TEMA 3
El tipo árbol

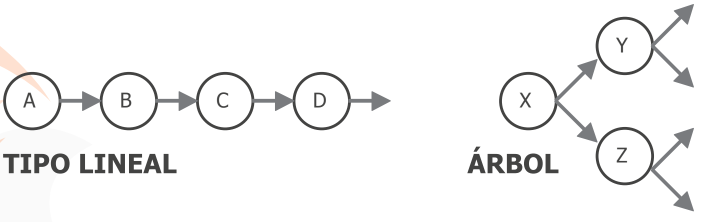
PROGRAMACIÓN Y ESTRUCTURAS DE DATOS

Tipo árbol

- 1. Definiciones generales
- 2. Árboles binarios
- 3. Árboles de búsqueda
  - 3.1. Árboles binarios de búsqueda
  - 3.2. Árboles AVL
  - 3.3. Árboles 2-3
  - 3.4. Árboles 2-3-4

1. Definiciones generales (I)

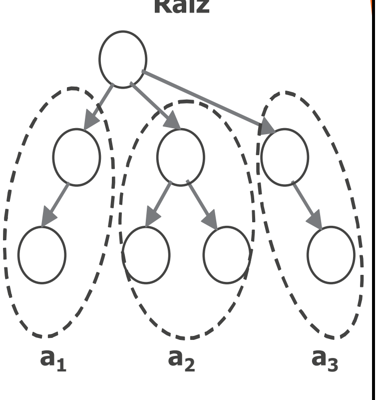
- La estructura de datos árbol aparece porque los elementos que lo constituyen mantienen una estructura jerárquica, obtenida a partir de estructuras lineales, al eliminar el requisito de que cada elemento tiene como máximo un sucesor:



- Los elementos de los árboles se llaman nodos

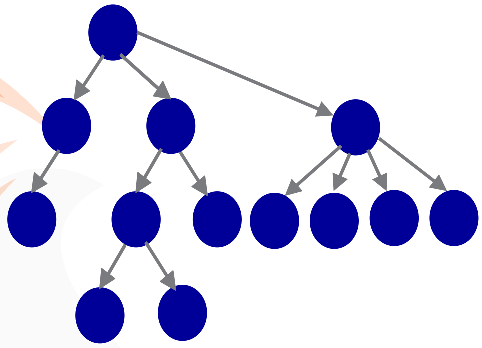
1. Definiciones generales (II)

- Definición inductiva de árbol:
  - un único nodo es un **árbol** (**raíz**)
  - dados *n* árboles $a_1, \ldots, a_n$ se puede construir uno nuevo como resultado de enraizar un nuevo nodo con los *n* árboles. Los árboles a_i pasan a ser **subárboles** del nuevo árbol y el nuevo nodo se convierte en raíz del nuevo árbol
- Árbol vacío o nulo $\Rightarrow$ 0 nodos

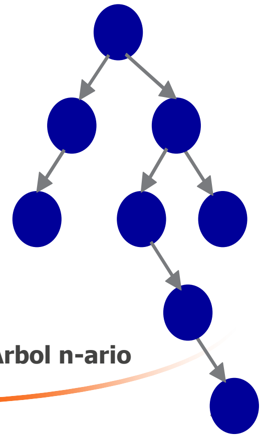


1. Definiciones generales (III)

- El proceso de enraizar puede involucrar:
  - un nº indeterminado de subárboles (**árboles generales**)
  - o bien, un nº máximo *n* de subárboles (**árboles n-arios**)



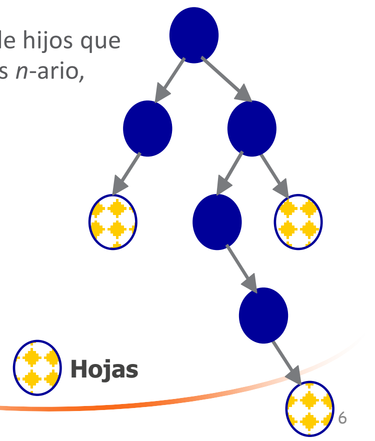
Árbol general



Árbol n-ario

1. Definiciones generales (IV)

- Un árbol n-ario con n = 2 se denomina árbol binario
- La información almacenada en los nodos del árbol se denomina etiqueta
- Las hojas son árboles con un solo nodo (árboles binarios: árbol compuesto por una raíz y 2 subárboles vacíos)
- Grado de un árbol es el número máximo de hijos que pueden tener sus subárboles (si el árbol es *n*-ario, el grado es *n*)

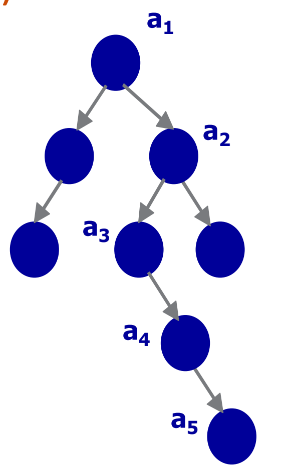


Hojas

1. Definiciones generales (V)

- Camino:
  - es una secuencia $a_1, \ldots, a_s$ de árboles tal que, $\forall\, i \in \{1 \ldots s - 1\}$, a_{i+1} es subárbol de a_i
  - el número de subárboles de la secuencia menos uno, se denomina **longitud del camino**

