Análisis y diseño de software

dit
UPM

Tema 2: Algoritmos++ /diccionarios

José A. Mañas

http://jungla.dit.upm.es/~pepe/doc/adsw/index.html

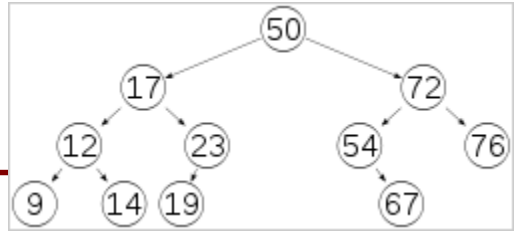
8.3.2018

ETSIT UPM

otros algoritmos

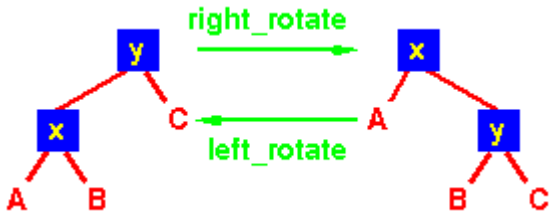
- AVL
  - BST balanceados
- red-black
  - BST balanceados “razonables”
- Tries, Radix, Patricia
- B-trees
- Filtros de Bloom
  - hash para filtros ràpidos

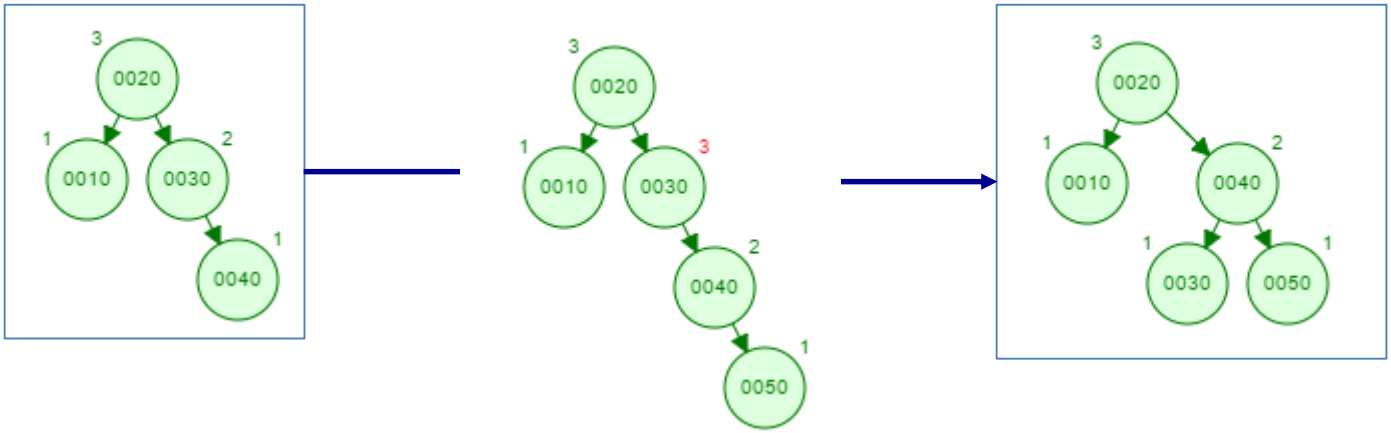
AVL-Trees



- Son árboles binarios de búsqueda
  - el árbol siempre está lo mejor equilibrado posible la profundidad de las dos ramas de hijos de un padre nunca difiere en más de 1 altura
  - el algoritmo de inserción puede re-equilibrar el árbol
  - el algoritmo de borrado puede re-equilibrar el árbol
  - el algoritmo de búsqueda siempre es O(log n)

