Estructuras de datos: Árboles binarios de búsqueda

Algoritmos

Facultad de Informática
Universidad de A Coruña

Santiago Jorge
santiago.jorge@udc.es

Referencias bibliográficas

- M. A. Weiss. Árboles. En *Estructuras de datos y algoritmos*, capítulo 4, páginas 93–154. Addison-Wesley Iberoamericana, 1995.
- R. Peña Marí. Implementación de estructuras de datos. En *Diseño de Programas. Formalismo y abstracción*, capítulo 7, páginas 257–290. Prentice Hall, segunda edición, 1998.
- G. Brassard y T. Bratley. Estructura de datos. En *Fundamentos de algoritmia*, capítulo 5, páginas 167–210. Prentice Hall, 1997.

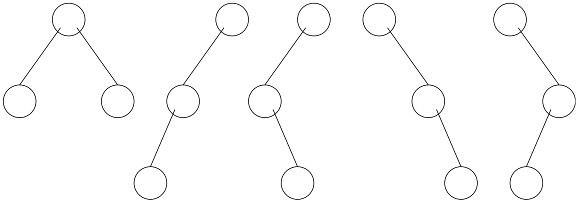
Preliminares

- El camino de un nodo n_1 a otro n_k es la secuencia de nodos $n_1, n_2, \ldots, n_k$ tal que n_i es el padre de n_{i+1}.
- La profundidad de un nodo n es la longitud del camino entre la raíz y n.
  - La raíz tiene profundidad cero.
- Para un árbol binario de búsqueda, el valor medio de la profundidad es $O(\log n)$.
  - Si la inserción en un ABB no es aleatoria, el tiempo computacional aumenta.
    - Para mantener el equilibrio: Árboles AVL, Splay Trees, ...
- La altura de n es el camino más largo de n a una hoja.
  - La altura de un árbol es la altura de la raíz.

Valor medio de la profundidad de un nodo $O(\log n)$

- Suposición: Las claves han sido insertadas aleatoriamente.
- La prueba es por inducción.
  - Ej.: posibles estructuras de un árbol con 3 nodos

Posibles estructuras de un árbol de 3 nodos

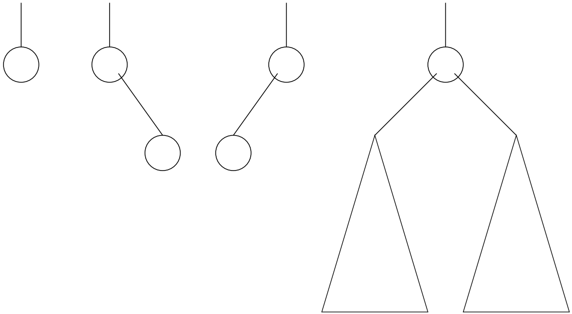


Operaciones básicas

- `Buscar`: devuelve la posición del nodo con la clave `x`.
- `Insertar`: coloca la clave `x`. Si ya estuviese,
  no se hace nada (o se "actualiza" algo).
- `Eliminar`: borra la clave `x`.
  - Si `x` está en una hoja, se elimina de inmediato.
  - Si el nodo tiene un hijo, se ajusta un apuntador antes de eliminarlo.
  - Si el nodo tiene dos hijos, se sustituye `x` por la clave más pequeña, `w`, del subárbol derecho.
    - A continuación se elimina en el subárbol derecho el nodo con `w` (que no tiene hijo izquierdo)

Posibilidades al borrar un nodo en un ABB

Alternativas al borrar un nodo en un ABB

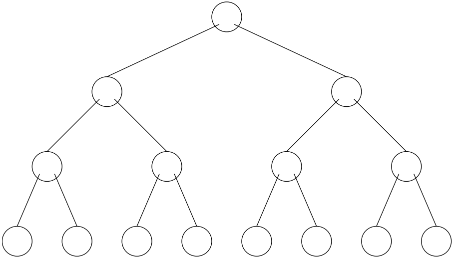


Eliminación perezosa

- Si se espera que el número de eliminaciones sea pequeño, la eliminación perezosa es una buena estrategia.
  - Al eliminar un elemento, se deja en el árbol marcándolo como eliminado.
  - Habiendo claves duplicadas, es posible decrementar el campo con la frecuencia de apariciones.
  - Si una clave eliminada se vuelve a insertar, se evita la sobrecarga de asignar un nodo nuevo.
- Si el número de nodos reales en el árbol es igual al número de nodos “eliminados”, se espera que la profundidad del árbol sólo aumente en uno (¿por qué?).
  - La penalización de tiempo es pequeña.

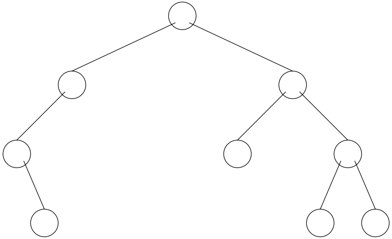
Si el número de nodos reales es igual al de “eliminados” su profundidad media aumenta en uno

- En un árbol completo, la mitad de los nodos están en el último nivel.
  - Si la mitad de los nodos tuvieran marca de borrado y los quitásemos, la profundidad del resto en media solo mejoraría en una unidad.



Si el número de nodos reales es igual al de “eliminados” la profundidad media aumenta en uno

- En un árbol aleatorio, alrededor de la mitad de los nodos son hojas.
  - Si la mitad de los nodos tuvieran marca de borrado y los quitásemos, la profundidad del resto en media solo mejoraría en una unidad.