    (Consideraremos que existe un camino de longitud 0 de todo subárbol a sí mismo)



$\forall\, i \in \{1 \ldots 4\}$ a_{i+1} es subárbol de a_i

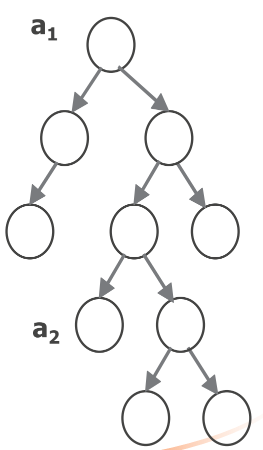
Longitud = 5 - 1 = 4

1. Definiciones generales (VI)

- a_1 es ascendiente de a_2 (y a_2 es descendiente de a_1) si existe un camino $a_1, \ldots, a_2$

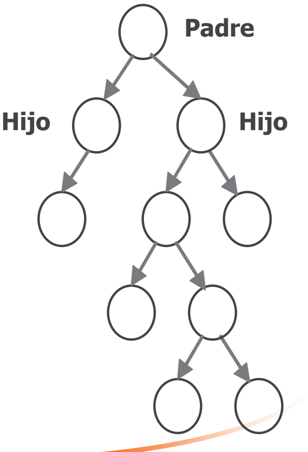
  (Según la definición de camino, todo subárbol es ascendiente/descendiente de sí mismo)

- Los ascendientes (descendientes) de un árbol, excluido el propio árbol, se denominan ascendientes (descendientes) propios



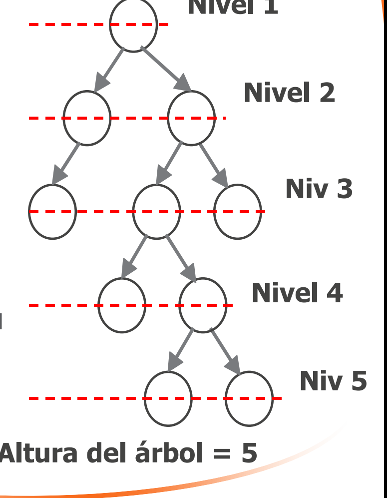
1. Definiciones generales (VII)

- Padre es el primer ascendiente propio, si existe, de un árbol
- Hijos son los primeros descendientes propios, si existen, de un árbol
- Hermanos son subárboles con el mismo padre
- Profundidad de un subárbol es la longitud del único camino desde la raíz a dicho subárbol



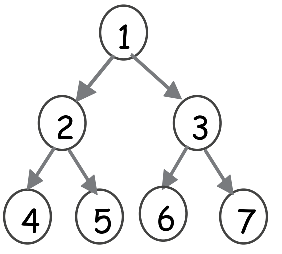
1. Definiciones generales (VIII)

- Nivel de un nodo:
  - el nivel de un árbol vacío es 0
  - el nivel de la raíz es 1
  - si un nodo está en el nivel *i*, sus hijos están en el nivel *i + 1*
- Altura (profundidad) de un árbol:
  - es el máximo nivel de los nodos de un árbol

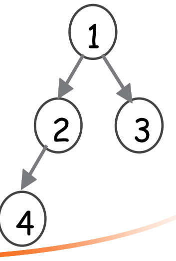


1. Definiciones generales (IX)

- Árbol lleno es un árbol en el que todos sus subárboles tienen *n* hijos (siendo *n* el grado del árbol) y todas sus hojas tienen la misma profundidad



- Árbol completo es un árbol cuyos nodos corresponden a los nodos numerados (la numeración se realiza desde la raíz hacia las hojas y, en cada nivel, de izquierda a derecha) de 1 a *n* en el árbol lleno del mismo grado. Todo árbol lleno es completo



2. Árboles binarios

- Definición de árbol binario y propiedades
- Especificación algebraica
- Recorridos
- Enriquecimiento de la especificación
- Representación secuencial y enlazada
- Otras operaciones interesantes
- Ejercicios

2. Árboles binarios

DEFINICIÓN

- Un árbol binario es un conjunto de elementos del mismo tipo tal que:
  - o bien es el conjunto vacío, en cuyo caso se denomina **árbol vacío** o **nulo**
  - o bien no es vacío, y por tanto existe un elemento distinguido llamado **raíz**, y el resto de los elementos se distribuyen en dos subconjuntos disjuntos, cada uno de los cuales es un árbol binario llamados, respectivamente **subárbol izquierdo** y **subárbol derecho** del árbol original

2. Árboles binarios

PROPIEDADES (I)

- Propiedades:
  - El máximo número de nodos en un nivel i de un árbol binario es $N(i) = 2^{i-1}$, $i \geq 1$

Demostración

Base inducción

nivel 1 (raíz): $N(1) = 2^{1-1} = 2^0 = 1$ (se cumple)

Paso inductivo

Se desea probar $N(i\text{-}1) \Rightarrow N(i)$, es decir, a partir de la suposición "temporal" de que N es cierta para $i\text{-}1$ debemos probar que es cierta para i

nivel i - 1: $N(i\text{-}1) = 2^{(i-1)-1} = 2^{i-2}$ (suponemos cierto)

nivel i : $N(i) = N(i\text{-}1) * 2 = 2^{i-2} * 2 = 2^{i-2+1} = 2^{i-1}$

2. Árboles binarios

PROPIEDADES (II)

