Tema 1. Introducción a los TAD.

1-. Introducción a los TADs

- TAD: tipo abstracto de datos.
  - Tipo de datos:
    ⟶ Clasifica los objetos de los programas.
  - Abstracto:
    ⟶ La manipulación de los datos solo dependen del comportamiento descrito en su especificación y es independiente de su implementación

- Especificación
  - Consiste en establecer las propiedades que lo define.
  - Definición
    - informal (lenguaje natural)
    - formal (algebraica)

- Implementación
  - Consiste en ~~representar~~ determinar una representación para los valores del tipo y en codificar sus operaciones a partir de esta representación
  - Debe ser
    - Estructurada
    - Eficiente
    - Legible

1.2-. Especificación algebraica

→ Establece las propiedades de un TAD mediante ecuaciones con variable cuantificadas.

→ Pasos
- Identificación de los objetos del TAD y sus operaciones
- Definición de la sintaxis de un TAD
- Definición de la semántica

→ Operación: Es una función que toma como parámetros cero o mas valores de diversos tipos, y produce como resultado un solo valor de otro tipo. El caso de cero parámetros representa una constante del tipo de resultado

Casos Base } F (crear / cero , ~) = ~ — resto de argumentos de F

casos recursivos } F (insertar / suc(x), ~) = ~ ← tratar la x

suc (F (x, ~))

solo si es sintácticamente correcto ← insertar

Ejemplo.

Modulo natural 1

Tipo natural

Operaciones → primera instancia posible

cero : → natural

suc : natural → natural

suma : natural natural → natural

Var

x, y : natural

Ecuaciones

suma(x, cero) = x

suma (cero, x) = x

suma (x, suce(y)) = suc (suma (x, y))

FModulo

suma (3, 2) =

suma (3, $\underbrace{\text{suc}}_{\text{suc}}$($\underbrace{\text{suc(cero)}}_{y}$)) =

suc (suma (3, suc(cero))) =

suc (suc (suma (3, cero))) =

suc(suc (3)) = 5

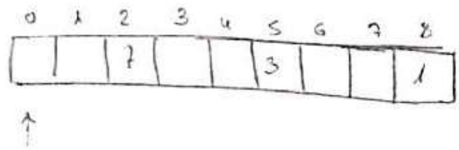
2-. Vectores.

· Un vector es un conjunto ordenado de pares <indice, valor>. Para cada indice definido dentro de un rango finito existe asociado un valor.

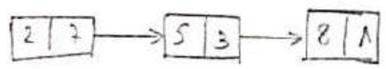
Ejemplo.

$V = \{<2,7> <5,3> <8,1>\}$

→ Representación secuencial



→ Representación enlazada



2.2-. Vectores. Especificación algebraica

Ejemplo.

$w = \{\} \longrightarrow$ crear()

$x = \{<2,7>\} \longrightarrow$ asig(crear, 2, 7)

$y = \{<2,7> <5,3>\} \longrightarrow$ asig(asig(crear, 2, 7), 5, 3)

---

recu(crear(), i) = error() // recuperar ~~un item~~ sobre un vector vacío cualquier posición siempre va a dar error item

recu(asig(v, i, x), j) // recuperar de un vector no vacío un item en la posición j

si (i == j) entonces x

si no recu(v, j) fsi

3-. Listas

· Una lista es una secuencia de cero o más elementos de un mismo tipo

· Propiedades:

- Se establece un orden secuencial escrito sobre sus elementos por la posición que ocupan dentro de la misma.
- La lista nos permite conocer cualquier elemento de la misma accediendo a su posición, algo que no podremos hacer con las pilas y con las colas.

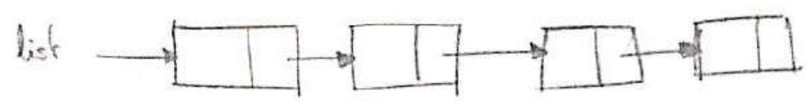
· Una lista <u>ordenada</u> es un tipo especial de lista en el que se establece una relación de orden definida entre los items

3.2-. Representación

· El TAD lista se utiliza para almacenar listas de un número <u>variable</u> de objetos.

