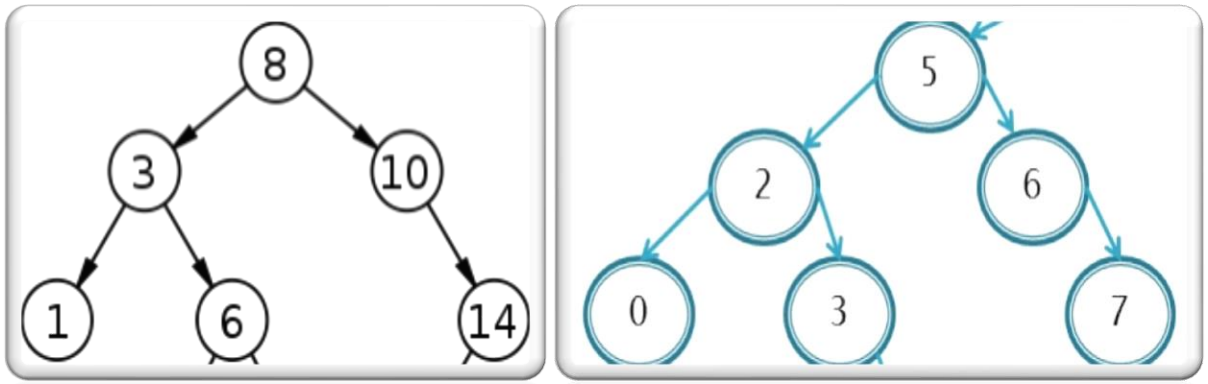
[Estructuras de Datos]



ARBOLES BINARIOS DE BÚSQUEDA.

MAPEOS Y DICCIONARIOS CON ABB.

Copyright

ÁRBOLES BINARIOS DE BÚSQUEDA

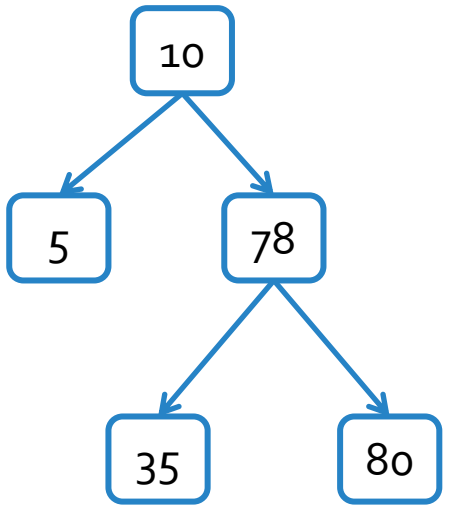
Se recomienda que todo lo documentado en las siguientes secciones sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

Introducción: ¿qué es un árbol binario de búsqueda?

- Un **árbol binario de búsqueda (ABB)** es un **ED** que permite implementar de forma **eficiente** **conjuntos**, **mapeos** y **diccionarios**.
- El **ABB** mantiene un **ordenamiento** en particular de los **elementos** que **almacena**.
- Sea **N** un **nodo** del **ABB**, luego el **ordenamiento** es tal que:
  - Los **elementos** del **subárbol izquierdo** a **N**, son **menores** que el elemento de **N**.
  - Los **elementos** del **subárbol derecho** a **N,** son **mayores** que el elemento de **N**.
- El interés de los **ABB** radica en que la **búsqueda** de un elemento suele **ser muy eficiente**, y que su recorrido **inorden** proporciona los elementos **ordenados** de forma **ascendente**.

Introducción: ¿qué es un árbol binario de búsqueda?

- Sea **N** un **nodo** del **ABB**, luego el **ordenamiento** es tal que:
  - Los **elementos** del **subárbol izquierdo** a **N**, son **menores** que el elemento de **N**.
  - Los **elementos** del **subárbol derecho** a **N,** son **mayores** que el elemento de **N**.



- *Arbol.insertar(10).*
- *Arbol.insertar(5).*
- *Arbol.insertar(78).*
- *Arbol.insertar(80).*
- *Arbol.insertar(35)*
- *Inorden: 5-10-35-78-80*

Notar que en su definición, no es considerado el hecho que dos elementos **puedan ser iguales**. Esto se debe a que esta situación **empeora** la eficiencia en las búsquedas.