– El máximo número de nodos en un árbol binario de altura *k* es $N(k) = 2^k - 1,\ k \geq 1$

<u>Demostración</u>

$$\left.\begin{array}{l} \text{nivel 1: } 2^{1-1} = 1 \text{ nodo} \\ \text{nivel 2: } 2^{2-1} = 2 \text{ nodos} \\ \text{nivel 3: } 2^{3-1} = 4 \text{ nodos} \\ \ldots \\ \text{nivel k: } 2^{k-1} \text{ nodos} \end{array}\right\} \text{Altura } k = 2^{1-1} + 2^{2-1} + \ldots + 2^{k-1} =$$

$$\left(\text{S.P.G. } (r = 2,\ a_1 = 2^0,\ n = k)\right)$$

$$= 1\,(2^k - 1) / 2 - 1 = 2^k - 1$$

2. Árboles binarios

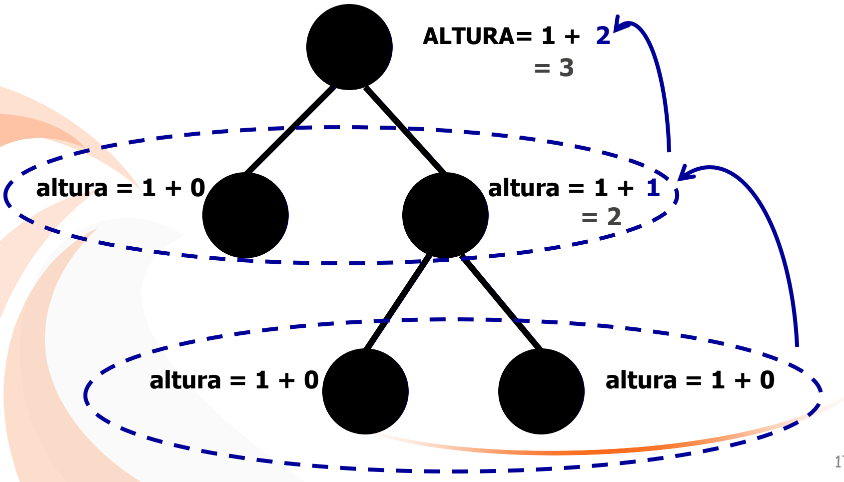
ESPECIFICACIÓN ALGEBRAICA (I)

```
MODULO ARBOLES_BINARIOS
    USA BOOL, NATURAL

    PARAMETRO TIPO item
        OPERACIONES
            error_item( ) → item
    FPARAMETRO
    TIPO arbin
    OPERACIONES
            crea_arbin( ) → arbin
            enraizar( arbin, item, arbin ) → arbin
            raiz( arbin ) → item
            esvacio( arbin ) → bool
            hijoiz, hijode( arbin ) → arbin
            altura( arbin ) → natural

VAR i, d: arbin; x: item;
ECUACIONES
raiz( crea_arbin( ) ) = error_item( )
raiz( enraizar( i, x, d ) ) = x
hijoiz( crea_arbin( ) ) = crea_arbin( )
hijoiz( enraizar( i, x, d  ) ) = i
hijode( crea_arbin( ) ) = crea_arbin( )
hijode( enraizar( i, x, d  )) = d
esvacio( crea_arbin( ) ) = CIERTO
esvacio( enraizar( i, x, d ) ) = FALSO
altura( crea_arbin( ) ) = 0
altura( enraizar( i, x, d ) ) =
      1 + max ( altura( i ), altura( d ) )

FMODULO
```

2. Árboles binarios

ESPECIFICACIÓN ALGEBRAICA (II)

altura(crea_arbin( )) = 0
altura = 1 + max(altura(hijoiz), altura(hijode))



2. Árboles binarios

RECORRIDOS

- Recorrer un árbol es visitar cada nodo del árbol una sola vez
- Recorrido de un árbol es la lista de etiquetas del árbol ordenadas según se visitan los nodos
- Se distinguen dos categorías básicas de recorrido:
  - **recorridos en profundidad**
  - **recorridos en anchura** o **por niveles**

2. Árboles binarios

RECORRIDOS EN PROFUNDIDAD (I)

- Si representamos por I: ir hacia la izquierda, R: visitar o escribir el item, D: ir hacia la derecha, existen 6 posibles formas de recorrido en profundidad: RID, IRD, IDR, RDI, DRI y DIR. Si sólo queremos hacer los recorridos de izquierda a derecha quedan 3 formas de recorrido:

  1. RID o **preorden** (orden previo)

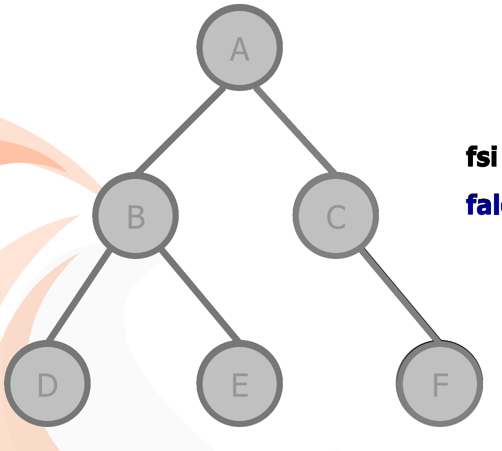
  2. IRD o **inorden** (orden simétrico)