· Representación secuencial
→ A partir de tipos base ("arrays")

· Representación enlazada
→ A partir de tipos base ("punteros a nodo")



Escaneado con CamScanner

3.3.- Especificación algebraica

Modulo lista usa Bool, natural

Parametro Tipo item, posicion

Operaciones

== (posicion, posicion) ⟶ Bool

error_item() ⟶ item

error_posicion () ⟶ posicion

fParametro

Tipo lista

Operaciones

Crear () ⟶ lista
inscabeza (lista, item) ⟶ lista

} Constructores generadores

esvacia (lista) ⟶ bool

concatenar (lista, lista) ⟶ lista

longitud (lista) ⟶ natural

primera, ultima (lista) ⟶ posición

anterior, siguiente (lista, posicion) ⟶ posición

insertar (lista, posición, item) ⟶ lista

borrar (lista, posición) ⟶ lista

obtener (lista, posición) ⟶ item

ecuaciones

esvacia (crear ()) = CIERTO

esvacia (inscabeza (L_1, x)) = FALSO

concatenar (crear () , L_1) = L_1

concatenar (L_1 , crear ()) = L_1

~~concatenar (L_1, x), L_2) inscabeza (concatenar (L_1~~

concatenar (inscabeza (L_1, x), L_2) = inscabeza (concatenar (L_1 , L_2), x)

longitud (crear ()) = 0

longitud (inscabeza (~~{~~ L_1, x)) = 1 + longitud (L_1)

primera (crear ()) = error - posicion(); ~~sig~~

~~siguiente~~ ultima(crear ()) = error_ posicion ();

~~si p != ultima~~

si esvacia (L_1) entonces

ultima (inscabeza (L_1, x)) = primera (inscabeza (L_1, x))

si no ultima (inscabeza (L_1, x)) = ultima (L_1)

anterior (L_1 , primera (L_1)) = error_ posición ();

siguiente (L_1, ultima(L_1)) = error - posición ();

si p != ultima (L_1) entonces

anterior (L_1, siguiente (L_1, p)) = p

anterior (inscabeza (L_1, x), primera (L_1)) = primera (inscabeza (L_1, x))

Operaciones

crear () → cola

encolar (cola, item) → cola

desencolar (cola) → cola

cabeza (cola) → item

esvacia (cola) → bool

ecuaciones

desencolar (crear()) = crear ()

si esvacia (c) entonces

desencolar (encolar (c, x)) = crear ()

si no desencolar (encolar (c, x)) = ~~encolar (desencolar~~

encolar (desencolar (c), x)

cabeza (crear ()) = error_item

si esvacia (c) entonces

cabeza (encolar (c, x)) = x

si no cabeza (encolar (c, x)) = cabeza (c)

esvacia (crear ()) = cierto

esvacia (encolar (c, x)) = falso

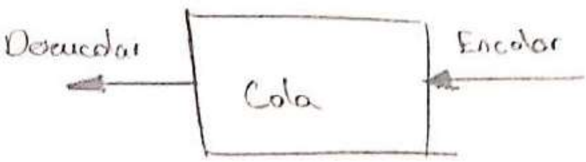
4.4-. Representación enlazada de pilas

- Representación enlazada
  - A partir de tipos base ("punteros a nodos")
  - A partir de tipos definidos por el usuario ("tlista")

- Ventajas
  - no hay definido un tamaño para la pila

5-. Colas

Una cola es otro tipo especial de lista en el cual los elementos se insertan por un extremo (fondo) y se suprimen por el otro. Listas "FIFO"



5.2-. Especificación algebraica

Modulo cola usa bool
    Parametro
        Tipo item
        Operaciones
            error() → item
    FParametro

Tipo cola

Ecuaciones

desapilar (crear ()) = crear ()

desapilar (apilar (p,e)) = p

cima (crear ()) = error ()

cima (apilar (p,e)) = e

esvacia (crear ()) = Cierto

esvacia (apilar (p,e)) = FALSO

FModulo

4.3-. Representación secuencial de pilas

- Representación secuencial (internamente un array)
  - A partir de tipos base ("array")
  - A partir de tipos definidos por el usuario ("tvector")

- Tipos de algoritmos
  - Realizando las inserciones por la primera componente. Ineficiente
  - Utilizamos un cursor que indique la posición actual del primer elemento de la pila.

