ARBOL BINARIO

ES UNA colección DE NODOS TAL Q:

- PUEDE ESTAR VACÍO.
- PUEDE ESTAR FORMADO POR UN ELEMENTO DISTINGUIDO R, LLAMADO RAÍZ Y SUS DOS SUBÁRBOLES T_1 Y T_2, DONDE LA RAÍZ DE CADA SUBÁRBOL T_i ESTÁ RELACIONADA A R POR MEDIO DE UNA ARISTA.

Descripción y terminología

- CADA NODO PUEDE TENER A LO SUMO, DOS HIJOS.
- LOS NODOS Q' NO TIENEN HIJOS, SE DENOMINAN HOJAS.
- LOS NODOS Q' COMPARTEN EL MISMO NODO PADRE, SON HERMANOS.

CAMINO; DESDE N_1 HASTA N_K, ES UNA SECUENCIA DE NODOS $N_1, N_2 \dots N_K$ TAL Q' N_i ES PADRE DE N_{i+1} PARA $1 \leq i < K$.

- LA LONGITUD DEL CAMINO ES LA CANTIDAD DE ARISTAS, ES DECIR $K-1$.
- EXISTE UNA LONGITUD CERO DE UN NODO A SÍ MISMO.
- EXISTE UN ÚNICO CAMINO DE LA RAÍZ A CADA NODO.

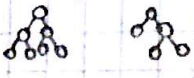
PROFUNDIDAD: DE N_i ES LA LONGITUD DEL ÚNICO CAMINO DESDE LA RAÍZ HASTA N_i.
- LA RAÍZ TIENE PROFUNDIDAD CERO.

GRADO: DE N_i ES EL NÚMERO DE HIJOS DEL NODO N_i.

ALTURA: DE N_i ES LA LONGITUD DEL CAMINO MÁS LARGO DESDE N_i HASTA UNA HOJA
- LAS HOJAS TIENEN ALTURA CERO.
- LA ALTURA DE UN ARBOL ES LA ALTURA DEL NODO RAÍZ

ANCESTRO/DESCENDIENTE: SI EXISTE UN CAMINO DESDE N_1 A N_2, SE DICE Q' N_1 ES ANCESTRO DE N_2 Y N_2 ES DESCENDIENTE DE N_1.

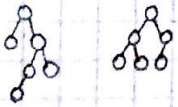
ARBOL BINARIO LLENO: DADO UN ÁRBOL BINARIO T DE ALTURA h, DIREMOS Q T ES LLENO, SI CADA NODO INTERNO ES DE GRADO DOS Y TODAS LAS HOJAS ESTÁN AL MISMO NIVEL(h). CADA NODO TIENE SOLO 2 O 0 HIJOS?



ES DECIR, RECURSIVAMENTE, T ES LLENO SI:

1 - T ES UN NODO SIMPLE (ÁRBOL BINARIO LLENO DE ALTURA 0), O
2 - T ES DE ALTURA h Y SUS SUBÁRBOLES SON LLENOS DE ALTURA $h-1$.

ARBOL BINARIO COMPLETO: DADO UN ARBOL BINARIO T DE ALTURA h, DIREMOS Q' T ES COMPLETO, SI ES LLENO DE ALTURA $h-1$ Y EL NIVEL h SE COMPLETA DE IZQ. A DERECHA.

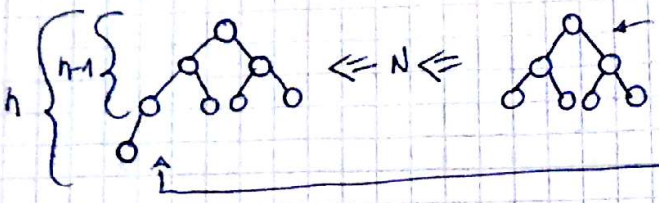


NOTA

Scanned by CamScanner

CANTIDAD DE NODOS DE UN ÁRBOL LLENO = $2^{h+1}-1$

CANTIDAD DE NODOS DE UN ÁRBOL COMPLETO VARÍA ENTRE (2^h) Y $(2^{h+1}-1)$??