  3. IDR o **postorden** (orden posterior)

  (El recorrido en postorden es el inverso especular del recorrido preorden, es decir, se recorre el árbol en preorden, visitando primero el subárbol derecho antes que el izquierdo, y se considera la lista resultante como el inverso de la solución)

2. Árboles binarios

RECORRIDOS EN PROFUNDIDAD (II)




```
algoritmo preorden ( a : arbin )
si ( no esvacio( a ) ) entonces
    escribe ( raiz ( a ) )
    preorden ( hijoiz ( a ) )
    preorden ( hijode ( a ) )
fsi
falgoritmo
```

PREORDEN: A B D E C F

2. Árboles binarios

RECORRIDOS EN PROFUNDIDAD (III)

```
algoritmo inorden ( a : arbin )

si ( no esvacio( a ) ) entonces
      inorden ( hijoiz ( a ) )
      escribe ( raiz ( a ) )
      inorden ( hijode ( a ) )
fsi
falgoritmo
```

```
algoritmo postorden ( a : arbin )

si ( no esvacio( a ) ) entonces
      postorden ( hijoiz ( a ) )
      postorden ( hijode ( a ) )
      escribe ( raiz ( a ) )
fsi
falgoritmo
```


2. Árboles binarios

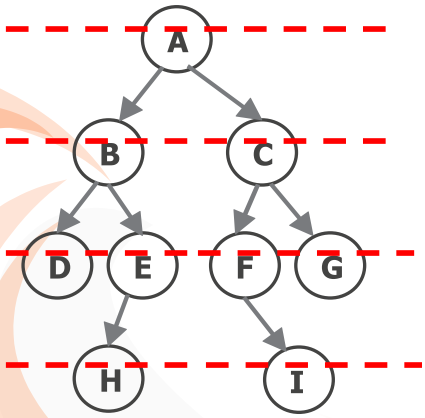
RECORRIDO EN ANCHURA (NIVELES)

- Consiste en visitar los nodos desde la raíz hacia las hojas, y de izquierda a derecha dentro de cada nivel

```
algoritmo niveles ( a : arbin )
var c: cola de arbin; aux: arbin; fvar
encolar(c, a)
mientras no esvacia(c) hacer
    aux := cabeza(c)
    escribe (raiz(aux))
    desencolar(c)
    si no esvacio(hijoiz(aux)) entonces encolar(c, hijoiz(aux))
    si no esvacio(hijode(aux)) entonces encolar(c, hijode(aux))
fmientras
falgoritmo
```

2. Árboles binarios

EJEMPLO DE RECORRIDOS (I)

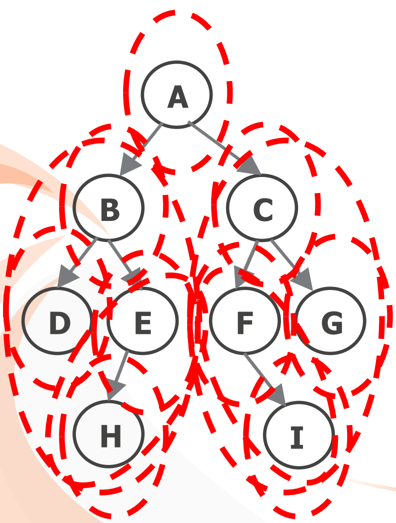


- Niveles:

a b c d e f g h i

2. Árboles binarios

EJEMPLO DE RECORRIDOS (II)

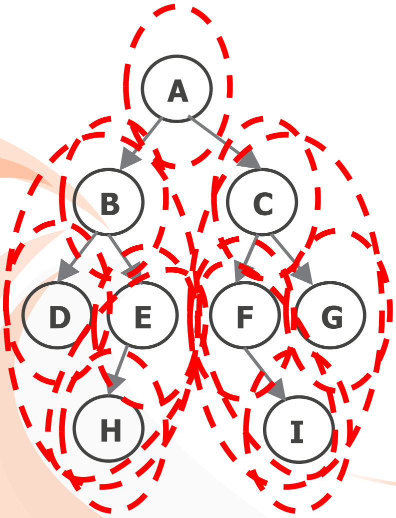


- Inorden:

d b h e a f i c g

2. Árboles binarios

EJEMPLO DE RECORRIDOS (III)

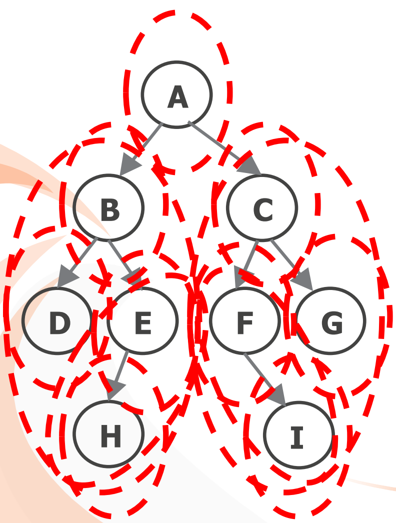


- Postorden:

d h e b i f g c a

2. Árboles binarios

EJEMPLO DE RECORRIDOS (IV)



