18.4

18.1.1 -

El Futuro se moldea

Árbol binario de búsqueda

Recorrido a orden de los elementos en forma ordenada

- Opción alternativa a la lista enlazada
- Con entradas de gran tamaño las listas enlazadas no son efectivas
- ABB: extensión del algoritmo de búsqueda binaria permite inserciones y eliminaciones
- Tiempo de ge aprox $O(\log N)$. Peor caso $O(N)$

- Satisface la propiedad de la búsqueda ordenada.

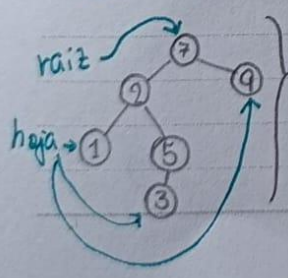
claves subárbol izq < x < claves subárbol derecho } Árboles binarios de búsqueda

- La propiedad que los define, implica que todos los elementos del árbol pueden ordenarse de forma consistente (recorrido en orden simétrico → elementos ordenados ascendentemente).

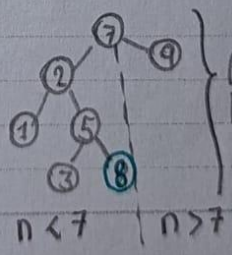
→ Esta misma no permite elementos duplicados.
elementos con = clave en est. secundaria
o
Si son ==, tener sólo 1 y contrepeticiones

Operaciones

- **buscar**: comenzar x la raíz y desplazarse x las ramas dependiendo de las comparaciones
- **buscarMin**: desde la raíz, izq hasta el último
- **buscarMax**: " , der " " " . } coste logarítmico O(n) en peor caso



Árbol binario de búsqueda



Árbol Binario

n < 7 | n > 7

El futuro se moldea

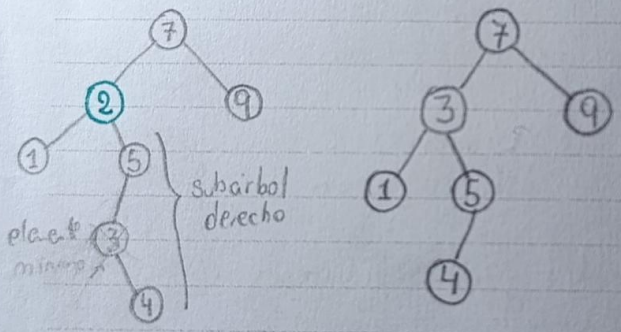
- **eliminar**: al encontrar el nodo → puede desconectar varias partes
  ↳ Árbol sin mucha profundidad

+ profundidad = + tiempo de ejecución

Si nodo a eliminar es → hoja: se elimina simplemente

→ tiene 1 hijo: saltar al nodo eliminado

→ 2 hijos: reemplar el elemento con el menor elemento del subárbol derecho y eliminar el nodo del menor elemento



CAP 17 (241 pdf)

Árboles

Se usan en: diseño de compiladores
procesamiento de textos
algoritmos de búsqueda

El Futuro se moldea

17.1 Árboles Generales

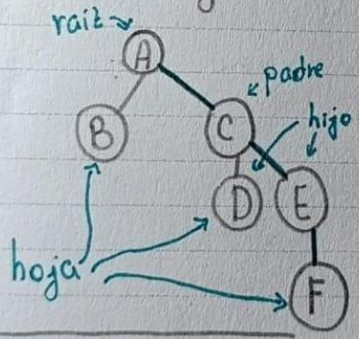
def 2 formas: recursiva : algoritmos simples para manipular árboles
no recursiva : + directa

17.1.1

Árbol - Conjunto de nodos y aristas que conectan pares de nodos.

Prop de un Árbol con raíz → 1 nodo como raíz
→ P(padre), c(hijo) raíz es sólo p
→ longitud de camino - camino único desde raíz hasta tal nodo con x número de aristas.