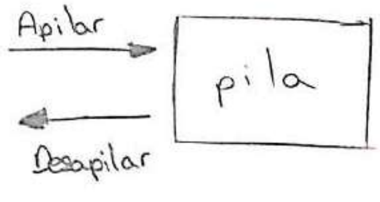
- Ventajas y desventajas
  - Desventaja: tamaño máximo de la pila
  - Ventaja: sencillez de implementación

4-. Pilas

Una pila es una lista en la que todas las inserciones y borrados se realizan en un único extremo, llamado tope o cima.

Sabemos por tanto que el ultimo elemento insertado en la pila será el primero en ser borrado de la misma.

Listas "LIFO" (Last in, First out



4.2-. Especificación algebraica

Modulo pila usa bool

Parametro

Tipo item

Operaciones

error() → item

FParametro

Tipo Pila

Operaciones

crear() → pila

apilar (pila, item) → pila

desapilar (pila) → pila

cima (pila) → item

esvacia (pila) → bool

Tema 2. La eficiencia de los algoritmos

1.- Noción de complejidad

· Calculo de complejidad : ~~cantidad~~ determinación de dos parametros o funciones de coste :

- Complejidad espacial ⟹ Cantidad de recursos
- Complejidad temporal ⟹ Cantidad de tiempo
  └→ Tiempo(A) = C + f(T) → función que depende del tamaño

→ contribución de los factores externos

⟶ Factores de complejidad temporal:

- Externos:
  - Máquina
  - El compilador
  - La experiencia del programador
- Internos:
  - Instrucciones asociadas al algoritmo

Ejemplo.

```
int ejemplo (int n) {
   int i, j, k;
   k = 1;
   for (i=0; i≤n; i++)
      for(j=i; j≤n; j++)
         k = k + k;
   return k;
}
```

$$f(\text{ejemplo}) = 1 + \overbrace{\Sigma_{i=0\dots n}}^{\boxed{1}} \Big(\overbrace{\Sigma_{j=i\dots n} 1}^{\boxed{2}}\Big)$$

$$\sum_{i=m}^{n} c = c\cdot(n-m+1)$$

$$\sum_{i=1}^{n} i = \frac{(a_1+a_n)\cdot n}{2} \quad (S.P.A)$$

$$\boxed{2}\ \sum_{j=i}^{n} 1 = 1\cdot(n-i+1)$$

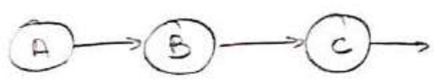
$$\boxed{1}\ \sum_{i=0}^{n} 1\cdot(n-i+1) =$$

$$\frac{(n+1+1)(n+1)}{2} = \frac{(n+2)(n+1)}{2}$$

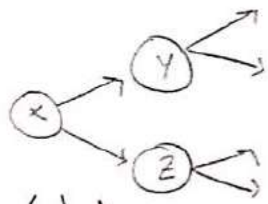
Tema 3. El tipo árbol

1-. Definiciónes generales

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



tipo lineal



árbol

· Árbol:

- un único nodo es un árbol (raiz)
- Dado n árboles $a_1, \dots, a_n$ se puede construir uno nuevo como resultado de enraizar un nuevo nodo con los n árboles.

· Árbol vacio $\Longrightarrow$ 0 nodos

· El proceso de enraizar:
- **Árboles generales**: un nº indeterminado de subárboles
- **árboles n-arios**: un nº máximo de subárboles

- un árbol n-ario con $n = 2 \Longrightarrow$ árbol binario

- La información almacenada en los nodos del árbol se denomina etiqueta

- Las hojas son arboles con un solo nodo

- Grado de un arbol es el número maximo de hijos que pueden tener subárboles

· Camino:

- es una secuencia $a_1, \dots, a_s$ de árboles tal que, $\forall i \in \{1 \dots s-1\}$, a_{i+1} es subárbol de a_i

- El número de subarboles de la secuencia menos uno, se denomina $\Longrightarrow$ longitud del camino

· A_1 es ascendente de a_2 si existe un camino $a_1, \dots, a_2$

· Los ascendientes de un árbol, se denominan $\Longrightarrow$ ascendientes propios

· Padre $\Longrightarrow$ primer ascendiente propio de un árbol
· Hijos $\Longrightarrow$ Son los primeros descendientes de un árbol
· Hermanos $\Longrightarrow$ Son subárboles con el mismo padre
· Profundidad $\Longrightarrow$ longitud del unico camino desde la raíz a dicho subárbol