ABB :: IMPLEMENTACIÓN MEDIANTE NODOS CON REF. AL PADRE e HIJOS

Se recomienda que todo lo documentado en las siguientes secciones sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

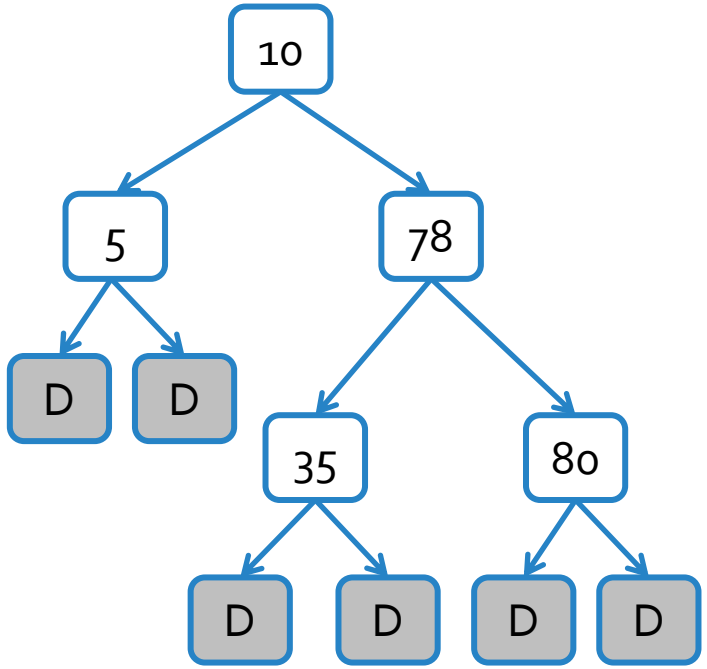
ED NodoABB

- Bajo esta implementación, se define una ED **NodoABB** que mantiene un rótulo, así como una **referencia a tres nodos** que representan los nodos **padre** e **hijo izquierdo** y **derecho**, respectivamente, del nodo modelado.

```
public class NodoABB<E> {

  protected E rotulo;
  protected NodoABB<E> padre, hi, hd;

  public NodoABB(E r, NodoABB<E> p){
    rotulo = r;
    padre = p;
  }

  public void setRotulo(E r){ rotulo = r;}
  public void setParent(NodoABB<E> p){padre = p;}
  public void setLeft(NodoABB<E> i){ hi = i;}
  public void setRight(NodoABB<E> d){hd = d;}
  public E getRotulo(){ return rotulo; }
  public NodoABB<E> getParent(){ return padre; }
  public NodoABB<E> getLeft(){ return hi;}
  public NodoABB<E> getRight(){ return hd; }
}
```

ED ABB

- Bajo esta implementación, se define una ED **ABB** que mantiene un **NodoABB** raíz, y un **comparador** de elementos, con lo que puede acceder a todo el árbol.
- El **ABB** que se implementará contemplando un **árbol propio**, donde cada **hoja** tiene dos nodos ***DUMMY*** como hijos, para facilitar la programación.



ED ABB

- Bajo esta implementación, se define una ED **ABB** que mantiene un **NodoABB** raíz, y un **comparador** de elementos, con lo que puede acceder a todo el árbol.

```
public class ABB<E extends Comparable<E>> {

  protected NodoABB<E> raiz;
  protected Comparator<E> comparador;

  public ABB() {
    raiz = new NodoABB<E>(null, null);
    comparador = new DefaultComparator<E>();
  }

  public ABB(Comparator<E> comp) {
    raiz = new NodoABB<E>(null, null);
    comparador = comp;
  }

 //Sigue en próxima slide.
}
```

*Como existe el caso en el que el cliente no especifique un comparador para los elementos y se use un **comparador por defecto**, se requiere que E sea **comparable**.*

*En caso de que **no se parametrice** un **comparador** de elementos E definido por el cliente, se utiliza un **comparador por defecto** que delega la tarea de la comparación al tipo E que es **comparable**.*

*En ambos constructores, se inicializa un **árbol vacío**. Recordar que el contexto de la ED considerada, un árbol vacío es un árbol con un nodo DUMMY como raíz.*

ED ABB

- Bajo esta implementación, se define una ED **ABB** que mantiene un **NodoABB** raíz, y un **comparador** de elementos, con lo que puede acceder a todo el árbol.

```
public class ABB<E extends Comparable<E>> {

  protected NodoABB<E> raiz;
  protected Comparator<E> comparador;

  //Constructores de clase...

  public NodoABB<E> raiz(){…}
  public NodoABB<E> buscar(E e){…}
  public void expandir(NodoABB<E> n, E e){…}
  public void eliminar(E e){…}
}
```

Permite recorrer la ED a partir de la raíz.

