

Estructuras de Datos

Clase 13 – Árboles binarios de búsqueda



Dr. Sergio A. Gómez
<http://cs.uns.edu.ar/~sag>

Departamento de Ciencias e Ingeniería de la Computación
 Universidad Nacional del Sur
 Bahía Blanca, Argentina



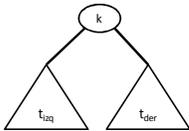
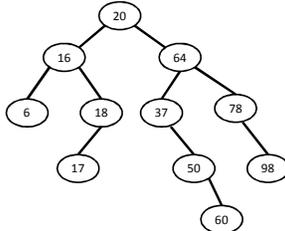
Motivaciones

- El árbol binario de búsqueda (ABB) es una estructura de datos útil para implementar conjuntos, mapeos y diccionarios.
- En un ABB las claves en los nodos se hallan ordenadas de una manera particular.
- El tiempo de insertar, recuperar y eliminar es proporcional a la altura del ABB.
- Si el árbol tiene n claves, la altura del ABB se halla entre $\log_2(n)$ y n .

Estructuras de datos - Dr. Sergio A. Gómez 2

Definición

- Un ABB es un árbol binario tal que:
 - Es vacío, o,
 - Es un nodo con rótulo k e hijos t_{izq} y t_{der} tales:
 - $k >$ claves de t_{izq} ,
 - $k <$ claves de t_{der} y,
 - t_{izq} y t_{der} son ABB.

Estructuras de datos - Dr. Sergio A. Gómez 3

Inserción en un ABB

- Las nuevas claves siempre se insertan como hijo de un nodo hoja.
- **Algoritmo** insertar(k, p) { Comienza en la raíz del ABB }
 - Si p es vacío entonces
 crear un nodo hoja con rótulo k
 - Sino
 si $k <$ clave(p) entonces
 insertar(k , hijo izquierdo de p)
 - sino si $k >$ clave(p) entonces
 insertar(k , hijo derecho de p)
 - sino si $k =$ clave(p) entonces
 reemplazar rótulo de p

Nota: La operación “reemplazar rótulo de p ” dependerá de si:
 (a) Implemento un conjunto: no hacer nada
 (b) Implemento un mapeo: modifico el valor de la entrada
 (c) Implemento un diccionario: Agrego una entrada con clave k

Estructuras de datos - Dr. Sergio A. Gómez 4

Ejemplos de inserciones

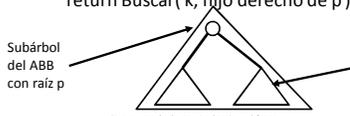
- **Ejemplo 1:** Dado un ABB vacío, realizar la siguiente secuencia de inserciones:
 10, 23, 8, 90, 78, 45, 3, 7, 12, 89, 44, 100, 95, 4
- **Ejemplo 2:** Dado un ABB vacío, realizar la siguiente secuencia de inserciones:
 2, 4, 6, 9, 10, 34, 78, 90, 100, 120
- **Ejemplo 3:** Dado un ABB vacío, realizar la siguiente secuencia de inserciones:
 120, 100, 45, 34, 29, 26, 16, 13, 5, 2

Estructuras de datos - Dr. Sergio A. Gómez 5

Búsqueda de una clave k partir del nodo p

Algoritmo Buscar(k, p) // La primera vez se llama con la raíz del ABB

- Si p es vacío entonces // CB: Se me terminó el árbol
 return falso // k no se encuentra en el ABB
- Sino si $k =$ clave(p) entonces // CB: encontré a k
 return true // k sí se encuentra en el ABB porque lo encontré
- Sino si $k <$ clave(p) entonces // k es menor a la clave de p
 return Buscar(k , hijo izquierdo de p)
- Sino // $k >$ clave(p)
 return Buscar(k , hijo derecho de p)



Si el $k >$ clave(p), tengo que buscar a k en el hijo de derecho de p , que también (conceptualmente) es un ABB porque descarté la raíz y el otro subárbol (hijo izquierdo de p).

Estructuras de datos - Dr. Sergio A. Gómez 6

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

Análisis de la complejidad temporal

- Sea h = altura del ABB y sea n = cantidad de claves del ABB.

$$T(h) = \begin{cases} c_1 & \text{si } h = 0 \\ c_2 + T(h - 1) & \text{si } h > 0 \end{cases}$$

con lo cual $T(h) = O(h)$.