- Preorden:

a b d e h c f i g

2. Árboles binarios

ENRIQUECIMIENTO DE LA ESPECIFICACIÓN

OPERACIONES

preorden, inorden, postorden(arbin) → lista
nodos (arbin) → natural
eshoja (arbin) → bool

VAR i, d: arbin; x: item;

ECUACIONES

preorden(crea_arbin( )) = crea_lista( )
preorden(enraizar(i, x, d)) = concatenar(insiz(x, preorden(i)), preorden(d))
inorden(crea_arbin( )) = crea_lista( )
inorden(enraizar(i, x, d)) = concatenar(insde(inorden(i), x), inorden(d))
postorden(crea_arbin( )) = crea_lista( )
postorden(enraizar(i, x, d)) = insde(concatenar(postorden(i), postorden(d)), x)
nodos(crea_arbin( )) = 0
nodos(enraizar(i, x, d)) = 1 + nodos(i) + nodos(d)
eshoja(crea_arbin( )) = FALSO
eshoja(enraizar(i, x, d)) = esvacio(i) Λ esvacio(d)

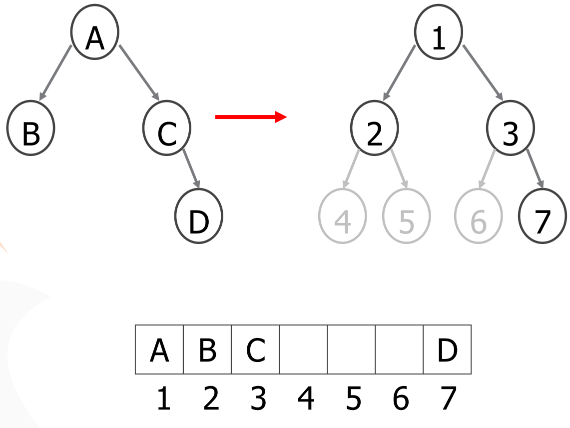
2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (I)

- Representación secuencial
  Se numeran secuencialmente los nodos del árbol hipotéticamente lleno desde la raíz a las hojas por niveles (comenzando por el nivel 1, después el nivel 2, etc.) y de izquierda a derecha en cada nivel. La representación secuencial se puede hacer usando un vector unidimensional:
  - la raíz se guarda en la dirección 1
  - si un nodo *n* está en la dirección *i*, entonces su hijo izquierdo estará en la dirección *2i* y su hijo derecho en la dirección *2i + 1*

2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (II)



2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (III)

- Representación enlazada

```
typedef int TItem;
class TNodo;

class TArbin{
    public:
      TArbin ( );                                   //CONSTRUCTOR
      TArbin (const TArbin & origen);               //CONSTRUCTOR DE COPIA
      ~TArbin ( );                                  //DESTRUCTOR
      TArbin & operator = (const TArbin & a);       //ASIGNACIÓN
      void Enraizar ( TArbin &iz, const TItem c, TArbin &de);
      TItem & Raiz ( );
      TArbin HijoIz ( ); TArbin HijoDe ( );
      bool EsVacio ( );
      int Altura ( );
    private:
      void Copiar (const TArbin & origen);
      TNodo *farb;
      TItem item_error;
};
```

2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (IV)

```
class TNodo{
     friend class TArbin;
     private:
        TItem fitem;
        TArbin fiz, fde;
};

/* -----------------------------------------------------------------------------------------*/
TArbin::TArbin ( ) {farb = NULL; }
/* -----------------------------------------------------------------------------------------*/
TArbin::TArbin (const TArbin & origen){
     Copiar (origen);
}
/* -----------------------------------------------------------------------------------------*/
void
TArbin::Copiar (const TArbin & origen){
     if (origen.farb != NULL){
        TNodo *aux = new TNodo( );
        aux → fitem = origen.farb → fitem;
        farb = aux;
        (farb → fiz).Copiar (origen.farb → fiz);
        (farb → fde).Copiar (origen.farb → fde);
     }
     else farb = NULL;
}
```


Tema 3. El tipo árbol

2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (V)

```
TArbin::~TArbin ( ) {
     if (farb != NULL){
        delete farb;
        farb = NULL;}
}
/* -----------------------------------------------------------------------------------*/
TArbin &
TArbin::operator = (const TArbin & a){
     this → ~TArbin( );
     Copiar (a);
    return *this;
}
/* -----------------------------------------------------------------------------------*/
void
TArbin::Enraizar (TArbin &iz, const TItem c, TArbin &de){
     TNodo *aux = new TNodo( );
     aux → fitem = c;
     (aux → fiz).farb = iz.farb;
     (aux → fde).farb = de.farb;
     iz.farb = de.farb = NULL;
     this → ~TArbin ( );
     farb = aux;
}
//deja vacíos el árbol original (*this), iz y de
```