Busca *el* ***NodoABB*** *en el que debe almacenarse el elemento. Como el ABB contempla nodos DUMMY, siempre retorna un* ***NodoABB****.*

Expande *un nodo DUMMY, reconvirtiéndolo en un nodo con rótulo y dos hijos DUMMY.*

Elimina *el elemento, manteniendo el orden establecido por el* ***ABB****. Si el elemento no existe, no modifica el ABB.*

Haciendo uso de ***buscar(…)*** y ***expandir(…)***, se puede implementar la **inserción** de elementos en el **ABB**.

Buscando el **nodo n** donde debería ubicarse un elemento E, luego:

1) Si **n** es *DUMMY*, se debe ***expandir*** **n** con el elemento E.

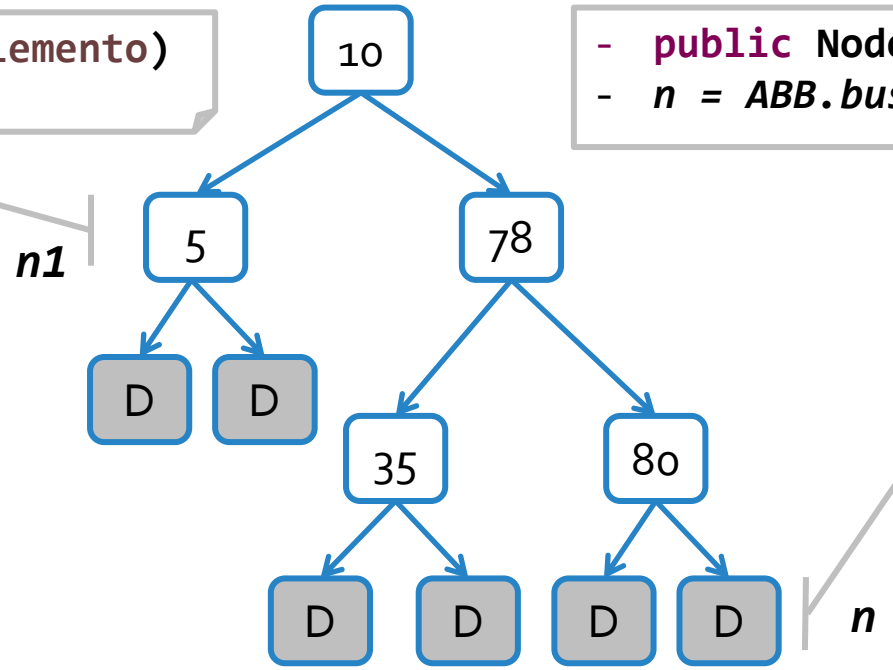
2) Si **n** no es *DUMMY*, el elemento ***ya existe*** (*duplicado*) y se deberá aplicar el criterio del TDA implementado.

ED ABB

- Bajo esta implementación, se define una ED **ABB** que mantiene un **NodoABB** raíz, y un **comparador** de elementos, con lo que puede acceder a todo el árbol.
- El **ABB** que se implementará contemplando un **árbol propio**, donde cada **hoja** tiene dos nodos ***DUMMY*** como hijos, para facilitar la programación.

```
- public NodoABB<E> buscar(E elemento)
- n1 = ABB.buscar(5)
```