· Propiedades:

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

- El máximo número de nodos en un arbol binario de altura K es $N(k) = 2^{k} - 1, K \geq 1$

2-. Arbol binario

· Recorridos:

- recorrer un árbol es visitar cada nodo del árbol una sola vez
- recorrido de un árbol es la lista de etiquetas del arbol ordenado según se visitan los nodos.
- Dos categorias básicas de recorrido:
  → recorrido en profundidad
  → recorrido en anchura o por niveles. Consiste en visitar los nodos desde la raiz hacia las hojas, y de izq. a der. dentro de cada nivel.

-. Representación secuencial y enlazada

· Representación secuencial:

- Se numeran secuencialmente los nodos del árbol hipotéticamente llena desde la raiz a las hojas por niveles y de izquierda a derecha en cada nivel. La representación secuencial se puede hacer usando un vector unidimensional:

· 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

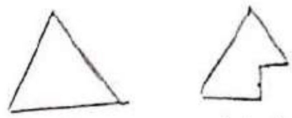
· Altura (profundidad):

- máximo nivel de los nodos de un árbol

· Árbol lleno: todo sus subárboles tienen n hijos

· Árbol completo: sus nodos corresponden a los nodos enumerados

- Todo árbol lleno es completo

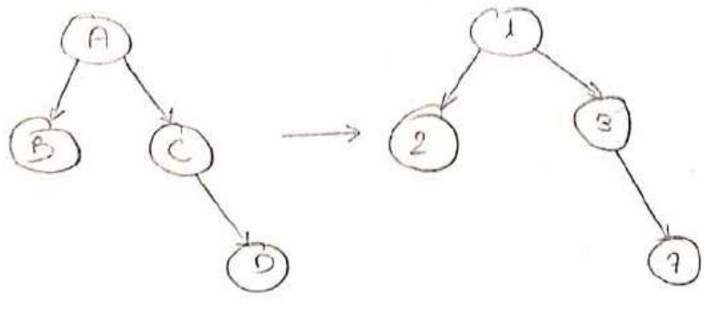


2-. Arboles binarios

- Un arbol binario es un conjunto de elementos del mismo tipo
  - Conjunto vacio ⟹ arbol vacio
  - Conjunto no vacio ⟹ existe una raíz y el resto de elementos se distribuyen en dos subconjuntos.

→ la raíz se guarda en la dirección 1

→ Si un nodo n está en la dirección i, entonces su hijo <u>izquierdo</u> estará en la <u>dirección 2i</u> y su hijo <u>derecho</u> en la <u>2i + 1</u>



A	B	C				D
1	2	3	4	5	6	7
		i			$\boxed{2i}$	$\boxed{2i+1}$

-. Otras operaciones interesantes

- Asignación (copia) entre árboles e iteradores.
    - <u>A = B</u>. hace una copia de B en A
    - <u>A = I</u>. Hace una copia sobre el árbol A, de la rama del árbol a la que apunta el iterador I.
    - <u>I = A</u>. Hace una copia sobre la rama del árbol a la que apunta el iterador I del árbol A
    - <u>I = J</u>. Sirve para inicializar el Iterador I de forma que apunte al mismo nodo al que apunta el Iterador J

- Movimientos de ramas entre árboles e iteradores:

  - Mover(A, B). Mueve el árbol B al árbol A. B se queda vacío
  - Mover(A, I). Mueve ~~del~~ la rama del árbol a la que apunta el iterador I al árbol A
  - Mover (I, A). Mueve el árbol A a la rama del árbol a la que apunta el Iterador I
  - Mover(I, J). Mueve la rama del árbol A a la que apunta el Iterador J a la rama del árbol a la que apunta el iterador I

3-. Árboles de busqueda

- Árboles de busqueda = 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}	

· Algoritmo de búsqueda

- Para buscar un valor x el árbol, primero se mira el nodo raíz y se realiza la siguiente comparación:

$$x < K_i \text{ ó } x > K_i \text{ o } x = K_i \ (1 \leq i \leq n-1)$$

1) En el caso que $x = K_i$, la busqueda ya se ha completado
2) si $x < K_i$, entonces x debe estar en el subárbol A_{i-1}, si este 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.