2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (VI)

```
TItem &
TArbin::Raiz ( ){
    if (farb != NULL) return farb → fitem;
    else return item_error;
}

/* ------------------------------------------------------------------------------------------*/
TArbin
TArbin::HijoIz ( ){
    if (farb != NULL) return farb → fiz;
    else return TArbin( );
}
/* ------------------------------------------------------------------------------------------*/
TArbin
TArbin::HijoDe ( ){
    if (farb != NULL) return farb → fde;
    else return TArbin( );
}
```


2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (VII)

```
bool
TArbin::EsVacio ( ){
    return (farb == NULL)
}

/* ------------------------------------------------------------------------------------------*/

int
TArbin::Altura ( ){
    int a1, a2;

    if (farb != NULL){
      a1 = (farb → fiz).Altura( );
      a2 = (farb → fde).Altura( );
      return (1 + (a1 < a2 ? a2 : a1));
    }
  else return 0;
}
```

2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (VIII)

```
/* Programa de prueba

int
main ( ){

    TArbin a, b, c;

    a.Enraizar (b, 1, c);
    b.Enraizar (a, 2, c);
    cout << "el hijo izquierda del árbol contiene un " << (b.HijoIz( )).Raiz( );
    // ESCRIBE 1
    cout << "la altura del árbol es " << b.Altura() << endl;
    // ESCRIBE 2
}
```


2. Árboles binarios

REPRESENTACIÓN SECUENCIAL Y ENLAZADA (IX)

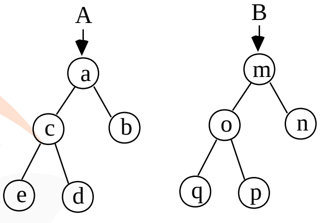
¿Constructor y destructor de TNodo?

```
TNodo::TNodo( ):fiz(),fde(){
    fitem=0;
}

TNodo::~TNodo( ){
    fitem=0;
}
```

2. Árboles binarios

OTRAS OPERACIONES INTERESANTES (I)

- Además de todas las operaciones vistas anteriormente, utilizaremos las operaciones de asignación y "movimiento" de árboles e iteradores:

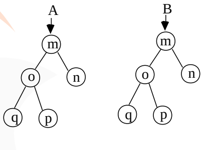


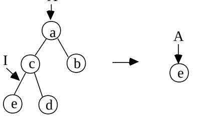
I, J = Iteradores

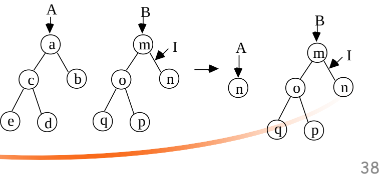
2. Árboles binarios

OTRAS OPERACIONES INTERESANTES (II)

- a) *Asignación (copia) entre árboles e iteradores*:
  - a1) A = B. Hace una copia de B en A
  - a2) A = I. Hace una copia sobre el árbol A, de la rama del árbol a la que apunta el Iterador I



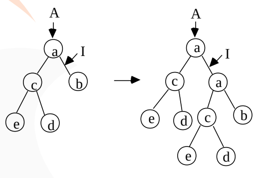




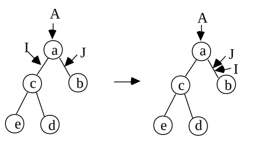
2. Árboles binarios

OTRAS OPERACIONES INTERESANTES (III)

- a) *Asignación (copia) entre árboles e iteradores*:
  - a3) I = A. Hace una copia sobre la rama del árbol a la que apunta el Iterador I del árbol A



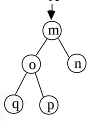
  - a4) I = J. Sirve para inicializar el Iterador I de forma que apunte al mismo nodo al que apunta el Iterador J



2. Árboles binarios

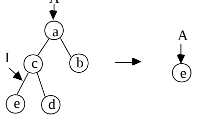
OTRAS OPERACIONES INTERESANTES (IV)

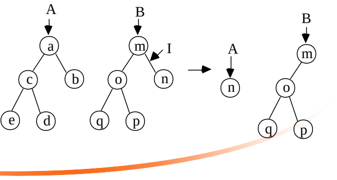
- b) *Movimiento de ramas entre árboles e iteradores*:
  - b1) Mover (A, B). Mueve el árbol B al árbol A. B se queda vacío



B vacío

  - b2) Mover (A, I). Mueve la rama del árbol a la que apunta el Iterador I al árbol A

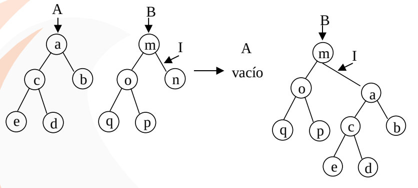


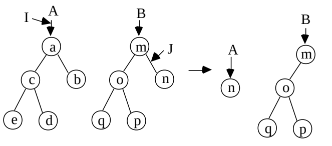


2. Árboles binarios

OTRAS OPERACIONES INTERESANTES (V)

- b) *Movimiento de ramas entre árboles e iteradores*:
  - b3) Mover (I, A). Mueve el árbol A a la rama del árbol a la que apunta el Iterador I
  - b4) Mover (I, J). Mueve la rama del árbol a la que apunta el Iterador J a la rama del árbol a la que apunta el Iterador I





2. Árboles binarios

EJERCICIOS *recorridos*

1a) Dado el siguiente árbol binario, calcular l[redacted] recorridos preorden, postorden, inorden [redacted] niveles

1b) ¿Se puede resconstruir un árbol binario dando solamente su recorrido inorden? ¿Cuántos recorridos como mínimo son necesarios? ¿Cuáles?