- La altura en el peor caso es $h = n$ (se da cuando se hicieron inserciones de claves en forma ascendente o descendente)
- La altura en el mejor caso es $h = O(\log_2(n))$ ya que las inserciones produjeron un árbol lleno que tiene $n = 2^{h+1} - 1$ nodos; de ahí, despejo $h = \log_2(n + 1) - 1$.

Estructuras de datos - Dr. Sergio A. Gómez 7

Análisis de la complejidad temporal

- Otra forma de estudiar el peor caso del mejor caso del ABB (que se da cuando el árbol está lleno y no encuentro la clave buscada):
- Sea n = cantidad de claves del ABB.

$$T(n) = \begin{cases} c_1 & \text{si } n = 0 \\ c_2 + T((n - 1)/2) & \text{si } n > 0 \end{cases}$$

con lo cual $T(n) = O(\log_2(n))$.

Si el árbol tiene n claves, el hijo en el que busco recursivamente tiene $(n-1)/2$ claves porque descarté la raíz y el otro subárbol.

Estructuras de datos - Dr. Sergio A. Gómez 8

Implementación en Java: Preliminares

- Para facilitar la programación usaremos un árbol propio donde cada hoja tiene dos hijos dummy (placeholder según GT)
- Estos dummies sirven para insertar nuevas claves en su lugar y representan ABB vacíos.
- Ejemplo:

Referencias:

- Nodo con clave k
- Nodo dummy

Estructuras de datos - Dr. Sergio A. Gómez 9

Implementación en Java: Preliminares

Supongamos que deseamos insertar el 62, para ello buscamos su ubicación:

El rótulo dummy rojo se reemplaza con 62 y se crean dos nodos dummy nuevos.

Estructuras de datos - Dr. Sergio A. Gómez 10

Implementación en Java: Preliminares

- ¿Cómo es el árbol vacío ahora?

Respuesta: El árbol vacío es un nodo dummy.

Estructuras de datos - Dr. Sergio A. Gómez 11

NodoABB

```
public class NodoABB<E> extends Comparable<E> {
    private E rotulo;
    private NodoABB<E> padre, izq, der;

    public NodoABB( E rotulo, NodoABB<E> padre ) {
        this.rotulo = rotulo;
        this.padre = padre;
        izq = der = null;
    }

    public E getRotulo() { return rotulo; }
    public NodoABB<E> getPadre() { return padre; }
    public NodoABB<E> getIzq() { return izq; }
    public NodoABB<E> getDer() { return der; }

    public void setRotulo( E rotulo ) { this.rotulo = rotulo; }
    public void setIzq( NodoABB<E> izq ) { this.izq = izq; }
    public void setDer( NodoABB<E> der ) { this.der = der; }
    public void setPadre( NodoABB<E> padre ) { this.padre = padre; }
}
```

Estructuras de datos - Dr. Sergio A. Gómez 12

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

ABB

```
import java.util.Comparator;

public class ABB<E> extends Comparable<E>> {
    protected NodoABB<E> raiz;
    protected int size;
    Comparator<E> comp;

    public ABB(Comparator<E> comp) {
        raiz = new NodoABB<E>(null,null);
        size = 0;
        this.comp = comp;
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez

13

```
public boolean pertenece( E k ) {
    return buscar(k).getRotulo() != null;
}

private NodoABB<E> buscar( E k ) {
    return buscarAux( k, raiz );
}

private NodoABB<E> buscarAux( E k, NodoABB<E> nodov ) {
    if( nodov.getRotulo() == null ) return nodov;
    else {
        int c = comp.compare( k, nodov.getRotulo() );
        if( c == 0 ) return nodov;
        else if( c < 0 ) return buscarAux( k, nodov.getIzq() );
        else return buscarAux( k, nodov.getDer() );
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez

14

```
public void insertar( E k ) {
    NodoABB<E> nodov = buscar( k );
    if( nodov.getRotulo() == null ) {
        nodov.setRotulo( k );
        nodov.setIzq( new NodoABB<E>( null, nodov ) );
        nodov.setDer( new NodoABB<E>( null, nodov ) );
        size++;
    }
}
```

