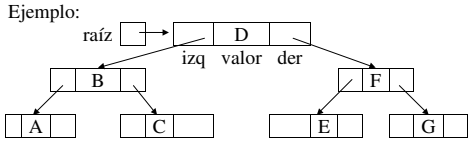
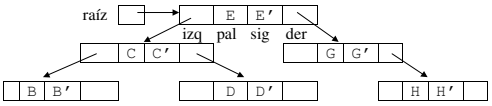
¿árbol binario de búsqueda o árbol de búsqueda binaria(ABB)?

Estructura de datos (recursiva) que
- está vacía , o
- contiene un nodo raíz con un valor y referencias a dos ABBs (izquierdo y derecho)
- los valores contenidos en el subárbol izquierdo son menores que el valor en la raíz
- los valores contenidos en el subárbol derecho son mayores que el valor en la raíz

Ejemplo:



Diccionario: implementación con ABB
- tiempo de todas las operaciones proporcional a altura del árbol
- si está balanceado, altura=$\log_2 n$




```
class Nodo{
public Comparable palabra;
public Object significado;
public Nodo izq, der;
public Nodo(
Comparable x,Object y,Nodo z,Nodo w){
   palabra=x; significado=y; izq=z; der=w;
}
}
```

```
class Diccionario
{
protected Nodo raiz;

public Diccionario(){
   raiz=null;
}
public Object buscar(Comparable x){
   Nodo r=referencia(x,raiz);
   return r==null ? null : r.significado;
}
public boolean cambiar(Comparable x,Object y){
   Nodo r=referencia(x,raiz);
   if(r==null) return false;
   r.significado = y;
   return true;
}
```

```
//búsqueda iterativa en un ABB
protected Nodo referencia(Comparable x,Nodo r){
   while(r!=null){
      int c=x.compareTo(r.palabra);
      if(c==0) return r;
      r = c<0 ? r.izq : r.der;
   }
   return null;
}
//búsqueda recursiva en un ABB
protected Nodo referencia(Comparable x,Nodo r){
   if(r==null) return null;
   int c=x.compareTo(r.palabra);
   if(c==0) return r;
   return referencia(x, c<0 ? r.izq : r.der);
}
```

```
public boolean agregar(Comparable x,Object y)
throws DiccLleno{
   if(referencia(x,raiz)!=null) return false;
   raiz=agregar(x,y,raiz);
   return true;
}
protected Nodo agregar
(Comparable x,Object y,Nodo r)
throws DiccLleno{
   if(r==null) return new Nodo(x,y,null,null);
   if(x.compareTo(r.palabra) < 0)
      r.izq=agregar(x,y,r.izq);
   else
      r.der=agregar(x,y,r.der);
   return r;
}
```

Solución 2: en un sólo recorrido no recursivo del ABB

```
public boolean agregar(Comparable x,Object y)
throws DiccLleno{
   Nodo q=new Nodo(x,y,null,null);
   if(raiz==null){raiz=q; return true;}
   Nodo r=raiz;
   while(true){
      int c=x.compareTo(r.palabra);
      if(c==0) return false; //ya existe
      if(c<0)
         if(r.izq==null){r.izq=q; break;}
         else r=r.izq;
      else
         if(r.der==null){r.der=q; break;}
         else r=r.der;
   }
   return true;
}
```

Problema. Completar la clase Diccionario1

```
class Diccionario1 extends Diccionario
{
public Diccionario1(){super();}

//entregar número de palabras en el Diccionario. Ejemplo: 6
public int numeroPalabras(){return numeroPalabras(raiz);}
protected int numeroPalabras(Nodo r){...}

//entregar palabras en orden alfabético. Ej: "B C D E G H "
public String palabras(){...}
protected String palabras(Nodo r){ ...}
}
```

```
//recursiva
public int numeroPalabras(){
   return numeroPalabras(raiz);
}
protected int numeroPalabras(Nodo r){
  if(r==null) return 0;
  return 1
       + numeroPalabras(r.izq)
       + numeroPalabras(r.der);
}
```

```
//iterativa
public int numeroPalabras(){try{
  int n=0;
   Stack s=new Stack();
  s.push(raiz);
  while(!s.empty()){
    Nodo r=(Nodo)s.pop();
    if(r!=null){
      ++n;
      s.push(r.izq);
      s.push(r.der);
    }
  }
  return n;
}catch(Exception x){}
}
```