$N = 2^{h+1}-1 \Rightarrow$ SI ES LLENO

SINO ES LLENO $h-1$ Y TIENE POR LO MENOS 1 NODO EN EL NIVEL h:

$2^{h-1+1}-1+1 = 2^h$

?

RECORRIDOS

PREORDEN, INORDEN, POSTORDEN.

POR NIVELES, SE PROCESAN LOS NODOS TENIENDO EN CUENTA SUS NIVELES, PRIMERO LA RAÍZ, LOS HIJOS Y LOS HIJOS DE ESTOS, ETC.

RECORRIDO PREORDEN

```
PUBLIC VOID PREORDEN () {
    IMPRIMIR (DATO);
    SI (TIENE HIJOIZQ) {
        HIJOIZQ.PREORDEN ();}
    SI (TIENE HIJODER) {
        HIJODER.PREORDEN; }
}
```

RECORRIDO POSTORDEN

```
PUBLIC VOID POSTORDEN () {
    SI (TIENE HIJOIZQ)
        HIJOIZQ.POSTORDEN()
    SI (TIENE HIJODA)
        HIJODERE.POSTORDEN()
    IMPRIMIR (DATO);
}
```

RECORRIDO INORDEN

```
PUBLIC VOID INORDEN () {
    SI (TIENE HIJOIZQ)
        HIJOIZQ. INORDEN()
    IMPRIMIR (DATO);
    SI TIENE (HIJODER)
        HIJODERE. INORDEN
}
```

RECORRIDO POR NIVELES

```
PUBLIC VOID PORNIVELES () {
    ENCOLA (RAIZ)
    MIENTRAS (COLA NO SE VACIE) {
        DESENCOLA (V);
        IMPRIMO DATO V.
        SI (TIENE HIJOIZQ)
            ENCOLA (HIJOIZQ).
        SI (TIENE HIJODER).
            ENCOLA (HIJODER)
    }
}
```

CÓMO SE LA CANTIDAD DE NIVELES?

Scanned by CamScanner

Arboles de expresión

Aplicaciones:
- En compiladores para analizar, optimizar y traducir programas.
- Evaluar expresiones algebraicas o lógicas.
  - No se necesita el uso de paréntesis.
- Traducir expresiones a notación sufija, prefija e infija.

Infija: $((a*((b*d)+c))+(e+(f*g)))$

Prefija: $+*a+*bdc+e*fg$

Postfija: $abd*c+*efg*++$

$a*(b*d+c)+(e+f*g)$

Construir a partir de una exp. postfija

Algoritmo:

```
Toma un caracter de la expresión
Mientras (existe caracter)
    Si es operando → Creo un nodo y lo apilo
    Si es operador (lo tomo como raíz de los 2 últimos nodos leídos)
        → Creo nodo R
        → Desapilo y lo agrego como Hijo Der.
        → "  "  "  "  "  Hijo Izq.
        → Apilo R
    Toma otro caracter.
Fin
```

Construir a partir de una expresión prefija ?

Algoritmo

```
ArbolExpresión (A: ArbolBin, exp: String)
    Si exp nulo → nada
    Si es un operador → - Creo nodo raíz
                        - ArbolExpresión (SubArbolIzq de A, exp (sin 1° caracter))
                        - "  "  (SubArbolDer, "  "  "  "  )
    Si es operando → Creo nodo (hoja).
```

Scanned by CamScanner

Construir de una expresión infija (infija → postfija)

a) Si es operando → se coloca en la salida
b) Si es operador → se mueve una pila, según la prioridad del operador en relación al tope

Operador con > prioridad q tope → Apila

operador con <= " " " → se desapila, pone en la salida
se vuelve a comprobar con tope de la pila

c) Si es un "(", ")" → "(" se apila

")" se desapila todo hasta el "(" incluído este.

d) Cuando se llega al final de la expresión, se desapilan todos los elementos restantes a la salida, hasta q la pila quede vacía.

