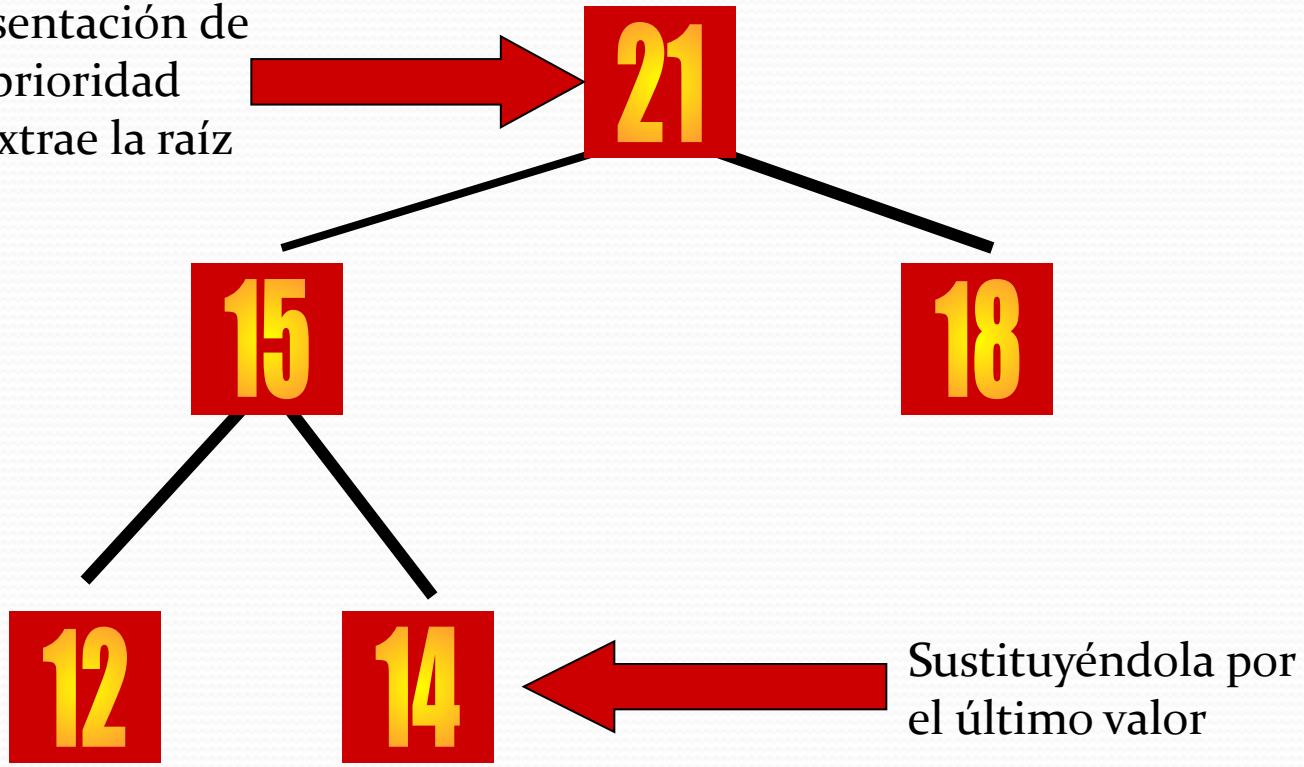
El valor contenido en dicho nodo debe ascender, intercambiándose repetidamente con su padre, hasta que la propiedad de orden haya sido restablecida.

Montículo: Extracción.



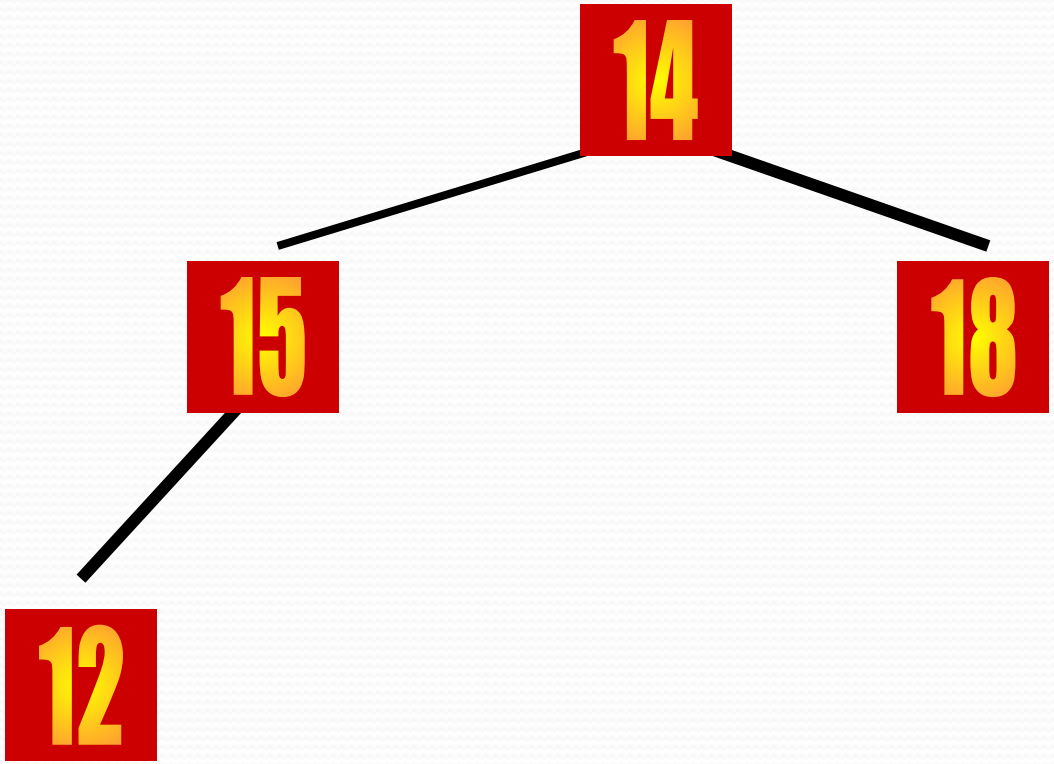
Montículo: Extracción.