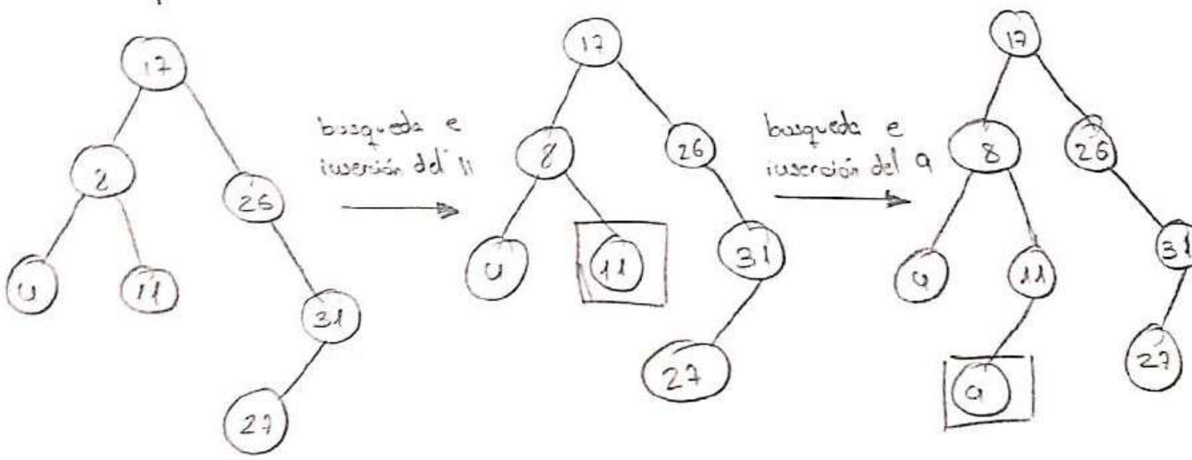
3.2-. Especificación algebraica

· 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 ~~repite~~ permite la repetición de etiquetas.

3.3-. Operaciones básicas

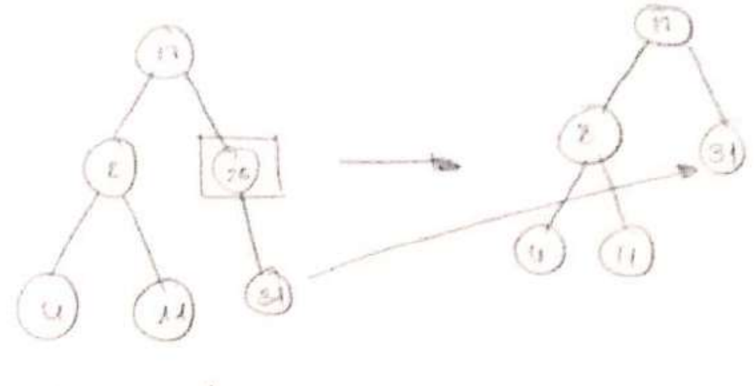
· Búsqueda e inserción de un elemento



· Recorrido inorden: todas las etiquetas ordenadas ascendentemente

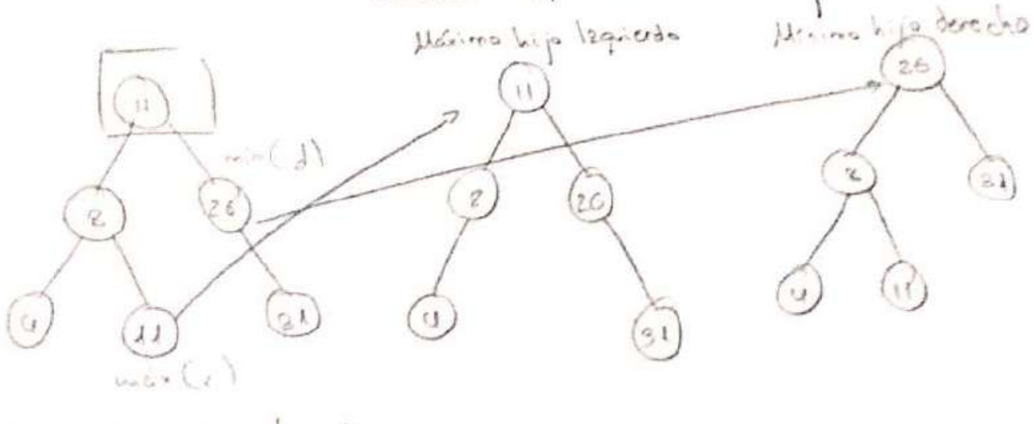
Borrado de un elemento

- El nodo donde se encuentra es una hoja
- El nodo 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



borrado del elemento 17

4-. Árboles AVL

· La eficiencia en la búsqueda de un elemento en un árbol binario de busqueda se mide en términos de:

