Diccionarios

Pablo Castro-UNRC

TAD Diccionario

El TAD Diccionario es una simplificación del TAD conjuntos. Solo tenemos las siguientes operaciones:

- Consultar pertenencia,
- Eliminar elementos,
- Insertar elementos.

Generalmente se quiere manejar una gran cantidad de datos por lo cual se pretende que las tres operaciones sean eficientes

TAD Diccionario

Una descripción más detallada:

- **Elementos:** Conjuntos finitos de un tipo T con una clave Comparable.
- **Operaciones:**
  - **Eliminar:** se elimina un elemento del conjunto,
  - **Insertar:** se inserta un elemento del conjunto,
  - **Pertenencia:** se consulta sobre la pertenencia de un elemento en el conjunto.

Interface Diccionario

```
// Ejemplo de una interfaz básica para diccionarios
public interface Diccionario<T extends Comparable<T>>{

    // inserta un elemento en el diccionario
    // pre: true
    // post: agrega el elemento elem en el diccionario, puede tirar una excepción
     // por falta de espacio
    public void insertar(T elem) throws ExcepcionDiccionario;

    // borra un elemento del diccionario
    // pre: true
    // post: remueve el elemento elem el diccionario
    public void remover(T elem);

    // permite consultar si existe un elemento en el diccionario
    // pre: true
    // post: retorna una referencia al elemento si se encuentra, y null en otro caso
    public T pertenece(T elem);

}
```

Diferentes implementaciones son posibles, acá usamos genericidad

Implementaciones

Algunas observaciones:

log 1000000=19.93

- Sobre estructuras ordenadas la búsqueda se puede hacer eficientemente en O(log n),

- Los elementos deben tener una clave Comparable para que los elementos se puedan ordenar,

- En general se manejan una gran cantidad de elementos, si las operaciones son O(n) no se pueden usar.

Con Arreglos Ordenados

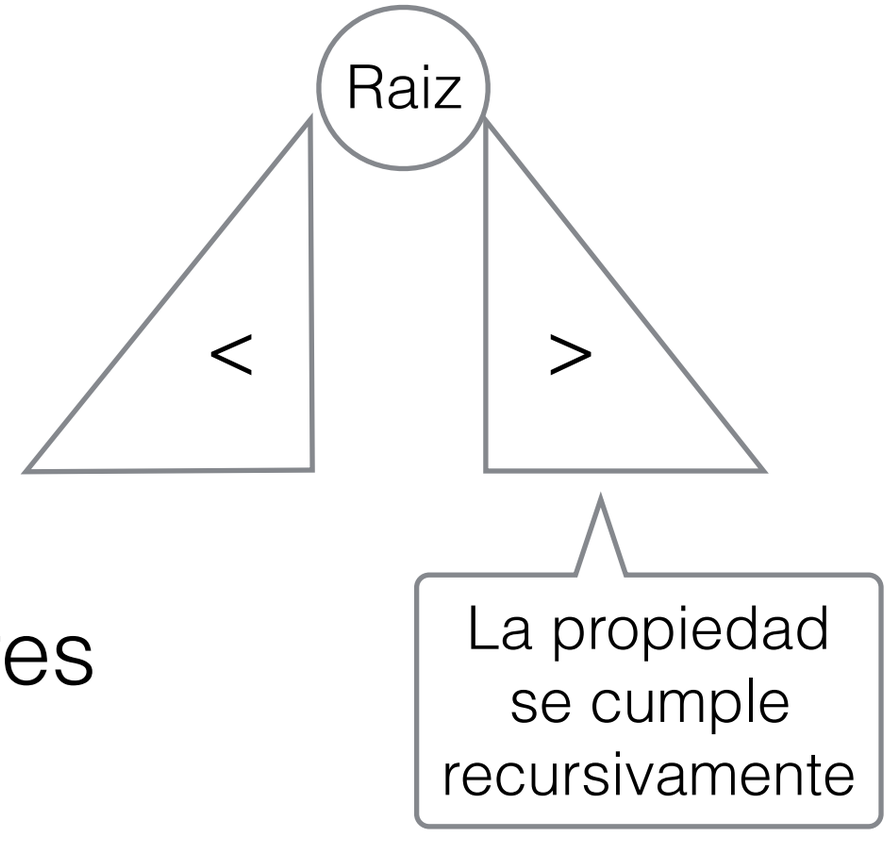
- La búsqueda es O(log n),
- El insertar es O(n), se busca la posición, y luego se tiene que correr n elementos,
- El eliminar es O(n) se busca el elemento a eliminar y luego se corren n elementos,
- Mejores implementaciones son posibles!