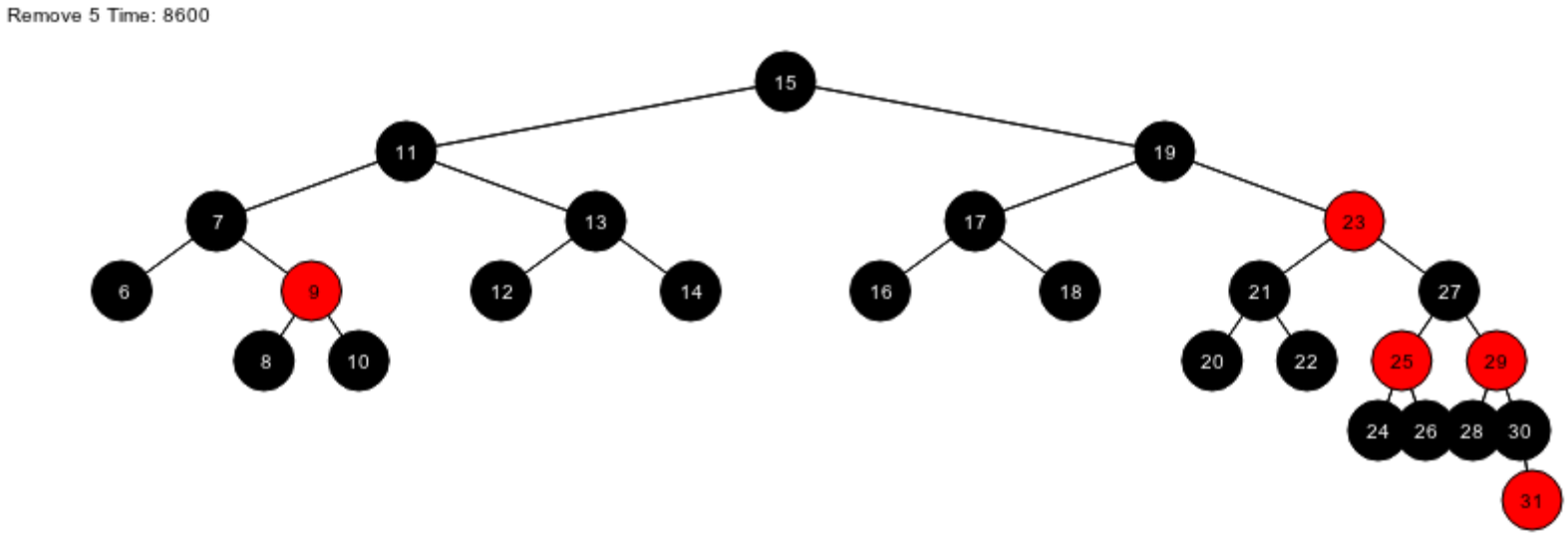
rotaciones


right_rotate
left_rotate
y
x
x
y
A
B
C
A
B
C


3
0020
1
0010
2
0030
1
0040
3
0020
1
0010
3
0030
2
0040
1
0050
3
0020
1
0010
2
0040
1
0030
1
0050