Como representación de una cola de prioridad siempre se extrae la raíz



Sustituyéndola por el último valor

Montículo: Extracción.



Luego hay que hundir el valor mediante intercambios con el mayor de sus hijos hasta su posición

En la inserción y en la extracción

- El aspecto más costoso podría ser la localización del primer hueco o del último nodo
  - Tratándose de un árbol binario completo, es posible utilizar la representación secuencial contigua para árboles binarios completos
    - Tanto el hueco como el último nodo siempre se localizan directamente al final de los elementos almacenados
      - Sólo hay que hacer, en cada caso, del orden de $log_2 N$ operaciones de comparación e intercambio
    - Se aprovecha el que la relación posicional entre un padre y sus hijos queda establecida por sencillas expresiones aritméticas

Árboles

Implementación de una cola de prioridad

Inicio

Implementación en Java de una cola de prioridad

```
package montículos;
import java.util.ArrayList;

public class ColaPrioridad {
   ArrayList montículo;
   int numElementos = 0;

   public ColaPrioridad() {
      montículo = new ArrayList(15);
   }

   public ColaPrioridad(int capacidadInicial) {
      montículo = new ArrayList(capacidadInicial);
   }
...
```

Implementación de una cola de prioridad usando un montículo

Lista de elementos de la cola en un ArrayList

Tamaño actual de la cola

Constructor por defecto con capacidad inicial para un árbol con cuatro niveles del montículo (15 nodos).
Cada vez que se llena, crece un nivel más (dobla su tamaño)

Constructor con capacidad inicial elegible

Implementación en Java de una cola de prioridad

```
...
   private void intercambiar(int x, int y) {
      Object z = montículo.get(x);
      montículo.set(x, montículo.get(y));
      montículo.set(y, z);
   }


   private void flotar(int p) {
      if ((p>0) &&
          (((Comparable)montículo.get(p)).compareTo((Comparable)montículo.get((p-1)/2)))>0){
         intercambiar(p, (p - 1) / 2);
         flotar((p - 1) / 2);
      }
   }


   public void insertar(Comparable e) {
      if (numElementos == montículo.size())
         montículo.ensureCapacity(montículo.size() * 2 + 1);

      montículo.add(numElementos++, e);
      flotar(numElementos - 1);
   }
...
```

Intercambia los valores de sus parámetros (x, y)

Hace ascender el elemento del montículo que está en p hasta que alcance la posición que le corresponde según su prioridad

Si el elemento en p no es la raíz, se compara con su padre para valorar cuál tiene mayor prioridad

Se intercambian los valores cuando el hijo es mayor

Se continúa la operación flotar con el padre

Añade un elemento al final del montículo y llama a flotar para hacerlo llegar a su posición

Implementación en Java de una cola de prioridad

```
...
private void hundir(int p) {
      int izq = 2 * p + 1;
      int der = 2 * p + 2;
      if (izq < numElementos) {
         if ((der < numElementos) &&
             (((Comparable)montículo.get(der)).compareTo((Comparable)montículo.get(izq)))>0) {

            if (((Comparable)montículo.get(der)).compareTo((Comparable)montículo.get(p))>0) {
               intercambiar(p, der);
               hundir(der);
            }
         }

         else {
            if (((Comparable)montículo.get(izq)).compareTo((Comparable)montículo.get(p))>0) {
               intercambiar(p, izq);
               hundir(izq);
            }
         }
      }
   }
...
```

Hace descender el elemento del montículo que está en p hasta que alcance la posición que le corresponde según su prioridad

Tiene hijos

Tiene ambos hijos y el derecho es el de mayor prioridad

La prioridad del hijo derecho es mayor que la de p

Sólo tiene hijo izquierdo o éste es el de mayor prioridad

La prioridad del hijo izquierdo es mayor que la de p

Implementación en Java de una cola de prioridad

```
...
   public void vaciar() {
      montículo = null;
      numElementos = 0;
   }

   public void extraer() {
      montículo.set(0, montículo.get(--numElementos));
      hundir(0);
   }

   public Comparable examinar() {
      return (Comparable) montículo.get(0);
   }

   public int tamaño() {
      return numElementos;
   }
}
```

Vacía la cola de prioridad

Extrae el último elemento, lo sitúa en lugar de la raíz y llama a hundir para colocarlo en su posición

Devuelve el elemento de mayor prioridad de la cola

Devuelve el número de elementos de la cola de prioridad