2. Árboles binarios

EJERCICIOS *nodosHoja*

2) Sea un árbol binario. Especificar la sintaxis y semántica de las operaciones:

nodosHoja, que devuelve el número de nodos hoja de un árbol binario

2. Árboles binarios

EJERCICIOS *simetricos* y *todos*

3) Sea un árbol binario cuyas etiquetas son números naturales. Especificar la sintaxis y semántica de las operaciones:

a) **simétricos**, que comprueba que 2 árboles binarios son simétricos

b) **todos**, que calcula la suma de todas las etiquetas de los nodos del árbol

Nota: Especificar la sintaxis de todas las operaciones de árboles binarios usadas

2. Árboles binarios

EJERCICIOS *transforma*

4) Se define la operación transforma que recibe un árbol binario y devuelve un árbol binario. Explicar qué hace esta operación detallando el comportamiento de las dos ecuaciones que aparecen a continuación:

VAR *i, d*: arbin; *x*: item,

transforma(crea_arbin( )) = crea_arbin( )

transforma(enraizar(*i, x, d*)) =

enraizar(transforma(*i*), *x* + todos(*i*) + todos(*d*), transforma(*d*))

Nota: La operación *todos* calcula la suma de todas las etiquetas de los nodos del árbol (números naturales)

2. Árboles binarios

EJERCICIOS *quita_hojas*

5) Utilizando exclusivamente las operaciones *crea_arbin()* y *enraizar(arbin, item, arbin)* definir la sintaxis y la semántica de la operación quita_hojas que actúa sobre un árbol binario y devuelve el árbol binario original sin sus hojas

2. Árboles binarios

EJERCICIOS *dos_hijos*

6) Especificar la sintaxis y la semántica de la operación dos_hijos que actúa sobre un árbol binario y devuelve CIERTO si todos los nodos tienen dos hijos (excepto los nodos hoja)

2. Árboles binarios

Preguntas de tipo test: Verdadero vs. Falso

- El nivel de un nodo en un árbol coincide con la longitud del camino desde la raíz a dicho nodo
- Dado un único recorrido de un árbol binario lleno, es posible reconstruir dicho árbol
- Un árbol binario completo con n nodos y altura k es un árbol binario lleno para esa misma altura

3. Árboles de búsqueda (I)

- Árboles de búsqueda = Árboles n-arios de búsqueda = Árboles multicamino de búsqueda
- Son un tipo particular de árboles, que pueden definirse cuando el tipo de los elementos del árbol posee una relación $\leq$ de orden total
- Un árbol multicamino de búsqueda T es un árbol n-ario vacío o cumple las siguientes propiedades:
  - 1. La raíz de T contiene $A_0, \ldots, A_{n-1}$ subárboles y $K_1, \ldots, K_{n-1}$ etiquetas
  - 2. $K_i < K_{i+1}$, $1 \leq i < n-1$
  - 3. Todas las etiquetas del subárbol A_i son:

    menores que K_{i+1} $\quad 0 \leq i < n-1$

    mayores que K_i $\quad 0 < i \leq n-1$
  - 4. Los subárboles A_i, $0 \leq i \leq n-1$ son también árboles multicamino de búsqueda

K_1	K_2	K_3	...	K_{n-1}
A_0	A_1	...	A_{n-1}	

3. Árboles de búsqueda (II)

- Algoritmo de búsqueda
  - Para buscar un valor x el árbol, primero se mira el nodo raíz y se realiza la siguiente comparación:
  - $x < K_i$ ó $x > K_i$ ó $x = k_i$ $(1 \leq i \leq n-1)$
  - 1) En el caso que $x = K_i$, la búsqueda ya se ha completado
  - 2) Si $x < K_i$, entonces por la definición de árbol multicamino de búsqueda, x debe estar en el subárbol A_{i-1}, si éste existe en el árbol
  - 3) Si $x > K_i$, x debe estar en A_i
- Los árboles multicamino de búsqueda son útiles cuando la memoria principal es insuficiente para utilizarla como almacenamiento permanente
- En una representación enlazada de estos árboles, los punteros pueden representar direcciones de disco en lugar de direcciones de memoria principal. ¿Cuántas veces se accede a disco cuando se realiza una búsqueda? ¿Cómo se puede reducir el número de accesos a disco?