Se puede marcar el elemento como eliminado mejorando el tiempo de ejecución

Árboles Binarios de Búsqueda (ABB)

Ofrecen una implementación elegante de Diccionarios:

- Son arboles binarios,
- Los nodos del hijo izquierdo son menores a la raíz,
- Los nodos del hijo derecho son mayores a la raíz,
- Los hijos izquierdo y derecho son ABB's



Implementación en JAVA

Una posible implementación utilizando memoria dinámica y genericidad:

```
class ABBNodo<T extends Comparable<T>>{
    private T elem;
    private ABBNodo<T> hi;
    private ABBNodo<T> hd;

    public ABBNodo(T elem){
        this.elem = elem;
    }
    // Implementar resto de los métodos…
```

Esta clase provee la mayor parte de las funcionalidades, que después se llaman desde ABB

```
class ABB<T extends Comparable<T>> implements Diccionario<T>, BinaryTreeBasis<T>{
    private ABBNodo<T> raiz;

    public ABB(){
        raiz = null;
    }

    public ABB(ABBNodo<T> raiz){
        this.raiz = raiz;
    }
   // Implementar resto de los métodos…
```

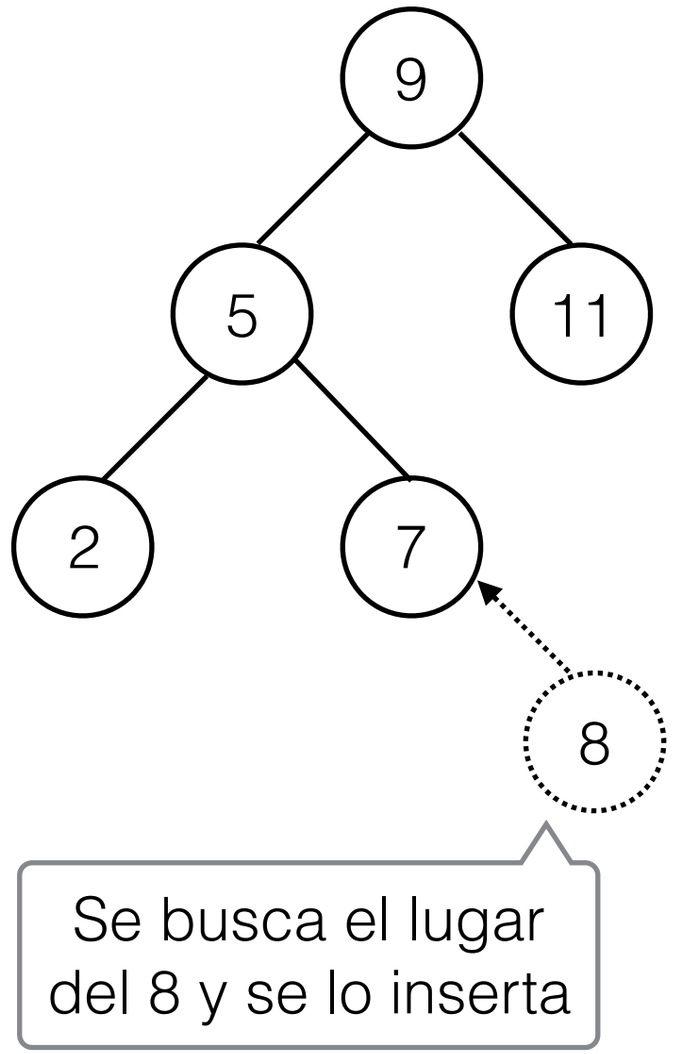
ABB implementa Diccionarios y también árboles genéricos

Inserción ABB's

1. Se busca el lugar en dónde insertar,

2. Si el elemento ya está no se hace nada, o se agrega la información correspondiente,

3. sino se pone a la izquierda o derecha del último nodo de acuerdo a la comparación.



Borrado en ABB

Hay que tener cuidado de preservar el invariante de ABB's:

- Se busca el elemento a borrar.