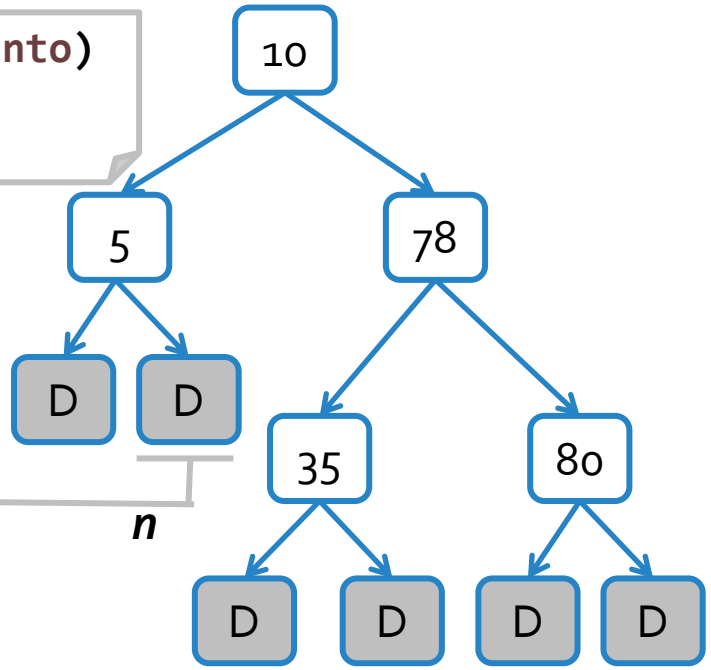
```
- public NodoABB<E> buscar(E elemento)
- n = ABB.buscar(85)
```



ED ABB

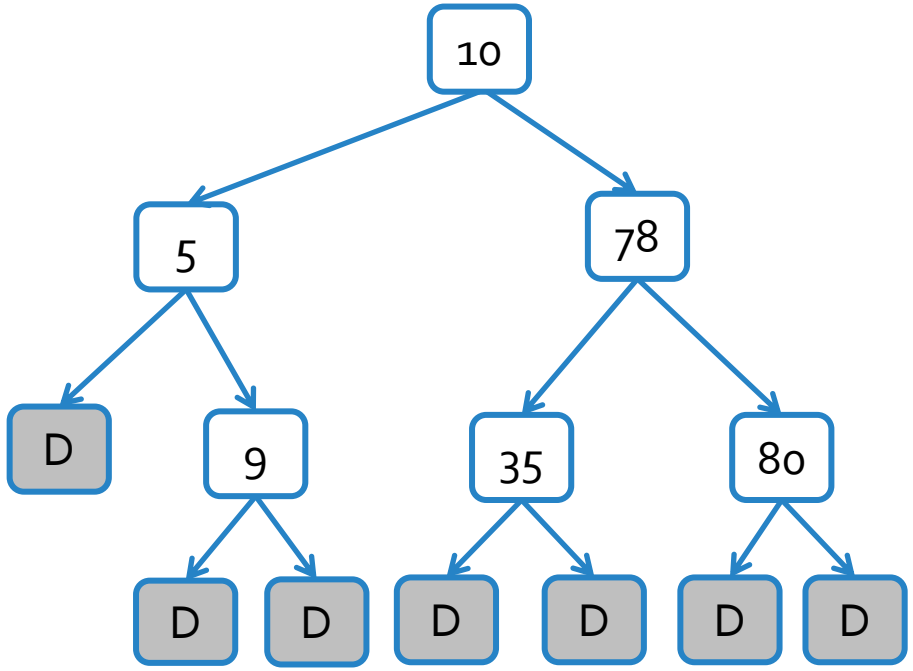
- Bajo esta implementación, se define una ED **ABB** que mantiene un **NodoABB** raíz, y un **comparador** de elementos, con lo que puede acceder a todo el árbol.
- El **ABB** que se implementará contemplando un **árbol propio**, donde cada **hoja** tiene dos nodos ***DUMMY*** como hijos, para facilitar la programación.

```
- public NodoABB<E> buscar(E elemento)
- n = ABB.buscar(9)
- ABB.expandir(n, 9)
```



ED ABB

- Bajo esta implementación, se define una ED **ABB** que mantiene un **NodoABB** raíz, y un **comparador** de elementos, con lo que puede acceder a todo el árbol.
- El **ABB** que se implementará contemplando un **árbol propio**, donde cada **hoja** tiene dos nodos ***DUMMY*** como hijos, para facilitar la programación.