3.1. Árboles binarios de búsqueda

ESPECIFICACIÓN ALGEBRAICA (I)

- Propiedades
  - todos los elementos en el subárbol izquierdo son ≤ que la raíz,
  - todos los elementos en el subárbol derecho son ≥ que la raíz,
  - los dos subárboles son binarios de búsqueda
    - en algunas variantes no se permite la repetición de etiquetas

MODULO ARBOL_BIN_BUSQUEDA USA BOOL, ARBOLES_BINARIOS
PARAMETRO TIPO item
OPERACIONES
<, ==, >: item, item → bool
error_item( ) → item
FPARAMETRO

OPERACIONES
insertar(arbin, item) → arbin
buscar(arbin, item) → bool
borrar(arbin, item) → arbin
min(arbin) → item

3.1. Árboles binarios de búsqueda

ESPECIFICACIÓN ALGEBRAICA (II)

VAR i, d: arbin; x, y: item;
ECUACIONES
insertar(crea_arbin( ), x) =
enraizar(crea_arbin( ), x, crea_arbin( ))
si (y < x) **entonces**
insertar(enraizar(i, x, d), y) =
enraizar(insertar(i, y), x, d)
si no si (y > x) insertar(enraizar(i, x, d), y) =
enraizar(i, x, insertar(d, y)) **fsi**

buscar(crea_arbin( ), x) = FALSO
si (y < x) **entonces**
buscar(enraizar(i, x, d), y) = buscar(i, y)
si no si (y > x) **entonces**
buscar(enraizar(i, x, d), y) = buscar(d, y)
si no buscar(enraizar(i, x, d), y) = CIERTO **fsi**

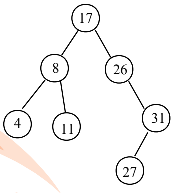
borrar(crea_arbin( ), x) = crea_arbin( )
si (y < x) **entonces**
borrar(enraizar(i, x, d), y) =
enraizar(borrar(i, y), x, d)
si no si (y > x) **entonces**
borrar(enraizar(i, x, d), y) =
enraizar(i, x, borrar(d, y)) **fsi**
si (y==x) **y** esvacio(d) **entonces**
borrar(enraizar(i, x, d), y) = i **fsi**
si (y==x) **y** esvacio(i) **entonces**
borrar(enraizar(i, x, d), y) = d **fsi**
si (y==x) **y** no esvacio(d) **y** no esvacio(i) **entonces**
borrar(enraizar(i, x, d), y) =
enraizar(i, min(d), borrar(d, min(d))) **fsi**
min(crea_arbin( )) = error_item( )
si esvacio(i) **entonces** min(enraizar(i, x, d)) = x
si no min(enraizar(i, x, d)) = min(i) **fsi**

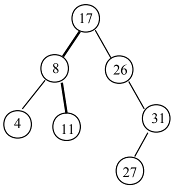
FMODULO

3.1. Árboles binarios de búsqueda

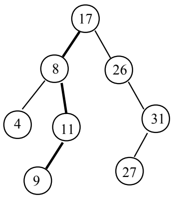
OPERACIONES BÁSICAS (I)

- Búsqueda e inserción de un elemento





búsqueda e inserción del 11



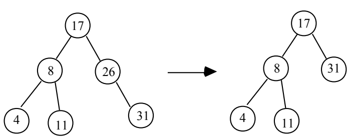
búsqueda e inserción del 9

- Recorrido en inorden: todas las etiquetas ordenadas ascendentemente
- ¿Cuál es el coste de las operaciones de búsqueda e inserción en el ABB? ¿Qué pasa si insertamos una serie de elementos ordenados en un ABB inicialmente vacío?

3.1. Árboles binarios de búsqueda

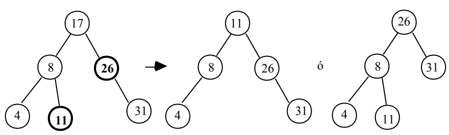
OPERACIONES BÁSICAS (II)

- Borrado de un elemento
  - El nodo donde se encuentra es una hoja
  - El nodo donde se encuentra tiene un único hijo. El nodo a eliminar es sustituido por su hijo



borrado del elemento 26

  - El nodo donde se encuentra, tiene dos hijos



borrar el elemento 17

3.1. Árboles binarios de búsqueda

EJERCICIOS *inserción* y *borrado*

1) En un árbol binario de búsqueda inicialmente vacío,
   a) Insertar los siguientes elementos: 20, 10, 30, 40, 5, 15, 50, 22, 25, 24, 26, 3, 35, 38, 39, 37
   b) Sobre el árbol resultante, realizar el borrado de: 5, 3, 30, 22, 39
      (utilizar el criterio de sustituir por el menor de la derecha)

3.1. Árboles binarios de búsqueda

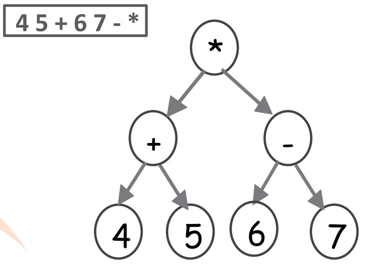
Preguntas de tipo test: Verdadero vs. Falso

- En el borrado de un elemento que se encuentre en un nodo con dos hijos no vacíos en un árbol binario de búsqueda, tenemos que intercambiar el elemento a borrar por el menor del subárbol de la izquierda o por el mayor del subárbol de la derecha
- El menor elemento en un árbol binario de búsqueda siempre se encuentra en un nodo hoja
- El coste temporal (en su peor caso) de insertar una etiqueta en un árbol binario de búsqueda es lineal respecto al número de nodos del árbol

Árboles binarios

Aplicaciones

- Evaluación de expresiones aritméticas (AB).



- Árboles de Huffman (ABB).
  - Emisor: transmisión de mensajes codificados
  - Receptor: árbol decodificador
- Treesort (ABB).
  - Ordenación de elementos utilizando un ABB
  - Si está equilibrado → Θ (n logn)