Nodos s/hijos → hojas



longitud de camino A-F = 3

N nodos : N-1 Aristas

profundidad de un nodo de un árbol : longitud raiz ⇆ nodo

altura : nodo ⇆ hoja mas profunda bajo él

profundida de la raiz : 0

Altura de C : 2 | tamaño de C : 3

hermanos : nodos con = padre

tamaño de un nodo = número de descendientes + Si mismo

altura de un árbol: altura de su raiz

tamaño de un árbol = tamaño de su raiz

El futuro se moldea

Un Árbol es → vacío
o
→ tiene una raíz y → 0
o
→ más subárboles

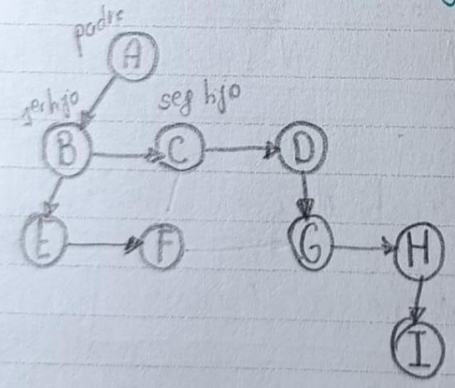
cada nodo de un árbol define un subarbol.

17.1.2 Implementación

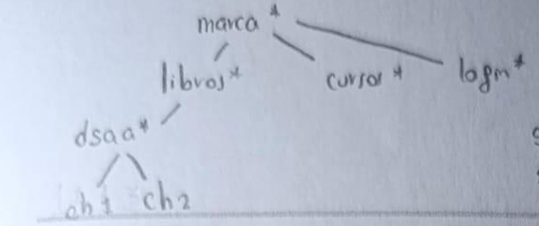
En cada nodo 1 referencia a cada hijo y sus datos
número de hijos no se conoce de antemano: mucho espacio necesario

↳ Guardar a los hijos de c/Nodo en una lista enlazada
↳ c/Nodo guard 2 referencias → hijo más a la izq (no hoja)
→ Su hermano de la der (no el + a la d

método primer hijo/siguiente hermano



17.1.3 Sist de ficheros [244] Dibujo 17.1



El Futuro se moldea

Si comienza en la raíz: nombre de camino completo
sino: nombre de camino relativo al directorio actual

marca/libros/dsaa/ch1

↳ Cada / indica 1 arista → Produciendo un nombre de camino

Pseudocódigo para listar el contenido de un directorio y sus subdirectorios en un sistema jerárquico de ficheros

```
void listar (int prof=0) // profundidad 0
{
    imprimirNombre(prof);
    if (esDirectorio())
        for cada fichero c en este directorio (para cada hijo)
            c.listar(prof+1);
```

recorrido en preorden

* No es legal especificar un valor por defecto en Java

↳ Primero se hace el trabajo en el nodo antes que en los hijos
Coste de tiempo lineal

Otro método: recorrido en postorden
↳ se hace el trabajo en un nodo después de procesar a sus hijos

Número tot de bloques = todos los hijos + raíz

↓

17.8 tamaño total - rutina postorden

Implementacion en Java

clase File en Java → Recorre jerarquías de directorios

1) getName
2) getPath
3) isDirectory
4) lenght - tamaño en bytes
5) list - devuelve un String de valores con los nombres de los ficheros en el directorio

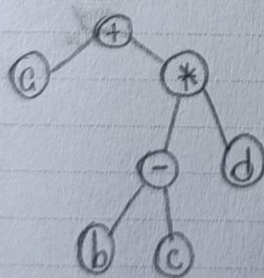
(ficheros → no pueden tener hijos)

17.2 Árboles Binarios

Ningún nodo puede tener más de 2 hijos → izq y der

Un Árbol Binario es → vacío
o
→ tiene 1 raíz → hijo izq (árbol) } pueden ser vacíos
→ hijo der

Árboles de expresión → diseño de compiladores
→ hojas son operandos (variables)
→ los otros nodos son operadores
→ Si todos los operadores son binarios ⇒ * el árbol es binario.