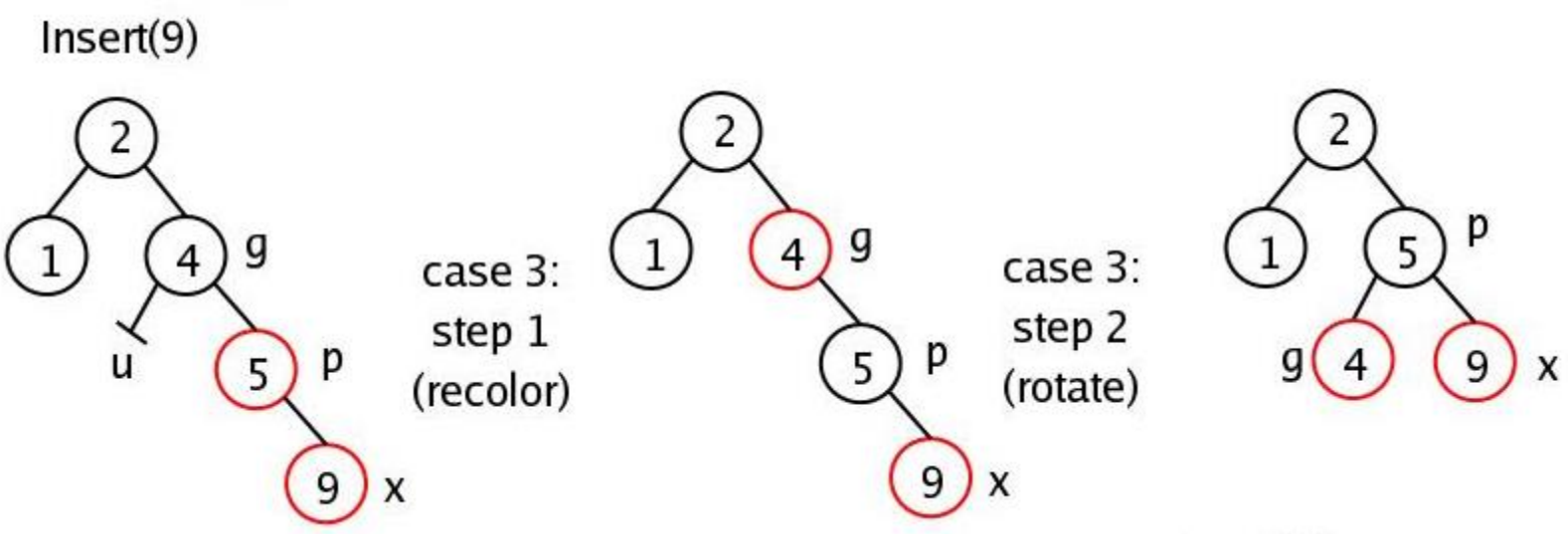
Red-Black Trees

- Otros árboles que se reequilibran pero de forma laxa
  - get, put y remove están garantizados O(log n)
- java: TreeMap



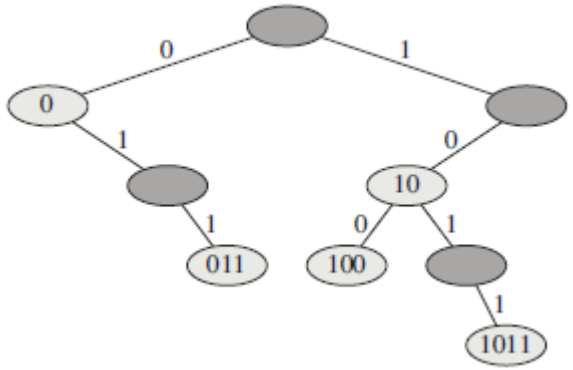
Red-Black Trees

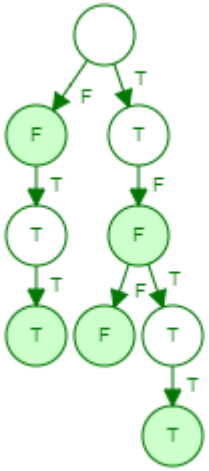
- Se colorean los nodos de forma que
  - la profundidad medida en negro es siempre la misma, desde cualquier nodo
  - los nodos rojos siempre están rodeados de negros