- Número de comparaciones
- La altura del árbol

· Arbol completamente equilibrado: los elementos del árbol deben estar repartidos en igual número ~~n~~ entre el subárbol izquierdo y el derecho, de tal forma que la diferencia en ~~un~~ número de nodos sea como mucho 1.

· Problema: el mantenimiento del árbol

· Árboles AVL: desarrollado por Adelson-Velskii y Landis (1962). Los AVL son arboles balanceados con respecto a la altura de los subárboles.

· Consecuencia 1. Un árbol vacio esta equilibrado con respecto a la altura

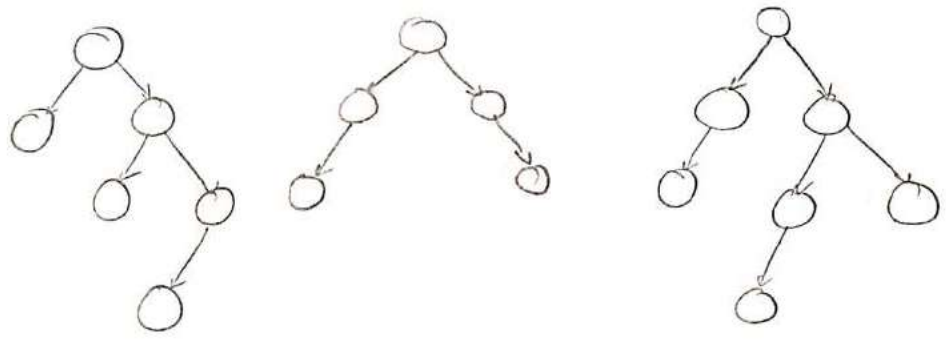
· Consecuencia 2. El árbol equilibrado óptimo será aquel que cumple:

$$n = 2^h - 1 \quad \begin{cases} n \longrightarrow \text{nº nodos} \\ h \longrightarrow \text{altura} \end{cases}$$

· Si T es un arbol binario no vacio con TL y TR como subárboles izquierdo y derecho respectivamente, entonces T está balanceado con respecto la altura si y solo si:

- TL y TR son balanceados respecto a la altura
- $|hr - hl| \leq 1$ donde hl y hr son las alturas respectivas de TR y Tl

· El factor de equilibrio {FE} (T) de un nodo T en un árbol binario se define como hr - hl. Para cualquier nodo T en un árbol AVL, se cumple FE(T) = -1, 0, 1



4.2-. Operaciones básicas. Inserción

· Representación de arboles AVL

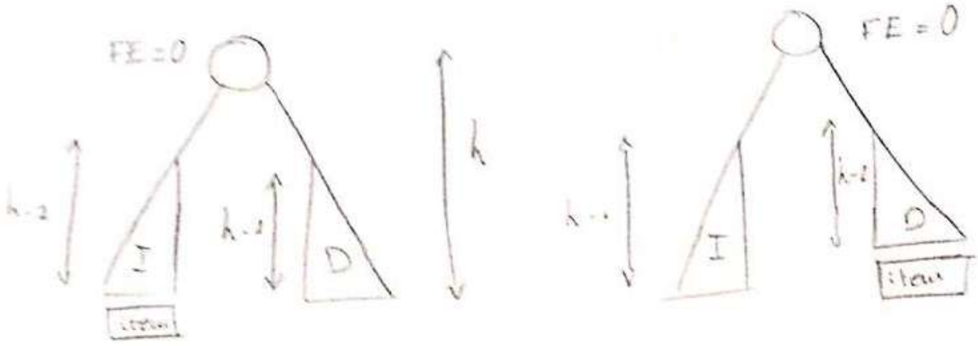
- Mantener la información sobre el equilibrio de forma implícita en la estructura del árbol.