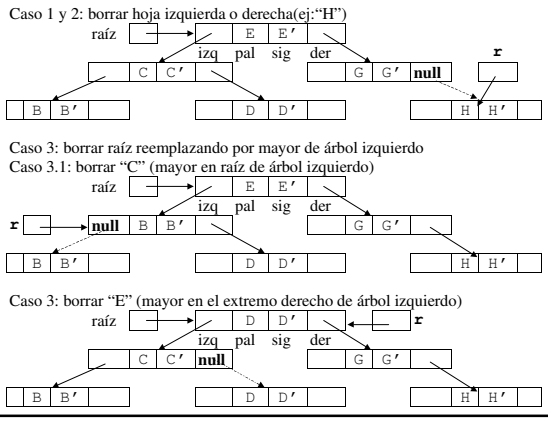
```
//iterativa
public int numeroPalabras(){try{
  int n=0;
   Stack s=new Stack();
  if(raiz!=null) s.push(raiz);
  while(!s.empty()){
    ++n;
    Nodo r=(Nodo)s.pop();
    if(r.izq!=null) s.push(r.izq);
    if(r.der!=null) s.push(r.der);
  }
  return n;
}catch(Exception x){}
}
```

```
public String palabras(){
  return palabras(raiz);
}
protected String palabras(Nodo r){
   if(r==null) return "";
   return palabras(r.izq)
        + r.palabra.toString() + " "
        + palabras(r.der);
}
```

Prop: ¿Cuál es el resultado de este otro caso?

```
public String palabras(){
  return palabras(raiz);
}
protected String palabras(Nodo r){
   if(r==null) return "";
   return palabras(r.der)
        + r.palabra.toString() + " "
        + palabras(r.izq);
}
```

```
public boolean borrar(Comparable x){
   Nodo r=referencia(x, raiz);
   if(r==null) return false;
   raiz=borrar(x,raiz);
   return true;
}
//borrar Nodo con x y devolver raíz del ABB
protected Nodo borrar(Comparable x,Nodo r){
   if(r==null) return null;
   int c=x.compareTo(r.palabra);
   if(c==0) return borrar(r);
   if(c<0)
      r.izq=borrar(x,r.izq);
   else
      r.der=borrar(x,r.der);
   return r;
}
```




```
protected Nodo borrar(Nodo r){
//caso 1: sólo árbol derecho
if(r.izq==null) return r.der;

//Caso 2:solo árbol izquierdo
if(r.der==null) return r.izq;

//Caso 3:reemplazar por mayor de arbol izq

//caso 3.1: mayor en raíz de árbol izquierdo
if(r.izq.der==null){
   r.palabra = r.izq.palabra;
   r.significado = r.izq.significado;
   r.izq = r.izq.izq; //enlazar hijos menores
}
```

```
//caso 3.2: mayor a la derecha de árbol izq
else{
   //buscar ref de antecesor de mayor
   Nodo rAnt=r.izq;
   while(rAnt.der.der!=null)
      rAnt=rAnt.der;

   //reemplazar raiz por mayor
   r.palabra = rAnt.der.palabra;
   r.significado = rAnt.der.significado;
   rAnt.der = rAnt.der.izq;//enlazar menores
}
return r;
}
```

Propuesto: borrar reemplazando por menor de árbol derecho

Solución 2: en un recorrido no recursivo

```
public boolean borrar(Comparable x)
{
//buscar ref r de x, recordando ref antecesor
Nodo r=raiz, rAnt=null;

while(r!=null)
{
  int c=x.compareTo(r.palabra);
  if(c==0) break;
  rAnt=r;
  r=c<0 ? r.izq: r.der;
}
if(r==null) return false;//no está
```

```
//borrar nodo r, actualizando antecesor
if(r==raiz)

   raiz=borrar(r);

else if(r==rAnt.izq)

   rAnt.izq=borrar(r);

else

   rAnt.der=borrar(r);

return true;
}
```