Estructuras de datos - Dr. Sergio A. Gómez

15

```
// insertar alternativo:
public void insertar2( E k ) {
    insertarAux( k, raiz );
}

private void insertarAux( E k, NodoABB<E> nodov ) {
    if( nodov.getRotulo() == null ) {
        nodov.setRotulo( k ); size++;
        nodov.setIzq( new NodoABB<E>( null, nodov ) );
        nodov.setDer( new NodoABB<E>( null, nodov ) );
    } else {
        int c = comp.compare( k, nodov.getRotulo() );
        if( c == 0 ) { /* hacer nada */ }
        else if( c < 0 ) insertarAux( k, nodov.getIzq() );
        else insertarAux( k, nodov.getDer() );
    }
}
```

16

Nota: El listado inorden de un ABB retorna las claves insertadas ordenadas en forma ascendente.

```
public String toString() {
    return inorden( raiz );
}

private String inorden( NodoABB<E> nodov ) {
    if( nodov.getRotulo() != null ) {
        return "(" + inorden( nodov.getIzq() )
            + nodov.getRotulo()
            + inorden( nodov.getDer() ) + ")";
    } else return "";
}
```

17

```
public class TestABB {
    private static void insertar_imprimir( ABB<Integer> t, int x ) {
        t.insertar( x ); System.out.println( "t.insertar(" + x + ") : " + t ); }

    public static void main( String [] args ) {
        ABB<Integer> t = new ABB<Integer>( new DefaultComparator<Integer>() );
        insertar_imprimir( t, 20 ); >java TestABB
        insertar_imprimir( t, 16 ); t.insertar(20): (20)
        insertar_imprimir( t, 64 ); t.insertar(16): ((16)20)
        insertar_imprimir( t, 6 ); t.insertar(64): (((16)20(64))
        insertar_imprimir( t, 100 ); t.insertar(6): (((6)16)20(64))
        insertar_imprimir( t, 3 ); t.insertar(100): (((((3)6)16)20(64(100))))
        System.out.println( "Pertenece(18): " + t.insertar(3): (((((3)6)16)20(64(100))))
            t.pertenece(18) ); Pertenece(18): false
        System.out.println( "Pertenece(100): " + t.pertenece(100) ); Pertenece(100): true
            t.pertenece(100) ); t.insertar2(999): (((((3)6)16)20(64(100(999))))
        t.insertar2( 999 ); t.insertar2(4): (((((3)6)16)20(64(100(999))))
        System.out.println( "t.insertar2(" + 999 + ") : " + t );
        t.insertar2( 4 );
        System.out.println( "t.insertar2(" + 4 + ") : " + t );
    }
}
```

18

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

Eliminación

- **Paso 1:** Buscar el nodo p con la clave k a eliminar
- **Paso 2:** Cuatro casos:
 - P es una hoja: Eliminarlo
 - P tiene sólo hijo izquierdo: hacer el bypass del padre de p con el hijo izquierdo de p
 - P tiene sólo hijo derecho: hacer el bypass del padre de p con el hijo derecho de p
 - P tiene dos hijos: Reemplazar el rótulo de p con el de su sucesor inorder y eliminar el sucesor inorder de p.
 - $T_{eliminar}(h) = O(h)$

Resolución de Problemas y Algoritmos - Dr. Sergio A. Gómez

19

```
// retorna null si no pudo eliminar a k, retorna k si la pudo eliminar
public E eliminar( E k ) {
    NodoABB<E> p = buscar( k );
    if( p.getRotulo() != null ) {
        E eliminado = p.getRotulo();
        eliminarAux( p );
        size--;
        return eliminado;
    } else return null;
}
```

Estructuras de datos - Dr. Sergio A. Gómez

20

```
private boolean isExternal( NodoABB<E> p ) {
    return p.getIzq().getRotulo() == null &&
        p.getDer().getRotulo() == null;
}

private boolean soloTieneHijolIzquierdo( NodoABB<E> p ) {
    return p.getIzq().getRotulo() != null &&
        p.getDer().getRotulo() == null;
}

private boolean soloTieneHijoDerecho( NodoABB<E> p ) {
    return p.getDer().getRotulo() != null &&
        p.getIzq().getRotulo() == null;
}
```