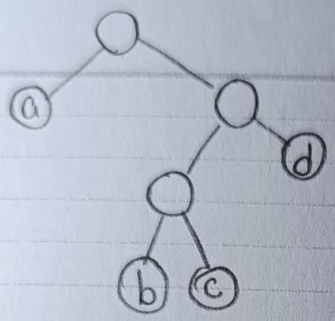
(a +((b-c) * d))

Árbol de codificación de Huffman

→ cada símbolo de alfabeto en una hoja

→ bajar x izq = 0
x der = 1

El Futuro se moldea



b = 100
c = 101
a = 0
d = 11

Árboles binarios son soporte de árboles binarios de búsqueda - inserciones, acceso a elem, colas de prioridad, elim.

17.3 Árboles y Recursión

método duplicar de NodoBinario, método recursivo:

1. creamos un nodo nuevo con el mismo dato que el nodo raíz actual
2. llamo recursivamente duplicar para árbol izq y der ...

```
NodoBinario duplicar()
{ NodoBinario raiz = new NodoBinario(dato);
  if (izq != null)
      raiz.izq = izq.duplicar
  if (der != null)
      raiz.der = der.duplicar
  return raiz;
}
```

```
static int tamanyo (NodoBinario raiz)
{ if (raiz == null)
      return 0;
  else
      return tamanyo(raiz.izq) + tamanyo(raiz.der) + 1;
```

```
static int altura (NodoBinario raiz)
{ if (a == null)
    return -1;
  else
      return 1 + Math.max (altura(a.izq), altura(a.der))
```

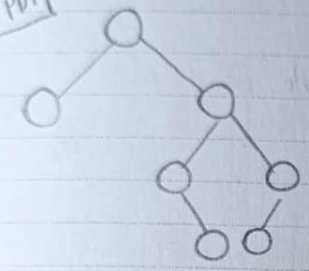
17.4 Recorrido de árboles: clases iteradoras

O(n)
- Preorden - nodo → sus hijos (duplicar)
  nodo → subarbol izq → subarbol der
- Postorden - hijos → nodo (tamaño, altura)
  subarbol izq → subarbol der → nodo

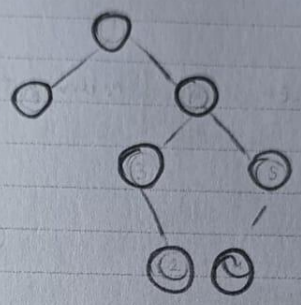
El Futuro se moldea

- Recorrido en orden Simétrico - hijo izq → nodo → hijo der
  ↳ usado para formar la expresión algebraica de un árbol de expresión

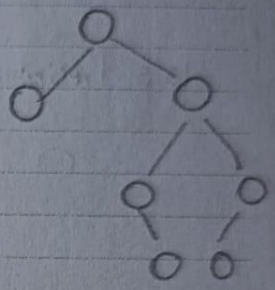
Preorden



Postorden



Simetrico



949 PDF

No es necesaria la recursión, podemos implementar nuestra propia pila → programa + rápido

17.4.1 Recorrido en postorden

usa una pila para almacenar el estado actual

3 situaciones → llamada recursiva hijo izq } curso de recorrido
↳ " " " der } c/nodo se coloca en
↳ procesar nodo actual } la pila 3 veces

Pila contiene nodos recorridos pero no tratados completamente

↓

al desapilarse la 3ª vez - nodo actual será visitado

Fig 17.25 pag 251 PDF

17.4.2 Recorrido en Orden Simétrico

Análogo al recorrido postorden pero un nodo se declara visitado cuando se desapila x 2ª vez antes de concluir a pila su hijo der (si existe)