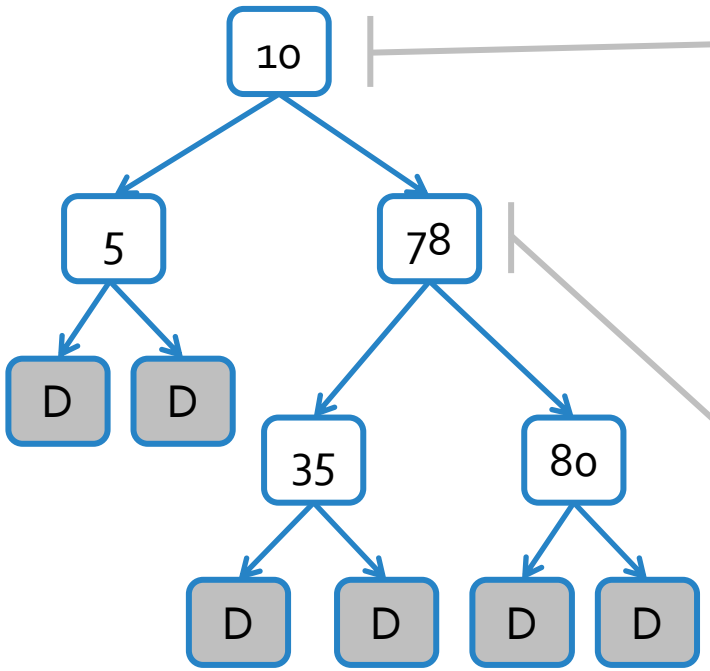


MAPEO Y DICCIONARIO :: IMPLEMENTACIÓN CON ABB

Se recomienda que todo lo documentado en las siguientes secciones sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

Uso de la ED ABB :: Entradas comparables

- Un **árbol binario de búsqueda (ABB)** es un **ED** que permite implementar de forma **eficiente** **conjuntos**, **mapeos** y **diccionarios**.
- ¿Cómo debe utilizarse un **ABB** para implementar un **mapeo** o **diccionario**?



- *El **ABB** es una ED para **elementos genéricos** E.*
- *En la implementación de mapeos y diccionarios, los **elementos genéricos** deberán ser **Entradas**.*

```
public class ABB<E extends Comparable<E>> {…}
```

- *El **ABB** es una ED que **requiere** que sus **elementos genéricos** E sean **comparables**.*
- *En la implementación de mapeos y diccionarios, las entradas utilizadas **deberán** ser **comparables**.*
- *Las **entradas comparables** manipularán **claves** y **valores**. En este sentido, la comparación entre ellas se dará en términos de las **claves de tipo genérico K**, que necesariamente deberá ser comparable también.*

Uso de la ED ABB :: Entradas comparables

- Las **entradas comparables** manipularán **claves** y **valores**. La comparación entre ellas se dará en términos de las **claves** de tipo genérico K, que necesariamente deberá ser comparable también.

```
public class EntradaComparable<K,V> implements Entry<K,V>
```

*La clase EntradaComparable debe ser **comparable**. Siendo comparable permite que el ABB pueda establecer la relación de orden entre todas ellas. Para ser **comparable**, debe implementar esta interfaz.*

```
public class EntradaComparable<K, V> implements Entry<K,V>, Comparable<EntradaComparable<K,V>>
```

*La comparación entre dos instancias de la clase EntradaComparable se dará en términos de sus **claves genéricas de tipo K**. Como las claves son genéricas y deben poder compararse, luego, deberán ser **comparables** también.*

```
public class EntradaComparable <K extends Comparable<K>, V> implements Entry<K,V>, Comparable<EntradaComparable<K,V>>
```

Uso de la ED ABB :: Entradas comparables

- Las **entradas comparables** manipularán **claves** y **valores**. La comparación entre ellas se dará en términos de las **claves** de tipo genérico K, que necesariamente deberá ser comparable también.

```
public class EntradaComparable <K extends Comparable<K>, V> implements Entry<K,V>, Comparable<EntradaComparable<K,V>>{
  private K key;
  private V value;

  public EntradaComparable(K k, V v) {
    key = k;
    value = v;
  }

  @Override
  public int compareTo(EntradaComparable<K,V> e) {
    return key.compareTo(e.getKey());
  }

  //Seters and geters de los atributos de instancia.

}
```

- *Como la clase EntradaComparable es **comparable**, debe implementar el método **compareTo(..)***
- *Como la clase EntradaComparable realiza su comparación en términos de las claves que almacena y estas claves son genéricas y **comparables**, se delegará la operación **compareTo(..)** de la clase EntradaComparable justamente a la operación **compareTo(..)** de la clase K.*

Mapeo con ABB :: definiendo su ED.