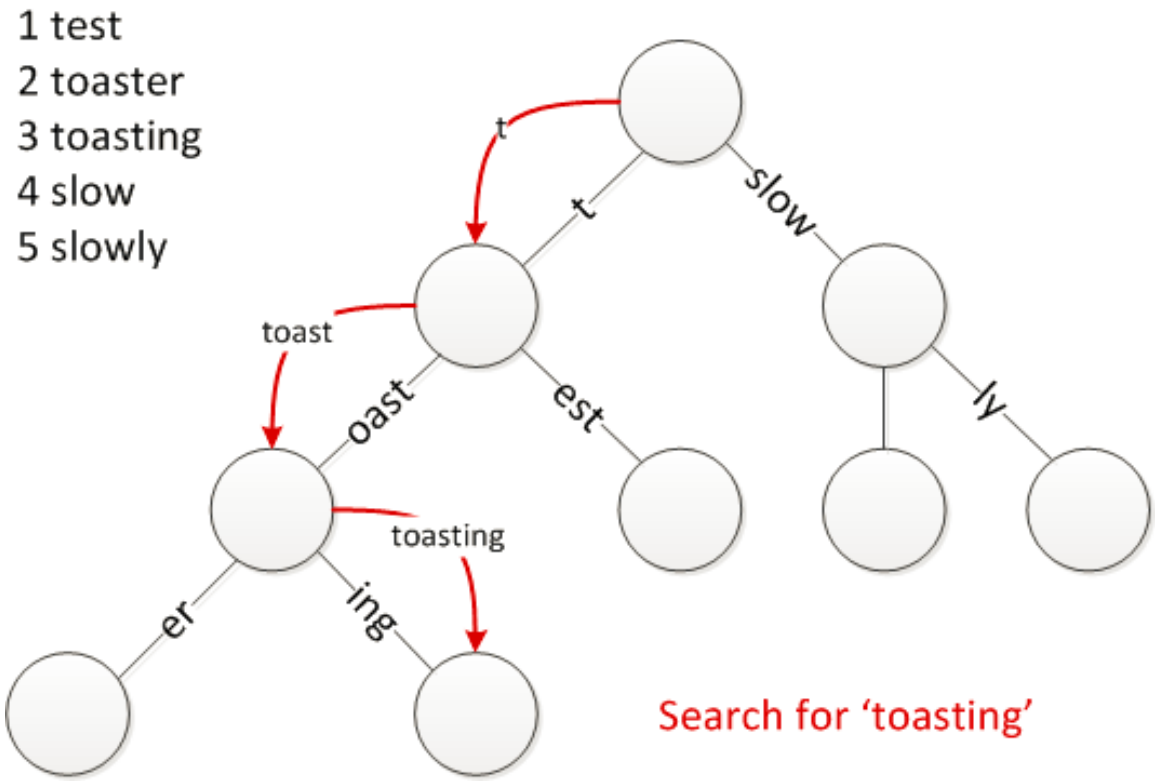
Radix, Trie, Patricia

- datos: 1011, 10, 011, 100, 0
- el árbol es independiente del orden de inserciones
- la profundidad es la de la cadena más larga
- ej. tablas routing ip





Radix, Trie, Patricia



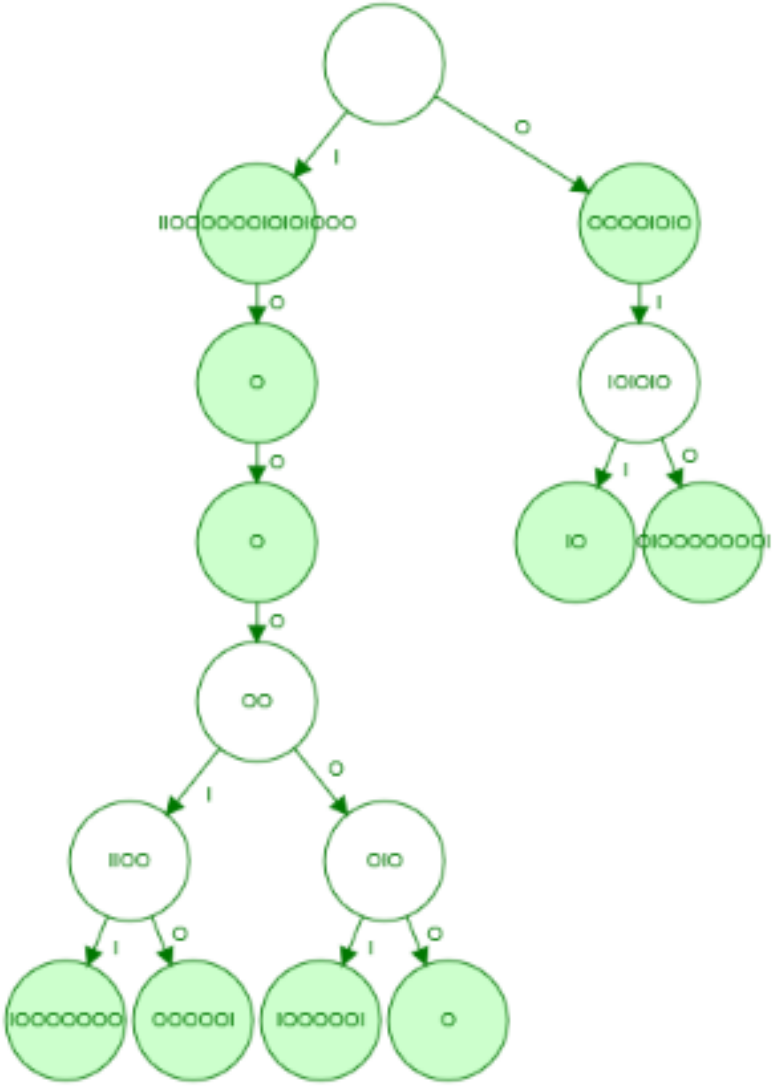
IP routing tables

- the problem is to find the longest prefix
  - ip address -> next gateway
  - ip 4: 32 bits | ip 6: 128 bits
- https://vincent.bernat.im/en/blog/2017-ipv4-route-lookup-linux