Prioridad mayor a menor: ^, *, /, +, -

los "(" siempre se apilan y se desapilan sólo cuando llega ")"

((5+3)*7)+((7*4)/2)

Scanned by CamScanner

Árboles Generales

Un árbol es una colección de nodos tal q':
- Puede estar vacío.
- Puede estar formado por un nodo distinguido R, llamado raíz y un conjunto de árboles $T_1, T_2, \dots T_K, K \geq 0$ (subárboles), donde la raíz de cada subárbol T_i está conectada a R, por medio de una arista.

- **Grado del Árbol**: es el grado del nodo con mayor grado.
- **Árbol lleno**: Dado un árbol T, de grado K y altura h, decimos q T es lleno, si cada nodo interno tiene grado K y todas las hojas están al mismo nivel (h).

Es decir recursivamente, T es lleno si:
- T- es un nodo simple (árbol binario de altura cero) ó
- T- es un árbol de altura h, y todos sus subárboles llenos hasta la altura h-1.

Cantidad de nodos en un árbol lleno $N = (K^{h+1} - 1)/(K-1)$?

- **Árbol completo**: Dado un árbol T, de grado K y altura h, T es completo si es lleno hasta altura h-1 y se completa de izquierda a derecha en el nivel h.

Cantidad de nodos en un árbol completo ~~entre $(K^h + K - 2)/(K-1)$ y~~ $(K^{h+1} - 1)/(K-1)$

Recorridos

Preorden - Inorden - Posorden - Niveles.

Recorrido Preorden

```
Public void Preorden ( ) {
    Imprimir (Dato)
    Obtener lista Hijos
    Mientras (tenga hijos) {
        Hijo ← obtener hijo;
        Hijo. Preorden();
    }
}
```

Recorrido por Niveles

```
Public void porNiveles() {
    encola(raiz);
    mientras la cola no se vacíe {
        V ← desencolar();
        Imprimir (dato de V);
        Para cada hijo de V
        encolar (hijo);
    }
}
```

Scanned by CamScanner

```
Public void RenNiveles (int K) {
q: cola ; NroNiv = 0 ; CantNodos = 0;

   Encolar Raíz en q;
   Encolar Null en q;

   Mientras (la cola ¡Vacía)
      Desencolar V de q;
      Si (Dato de V = Null y q no está Vacía)
         Encolo Null en q;
         NroNiv++;
      Sino
         Si (Dato de V no es Null)
            Si (NroNiv == K)
               Mientras (el Dato de V no es Null)
                  CantNodos++;
                  Desencolar V de q;
            Sino para cada hijo W de V
                  Encolo W en q;

Retorn CantNodos
```

Scanned by CamScanner

ARBOL BINARIO DE BUSQUEDA

UN ARBOL BINARIO DE BUSQUEDA ES UNA COLECCIÓN DE NODOS CONTENIENDO CLAVES, Q' DEBEN CUMPLIR CON UNA PROPIEDAD ESTRUCTURAL Y UNA DE ORDEN.

LA PROPIEDAD ESTRUCTURAL: ES UN ARBOL BINARIO

LA PROPIEDAD DE ORDEN ES LA SIGUIENTE: PARA CADA NODO N DEL ÁRBOL, SE CUMPLE Q' TODOS LOS NODOS UBICADOS EN EL SUBARBOL IZQUIERDO CONTIENEN CLAVES MENORES Q' LA DE LA CLAVE DEL NODO N Y LOS NODOS UBICADOS EN EL SUBÁRBOL DERECHO CONTIENEN CLAVES MAYORES Q' LA CLAVE DEL NODO N

BUSCAR UN ELEMENTO

```
PUBLIC BOOLEAN EXISTE (a: ARBOL, elemento: K) {
    IF (a == NULL) RETURN FALSE
    IF (a.DATO == elemento) RETURN TRUE
    IF (a.DATO > elemento)
        RETURN EXISTE (a.HIJOIZQ, elemento).
    IF (a.DATO < ELEMENTO)
        RETURN EXISTE (a.HIJO DERECHO, elemento)
}
```

INSERTAR UN ELEMENTO

```
Public void
```

Scanned by CamScanner

Arboles AVL

Es un árbol binario de búsqueda q' cumple con la condición de este enunciado.