21

```
private void eliminarAux( NodoABB<E> p ) {
    if( isExternal(p) ) { // p es hoja: Convertir el nodo en un dummy y soltar sus hijos dummy.
        p.setRotulo( null ); p.setIzq( null ); p.setDer( null );
    } else { // p no es hoja
        if( soloTieneHijolIzquierdo(p) ) {
            // Enganchar al padre de p con el hijo izquierdo de p
            if( p.getPadre().getIzq() == p ) // p es el hijo izquierdo de su padre
                p.getPadre().setIzq( p.getIzq() ); // el hijo izq del padre de p es ahora el hijo de p
            // p es el hijo derecho de su padre
            p.getPadre().setDer( p.getIzq() ); // el hijo derecho del padre de p es el hijo de p
            p.getIzq().setPadre( p.getPadre() ); // Ahora el padre del hijo izq de p es su abuelo
        } else if( soloTieneHijoDerecho(p) ) {
            // Enganchar al padre de p con el hijo derecho de p
            if( p.getPadre().getIzq() == p ) // p es hijo izquierdo de su padre
                p.getPadre().setIzq( p.getDer() ); // el hijo izq del padre de p es el hijo de p
            else
                p.getPadre().setDer( p.getDer() ); // el hijo derecho del padre de p es el hijo de p
            p.getDer().setPadre( p.getPadre() ); // Ahora el padre del hijo der. de p es su abuelo
        } else { // p tiene dos hijos: seteo como rótulo de p al rótulo del sucesor inorder de p.
            p.setRotulo( eliminarMinimo( p.getDer() ) );
        }
    }
}
```

22

```
// Elimina el nodo con rótulo mínimo del subárbol que tiene como raíz a p
// El mínimo rótulo del subárbol que tiene como raíz a p es el rótulo del primer nodo que
// encuentro yendo a la izquierda que no tiene hijo izquierdo
private E eliminarMinimo( NodoABB<E> p ) {
    if( p.getIzq().getRotulo() == null ) { // El hijo izquierdo de p es un dummy
        E aRetornar = p.getRotulo(); // salvo el rótulo a devolver
        if( p.getDer().getRotulo() == null ) { // p es hoja (pues sus hijos son dummy)
            p.setRotulo( null ); // Convierto a p en dummy haciendo nulo su rótulo
            p.setIzq( null ); // y desenganchando sus dos hijos dummy
            p.setDer( null );
        } else { // p solo tiene hijo derecho (xq no tiene izquierdo)
            // Engancho al padre de p con el hijo derecho de p.
            // Seguro tiene que ser el hijo derecho de su padre.
            p.getPadre().setDer( p.getDer() );
            p.getDer().setPadre( p.getPadre() );
        }
        return aRetornar;
    } else { // Si p tiene hijo izquierdo, entonces p.getRotulo() no es el mínimo.
        // El mínimo tiene que estar en el subárbol izquierdo
        return eliminarMinimo( p.getIzq() );
    }
}
```

23

Variaciones

- **Implementación de Map<K,V> con ABB, dos soluciones:**

- El árbol se parametriza con E y E se instancia con entradas de K y V.
- El árbol y el nodo se pueden parametrizar con K y V.

- **Implementación de diccionario con ABB:**

Los nodos tienen una clave y una lista de entradas con la esa clave.

$$T_{findAll}(n) = O(h + s)$$

$$T_{find}(n) = O(h) \text{ donde:}$$

n = cantidad de entradas,

h = altura del ABB y

s = largo lista de entradas con una misma clave.

Estructuras de datos - Dr. Sergio A. Gómez

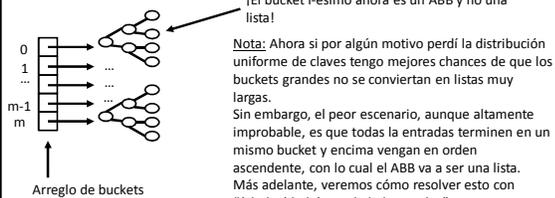
24

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente:
 “Estructuras de Datos. Notas de Clase”. Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.

Optimizaciones a la tabla de hash abierto

Teníamos n entradas insertadas en un arreglo de m buckets donde cada bucket es una lista de entradas colisionadas. En lugar de estructurar los buckets como una lista, se pueden estructurar como ABBs.

Entonces:



Estructuras de datos - Dr. Sergio A. Gómez

25

Bibliografía

- Capítulo 10 de M. Goodrich & R. Tamassia, Data Structures and Algorithms in Java. Fourth Edition, John Wiley & Sons, 2006.

Estructuras de datos - Dr. Sergio A. Gómez

26

El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: "Estructuras de Datos. Notas de Clase". Sergio A. Gómez. Universidad Nacional del Sur. (c) 2013-2019.