10.0.0.0/8	00001010
192.168.0.0/16	11000000 10101000
192.168.0.0/17	11000000 10101000 0
192.168.0.0/18	11000000 10101000 00
192.168.4.0/24	11000000 10101000 00000100
192.168.5.4/30	11000000 10101000 00000101 000001
192.168.12.4/30	11000000 10101000 00001100 000001
192.168.12.128/32	11000000 10101000 00001100 10000000
10.169.1.0/24	00001010 10101001 00000001
10.170.0.0/16	00001010 10101010

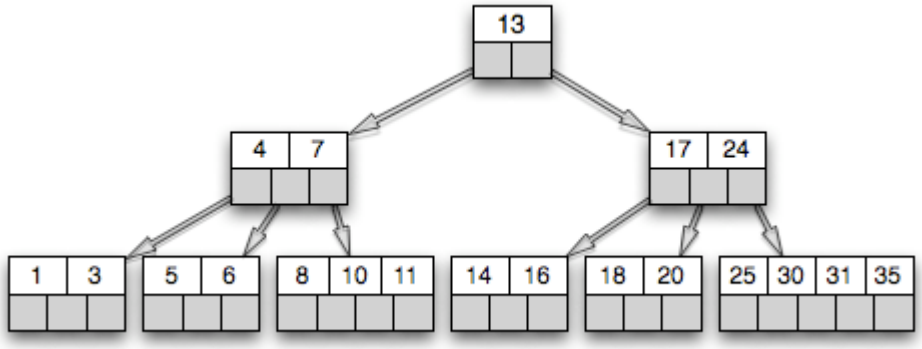
IP routing tables

00001010
11000000 10101000
11000000 10101000 0
11000000 10101000 00
11000000 10101000 00000100
11000000 10101000 00000101 000001
11000000 10101000 00001100 000001
11000000 10101000 00001100 10000000
00001010 10101001 00000001
00001010 10101010



B-trees

- Son árboles con varios valores e hijos por nodo
  - generalización de un BST
  - permite procesar bloques de información (pequeños arrays) optimizando el acceso
  - bases de datos
  - lecturas de discos



Bloom filter

- filtro probabilístico de búsqueda de contenido
  - si da positivo, probablemente está
  - si da negativo, es seguro que no está
- dada una String, tenemos varios hash(s)
- para guardar
  - marcamos (true) los bits indicados por los hash
- para buscar
  - se recalculan los hash
  - si están todos los bits marcados, TRUE
    - puede ser un falso positivo debido a colisiones múltiples
  - si algún bit no está marcado, FALSE

Bloom filter

```
private static final int N_BITS = 15;
private static final int N_HASH = 3;

   private boolean[] table = new boolean[N_BITS];

   public void add(String word) {
      for (int i = 0; i < N_HASH; i++) {
         int h = hi(i, word);
         table[h] = true;
      }
   }

   public boolean search(String word) {
      for (int i = 0; i < N_HASH; i++) {
         int h = hi(i, word);
         if (table[h] == false)
            return false;
      }
      return true;
   }
```

```
private int hi(int i, String word) {
      int h1 = word.hashCode();
      int h2 = h1 >> 16;
      int h = h1 + i * h2;
      if (h < 0)
         h = -h;
      return h % N_BITS;
   }
```

Bloom filter

```
private void test() {
      add("alfa");
      add("beta");
      add("gamma");
      add("delta");

      System.out.println(search("alfa"));
      System.out.println(search("epsilon"));
      System.out.println(search("zeta"));
  }
```

```
True
False
True
```