17.4.3 Recorrido en Preorden

Análogo al recorrido post orden pero un nodo se declara visitado la primera vez que es desapilado.
Iter apila al hijo der y luego al hijo izq

queremos procesar primero el izq

clase PreOrden se deriva directamente de ArbdIter

18.4 Árboles AVL

Árbol AVL - primer tipo de árbol binario de búsqueda bien equilibrado
(Adelson - Velskii y Landis)

El Futuro se moldea

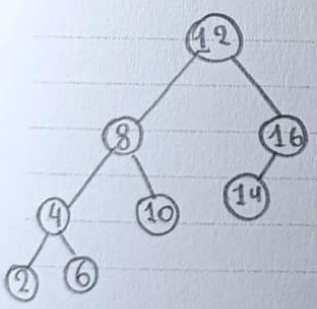
- Se tratan de árboles binarios de búsqueda con una condición adicional de equilibrio.
- Profundidad del Árbol siempre $O(\log N)$.
  - ↳ exigir que los subárboles izq y der tengan la misma profundidad recursivamente
    - ↳ demasiado restrictiva (es complicado al insertar elementos nuevos)

* Noción de equilibrio + débil pero + fuerte en profundidad logarítmica.

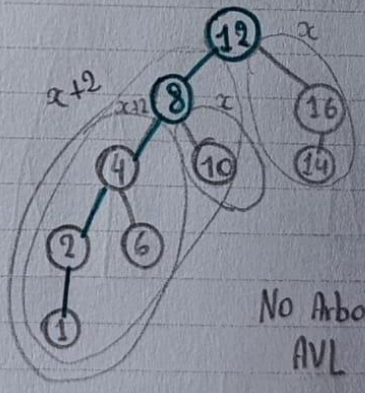
18.4.1 Propiedades

Def: Árboles binarios de búsqueda con una prop adicional de equilibrio, altura hijo der ± 1 altura hijo der.

-1: Árbol vacío



Árbol AVL



⊙ nodos no equilibrados

No Árbol AVL

- Tamaño del menor árbol AVL de altura H

$$S_H = S_{H-1} + S_{H-2} + 1$$

- Todas las operaciones de búsqueda de un AAVL tiene cota logarítmica en el peor caso.

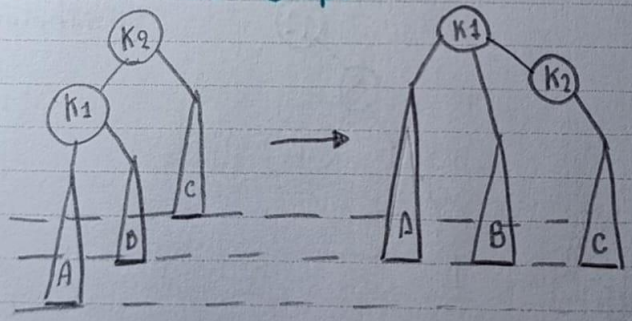
- al insertar o eliminar se puede destruir el equilibrio, por lo que este se debe recuperar antes de finalizar la operación.

Existen 2 casos básicos de incumplimiento de la propiedad, aunque al programar tomamos en cuenta 4. 267 PDF

Caso 1:
inserción izq → izq o der → der
Solu → Rotación simple de árbol
↳ Intercambia papeles de padre e hijo

Caso 2:
inserción se produce dentro
Solu → Rotación doble

18.4.2 Rotación Simple



CÓDIGO: 268 pdf

der del hijo → izq del padre

18.4.3 Rotación doble

Una rotación simple entre el hijo de x y su nieto, seguida de una rotación simple de x y su nuevo hijo

El Futuro se moldea

18.4.4 Resumen de la inserción en un árbol AVL

forma + sencilla: emplear un algoritmo recursivo

Si hay un desequilibrio realizar una rotación doble o simple.

es + eficiente almacenar en el nodo el resultado de la comparación en lugar de su altura
↳ evita cálculos repetitivos sobre el equilibrio