- Atribuir a, y almacenar con, cada nodo el factor de equilibrio.

```
TNodoArb {
    Titem item;
    TArbin iz, de;
    int fe }
```

- Inserción en árboles AVL. casos:

  - Después de inserción del item, los subárboles I y D igualarán sus alturas

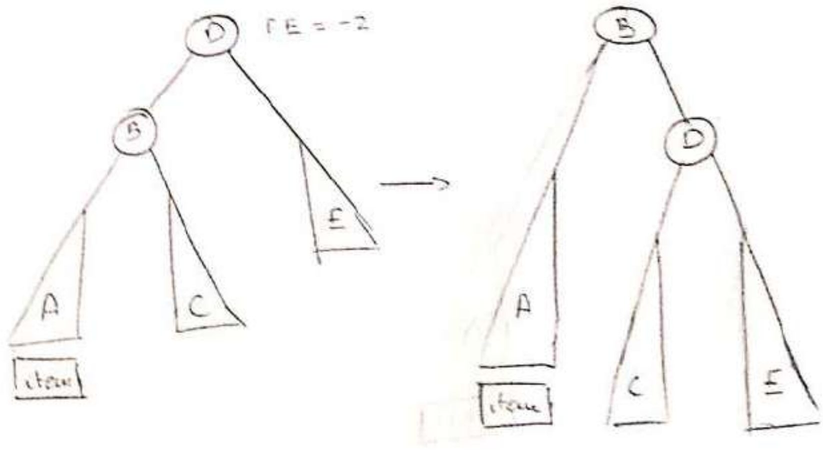


  - Despues de la inserción, I y D tendrán distinta altura, pero sin vulnerar la condición de equilibrio.

  - Si hI > hD y se realiza inserción en I, ó hI < hD y se realiza inserión en D

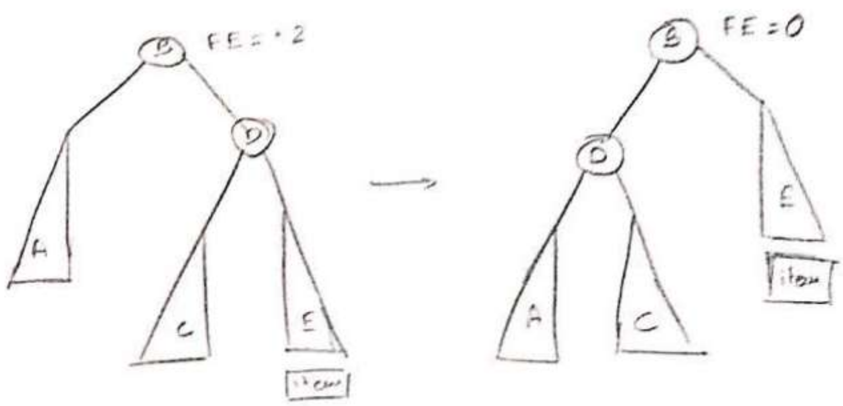
→ Formas de rotación

- Rotación II
  (-2, -1)



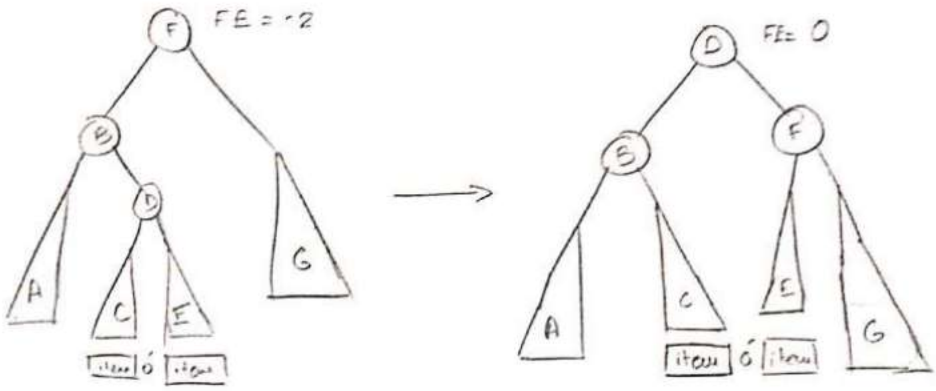
- Rotación DD

(+2, +1)



- Rotación ID

(−2, +1)



- Rotación DI

(+2, −1)