- Bajo esta implementación, se define una ED **MapeoConABB** que mantiene un **ABB** de Entradas<K,V>. En particular, las **entradas** serán **comparables**.

```
public class MapeoConABB<K extends Comparable<K>,V> implements Map<K,V>{

  protected ABB<EntradaComparable<K,V>> abb;
  protected int size;

  public MapeoConABB(){
    abb = new ABB<EntradaComparable<K,V>>();
    size = 0;
  }
  ...
}
```

*Al utilizar un **ABB**, se debe **asegurar** que se pueda establecer **una relación de orden** entre las **claves**, por lo que el tipo K **debe ser comparable**. A diferencia de otras implementaciones de mapeo, **no alcanza** con saber cuándo **dos claves son iguales**, sino se debe saber **cuándo son menores o mayores que otras**, también.*

*Al utilizar un **ABB**, se debe **asegurar** que sus elementos son **comparables**. Como el **ABB** almacenará **Entradas<K,V>**, se deberá configurar una **clase de entradas** que sean **comparables**.*

*Como las **Entradas** son **comparables**, el **ABB** puede utilizar un **comparador** por defecto que es suficiente para estimar la relación de orden entre las **entradas** con las que se trabaje.*

Mapeo con ABB :: put().

```
public class MapeoConABB<K extends Comparable<K>,V> implements Map<K,V>{
  public V put(K key, V value) throws InvalidKeyException {
    EntradaComparable<K,V> entrada;
    NodoABB<EntradaComparable<K,V>> nodo;
    V a_retornar = null;

    check_key(key);

    entrada = new EntradaComparable<K,V>(key, value);
    nodo = abb.buscar(entrada);

    // Si el nodo correspondiente a entry es DUMMY, no existe en el ABB una entrada con clave key, por lo que se debe
    // agregar una nueva entrada. Caso contrario, existe entrada con clave key, y se debe modificar su valor.
    if (nodo.getRotulo() == null) {
      abb.expandir(nodo, entrada);
      size++;
    }else {
      a_retornar = nodo.getRotulo().getValue();
      nodo.getRotulo().set_value(value);
    }
    return a_retornar;
  }
}
```

Diccionario con ABB :: definiendo su ED.

- Bajo esta implementación, se define una ED **DiccionarioConABB** que mantiene un **ABB** de Entradas<K,PositionList<Entry<K,V>>>. En particular, las **entradas** serán **comparables**.

```
public class DiccionarioConABB<K extends Comparable<K>,V> implements Dictionary<K,V>{

  protected ABB<EntradaComparable<K,PositionList<Entry<K,V>>>> abb;
  protected int size;

  public DiccionarioConABB(){
    abb = new ABB<EntradaComparable<K,PositionList<Entry<K,V>>>>();
    size = 0;
  }
  ...
}
```

*Ante **entradas** con **igual clave**, en lugar de **modificar** el valor de la **entrada** como sucedía en el mapeo, se debe considerar **almacenar** una **lista de entradas** correspondientes a dicha clave.*

Diccionario con ABB :: insert().

```
public class DiccionarioConABB<K extends Comparable<K>,V> implements Dictionary<K,V>{
  public Entry<K, V> insert(K key, V value) throws InvalidKeyException {
    EntradaComparable<K,PositionList<Entry<K,V>>> entrada;
    NodoABB<EntradaComparable<K,PositionList<Entry<K,V>>>> nodo;
    Entry<K,V> a_retornar;

    check_key(key);
    entrada = new EntradaComparable<K,PositionList<Entry<K,V>>>(key, null);
    nodo = abb.buscar(entrada);
    a_retornar = new EntradaComparable<K,V>(key, value);

    // Si el nodo correspondiente a entry no es DUMMY, existe al menos una entrada en ABB con la clave key.
    if (nodo.getRotulo() != null) {
      nodo.getRotulo().getValue().addLast(a_retornar);
    }else {
      abb.expandir(nodo, entrada);
      entrada.set_value(new DoubleLinkedList<Entry<K,V>>());
      entrada.getValue().addLast(a_retornar);
    }
    size++;
    return a_retornar;
  }
}
```

Fin de la presentación.