La propiedad de balanceo q' cumple dice: para cada nodo del árbol, la diferencia de alturas entre el subárbol izquierdo y el subárbol derecho es a lo sumo 1.

Características

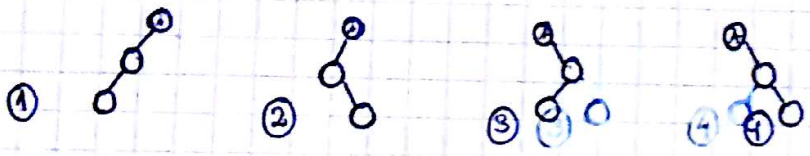
- La propiedad de balanceo es fácil de mantener y garantiza q' la altura de árbol sea de $O(\log n)$
- En cada nodo del árbol se guarda información de la altura.
- La altura del árbol vacío es -1.

Desbalanceo árbol

- Se recorre el camino de búsqueda en orden inverso.
- Se controla el equilibrio y balanceo de cada nodo.
- Si está desbalanceado se realiza una modificación simple: Rotación.
- Después de balancear el nodo la inserción termina.
- Este proceso puede llegar a la raíz.

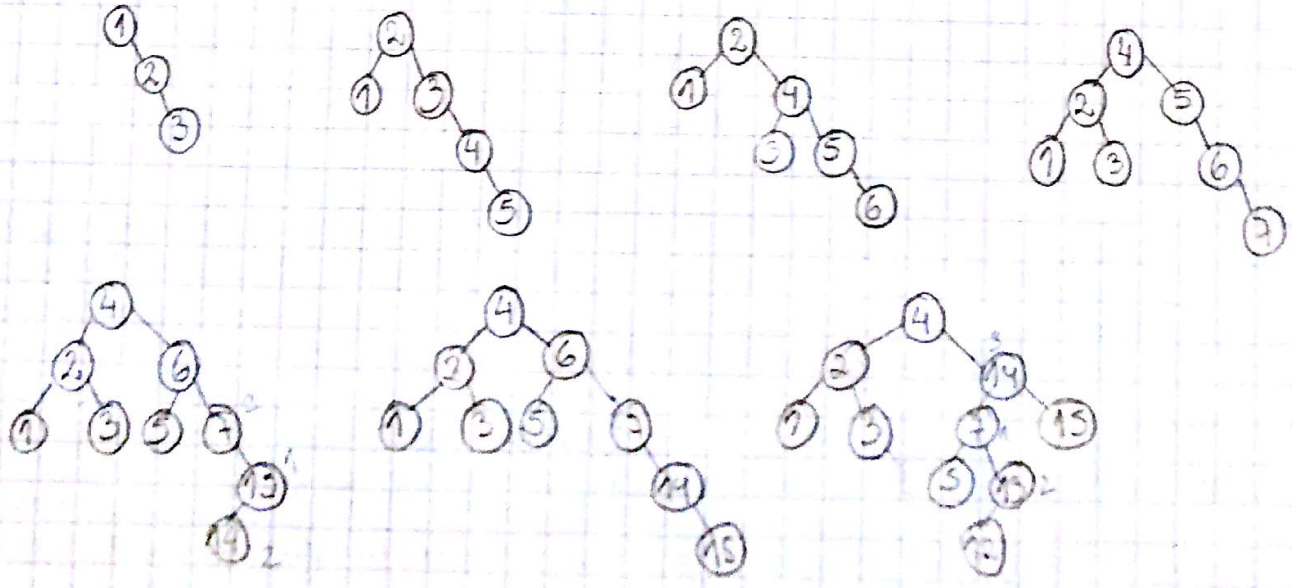
La rotación es una modificación simple de la estructura del árbol q' restaura la propiedad de balanceo, preservando el orden de los elementos