Implementación de árboles binarios de búsqueda (i)

```
tipo
     PNodo = ^Nodo
     Nodo = registro
              Elemento : TipoElemento
              Izquierdo, Derecho : PNodo
            fin registro
     ABB = PNodo

procedimiento CrearABB(var A) O(1)
   A := nil
fin procedimiento
```

Implementación de árboles binarios de búsqueda (ii)

```
función Buscar(x, A): PNodo  {c.medio:O(log n) c.peor:O(n)}
  si A = nil entonces devolver nil
  sino si x = A^.Elemento entonces devolver A
  sino si x < A^.Elemento entonces
        devolver Buscar (x, A^.Izquierdo)
  sino devolver Buscar (x, A^.Derecho)
fin función

función BuscarMin(A): PNodo {c.medio:O(log n) c.peor:O(n)}
  si A = nil entonces devolver nil
  sino si A^.Izquierdo = nil entonces devolver A
  sino devolver BuscarMin (A^.Izquierdo)
fin función
```

Implementación de árboles binarios de búsqueda (iii)

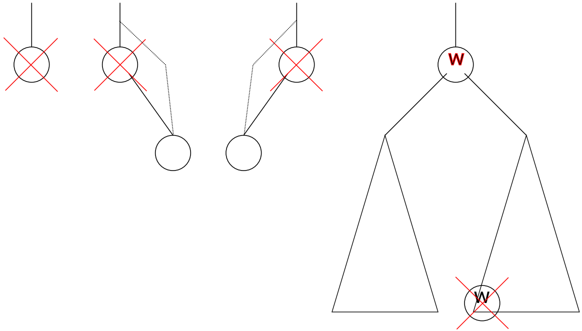
```
procedimiento Insertar(x, var A)
                                    {c.medio:O(log n) c.peor:O(n)}
  si A = nil entonces
    nuevo (A);
    si A = nil entonces error ''sin memoria''
    sino
      A^.Elemento := x;
      A^.Izquierdo := nil;
      A^.Derecho := nil
  sino si x < A^.Elemento entonces
          Insertar (x, A^.Izquierdo)
  sino si x > A^.Elemento entonces
          Insertar (x, A^.Derecho)
  { si x = A^.Elemento : nada }
fin procedimiento
```

Implementación de árboles binarios de búsqueda (iv)

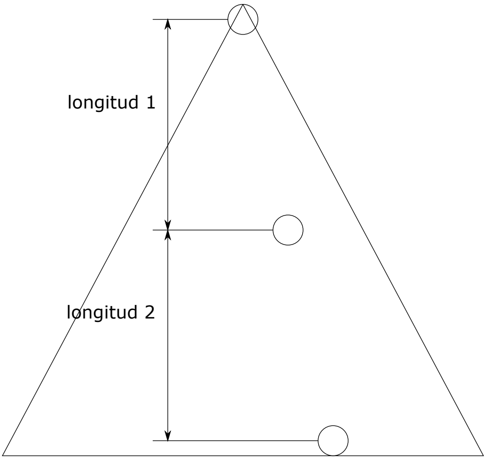
```
procedimiento Eliminar(x, var A)          {c.medio:O(log n)}
  si A = nil entonces error‘‘no encontrado’’{c.peor:O(n)}
  sino si  x < A^.Elemento entonces
           Eliminar (x, A^.Izquierdo)
  sino si  x > A^.Elemento entonces
           Eliminar (x, A^.Derecho)
  sino    { x = A^.Elemento }
    si A^.Izquierdo = nil entonces
      tmp := A; A := A^.Derecho; liberar (tmp)
    sino si A^.Derecho = nil entonces
      tmp := A; A := A^.Izquierdo; liberar (tmp)
    sino tmp := BuscarMin (A^.Derecho);
           A^.Elemento := tmp^.Elemento;
           Eliminar (A^.Elemento, A^.Derecho)
fin procedimiento
```

Posibilidades al borrar un nodo en un ABB

Alternativas al borrar un nodo en un ABB



Altura recorrida al eliminar un nodo



Recorridos de un árbol (i)

- En orden: Se procesa el subárbol izquierdo, el nodo actual y, por último, el subárbol derecho. O(n)

```
procedimiento Visualizar (A)
  si A <> nil entonces
    Visualizar (A^.Izquierdo);
    Escribir (A^.Elemento);
    Visualizar (A^.Derecho)
fin procedimiento
```

- Post-orden: Ambos subárboles primero. O(n)

```
función Altura (A) : número
  si A = nil entonces devolver -1
  sino devolver 1 + max (Altura (A^.Izquierdo),
                                Altura (A^.Derecho))
fin función
```

Recorridos de un árbol (ii)

- Pre-orden: El nodo se procesa antes. Ej: una función que marcase cada nodo con su profundidad. O(n)
- Orden de nivel: Todos los nodos con profundidad p se procesan antes que cualquier nodo con profundidad $p+1$.
  - Se usa una cola en vez de la pila implícita en la recursión. O(n)

```
procedimiento OrdenDeNivel (A)
  CrearCola(C);
  si A <> nil entonces InsertarEnCola(A, C);
  mientras no ColaVacía(C) hacer
    p:= QuitarPrimero(C);
    escribir(p^.Elemento); {Operación principal}
    si p^.Izq <> nil entonces InsertarEnCola(p^.Izq, C);
    si p^.Der <> nil entonces InsertarEnCola(p^.Der, C);
  fin mientras
fin procedimiento
```