- Si no se encuentra, se termina,

- En otro caso, se elimina el elemento, hay varios casos:

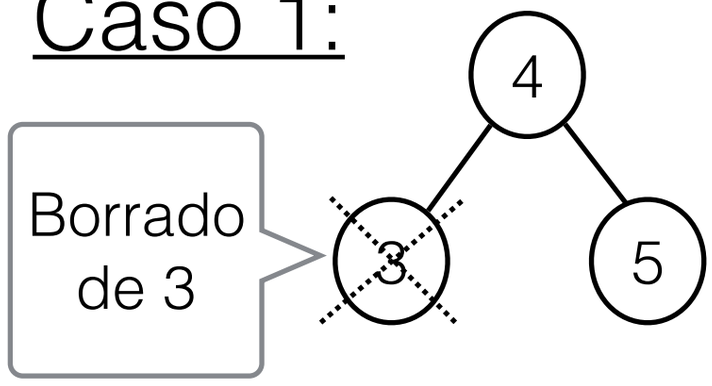
  - El elemento es un hoja, fácil el elemento se borra,

  - El elemento no tiene hi ó hd, se reemplaza el elemento por la raíz del hijo que existe,

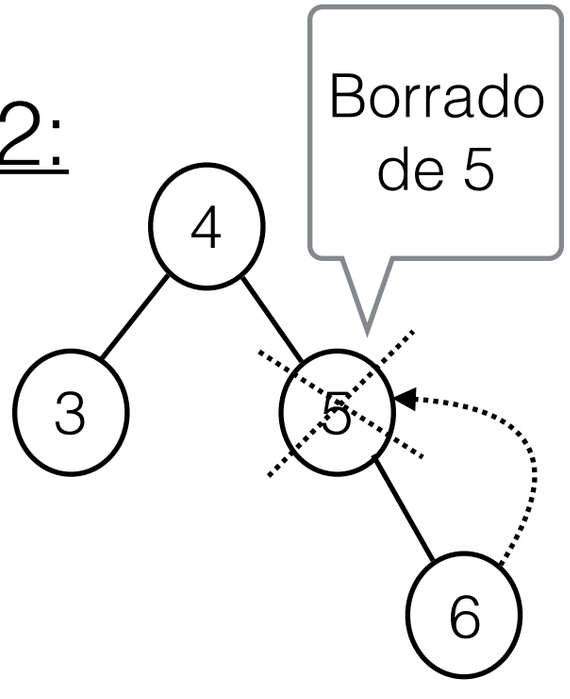
  - El elemento tiene ambos hijos, se lo reemplaza por el máximo de la izq. o mínimo de la derecha.

Borrado en ABB's

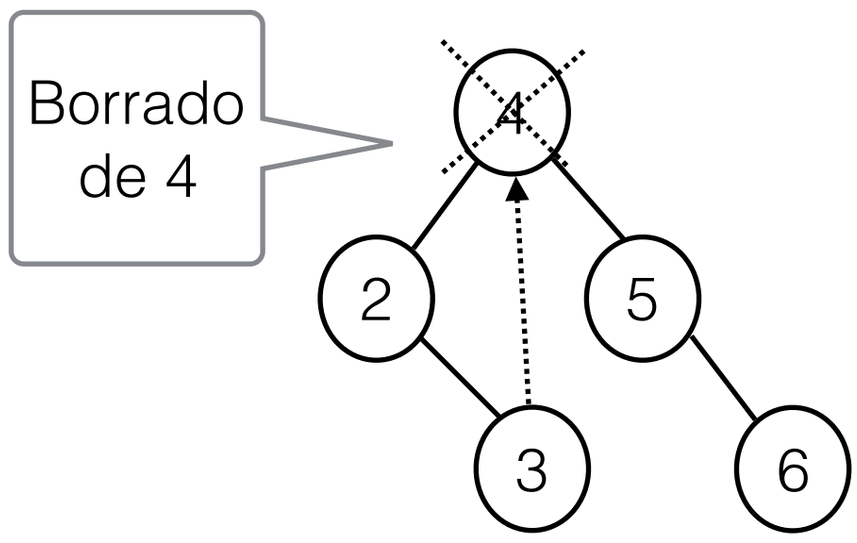
Caso 1:



Caso 2:



Caso 3:



Tiempo de Ejecución en ABB's

- **Buscar**: La búsqueda está acotado por la altura, si el árbol está balanceado es O(log n), pero en el peor caso es O(n),

- **Insertar**: También, buscar es en el peor caso O(n), agregar el elemento es constante, por lo tanto es O(n),

- **Eliminar**: Buscar el elemento es O(n), buscar el máximo es O(n), por lo cual es O(n)

Todas estas operaciones son O(log n) cuando el árbol está balanceado.

TreeSort

Podemos utilizar los ABB's para ordenar una secuencia:

- Se recorre la secuencia, insertando uno por uno los elementos en un ABB nuevo,
- Se saca el max. del ABB y se lo inserta en una secuencia, se repite este procedimiento hasta que el ABB esté vacío,

En el peor caso este algoritmo es $O(n^2)$, sin embargo, si todos los elementos tienen la misma probabilidad de ser insertados, los árboles tienden a ser balanceados y entonces el algoritmo es O(n*log n)