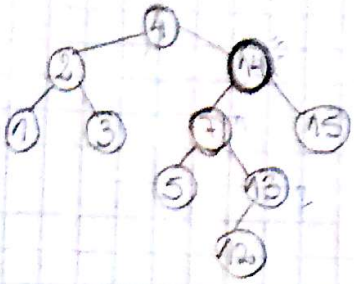
4 casos posibles de desbalanceo

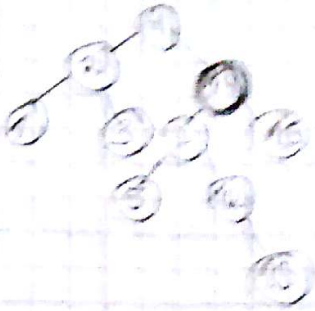


2 clases de rotaciones

Rotación simple: Casos 1 y 4: inserción nodo externo.
Rotación doble: casos 2, 3 " " interno.







Scanned by CamScanner

Cola de Prioridades

Es una estructura de datos q' permite al menos 2 operaciones:
- INSERT, inserta un elemento en la estructura
- DeleteMIN, encuentra, borra y elimina el elemento mínimo.

Heap Binario

Es una implementación de colas de prioridades q' no usa punteros y permite implementar ambas operaciones con $\Theta(\log n)$ en el peor de los casos.

Cumple con dos propiedades:
- Propiedad estructural, es un árbol binario completo.
  El nro nodos de un árbol binario completo de altura h, satisface $2^h \leq n \leq (2^{h+1}-1)$
  La altura h del árbol es de $\Theta(\log n)$.

  Dado q' un árbol binario completo es una estructura de datos ideal, puede almacenarse en un arreglo, tal q':
  - La raíz se encuentra en la posición 1.
  - Para cada elemento q' está en la posición i:
    - El hijo izq. está en la posición $i*2$.
    - El hijo der. está en la posición $i*2+1$.
    - El padre está en la posición $\lfloor i/2 \rfloor$.

- Propiedad de orden
  → MIN HEAP
    - El elemento mínimo está almacenado en la raíz
    - El dato almacenado en cada nodo, es menor o igual al de sus hijos.
  → MAX HEAP
    - Se usa en propiedad inversa

Implementación de HEAP

Una HEAP consta de:
- Un arreglo donde tengo los datos
- Un valor q' me indica el nro de elementos almacenados.

Ventaja
- No se necesitan punteros
- Fácil implementación de las operaciones

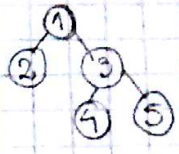
Operación INSERT
- Se inserta como último ítem de la HEAP.
- La propiedad de la HEAP puede ser violada.
- Se debe hacer un FILTRADO HACIA ARRIBA para restaurar la propiedad

FILTRADO HACIA ARRIBA (Recolocar up)
- Restaura la propiedad del orden intercambiando K a lo largo del camino hacia arriba desde el nivel de la inserción.
- El filtrado termina cuando K alcanza la raíz o un nodo cuyo padre tiene una clave menor.
- Ya q' el algoritmo recorre la altura de la HEAP tiene $\Theta(\log n)$ intercambios.

Scanned by CamScanner

```
Insert (Heap H, Comparable x) {
    h.tamaño = h.tamaño+1;
    h.dato[h.tamaño] = x
    Percolate-up (h, h.tamaño);
}

Percolate-up (Heap h, Integer i) {
    temp = h.dato[i]
    while (i/2 > 0 AND h.dato[i/2] > temp) {
                  PADRE
```



Scanned by CamScanner