Grupo Weboo
Universidad de La Habana

Examen Final de Programación
Curso 2015-2016

Árbol Binario de Búsqueda Extendido

NOTA: Si usted está leyendo este documento sin haber extraído el compactado que se le entregó, ciérrelo ahora, extraiga todos los archivos en el escritorio, y siga trabajando desde ahí. Es un error común trabajar en la solución dentro del compactado, lo cual provoca que los cambios **no se guarden**. Si usted comete este error y entrega una solución vacía, no tendrá oportunidad de reclamar.

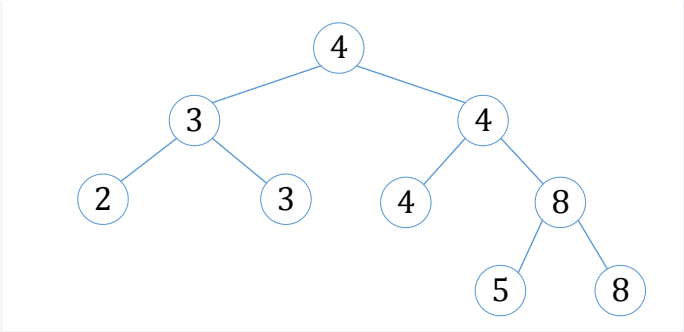
Un árbol binario de búsqueda es un árbol binario en el que para todo nodo se cumple que los valores de los nodos de su subárbol izquierdo son menores estrictos que el valor de dicho nodo y los valores de los nodos de su subárbol derecho son mayores estrictos.

Le llamaremos **árbol binario de búsqueda extendido** (ABBE) a un árbol binario de búsqueda que:

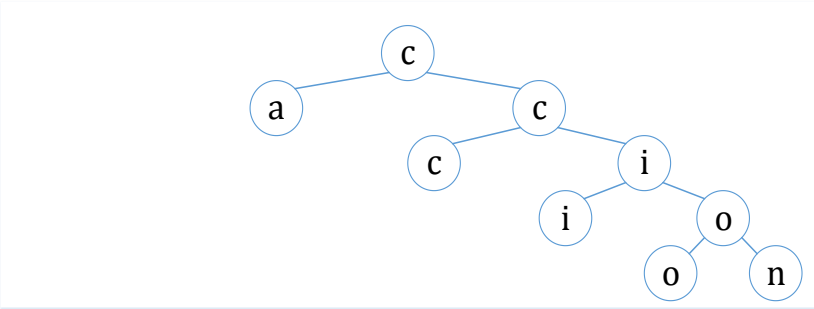
1. Los subárboles de cada nodo pueden contener valores iguales al valor del nodo (lo mismo pudieran estar a su izquierda que a su derecha).
2. Todo valor contenido en el árbol se encuentra además en exactamente un nodo hoja.

Los ejemplos de las figuras a continuación cumplen con esta definición

Ejemplo 1.



Ejemplo 2.



Implementación

Usted debe implementar dicha estructura de datos `ABBE` con las siguientes operaciones:

```
class ABBE<T> where T: IComparable<T>
{
    /// <summary>
    /// Crea una nueva instancia de un Árbol Binario de Búsqueda Extendido.
    /// </summary>
    public ABBE() { … }

    /// <summary>
    /// Inserta un nuevo valor en el árbol garantizando que todos los nodos del árbol
mantengan su valor.
    /// </summary>
    public void Inserta(T valor) { … }

    /// <summary>
    /// Un iterador que devuelve cada valor diferente contenido en el árbol y la cantidad
de veces que se repite.
    /// </summary>
    public IEnumerable<Frecuencia<T>> FrecuenciaPorValor() { … }
}
```

Donde la clase `Frecuencia` representa al par `Valor` y `Cantidad`:

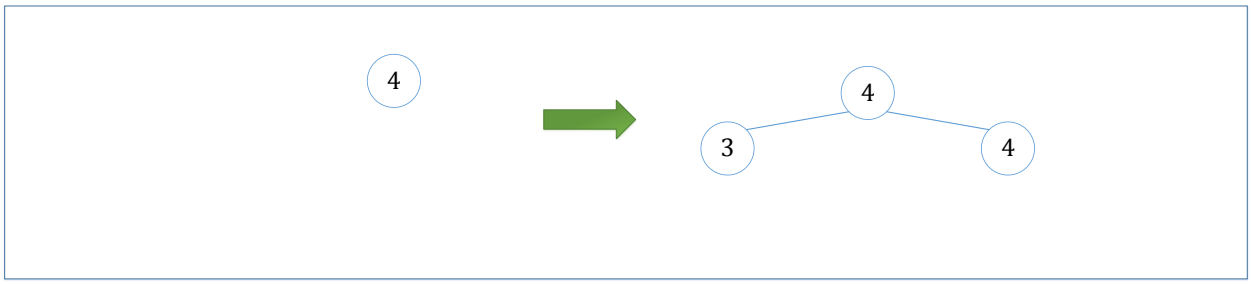
```
class Frecuencia<T>
{
    public Frecuencia(T valor, int cantidad) { ... }

    /// <summary>
    /// Un valor contenido en el árbol.
    /// </summary>
    public T Valor { get; private set; }

    /// <summary>
    /// Cantidad de veces que se repite el valor en al árbol.
    /// </summary>
    public int Cantidad { get; private set; }
}
```

Ejemplo

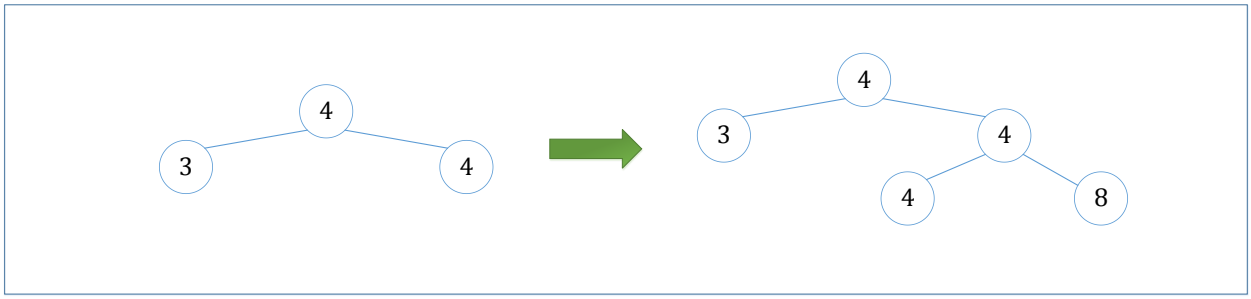
El árbol del ejemplo 1 se obtiene a partir de las siguientes inserciones.

```
ABBE<int> t = new ABBE<int>();
t.Inserta(4);
t.Inserta(3);
t.Inserta(8);
t.Inserta(5);
t.Inserta(2);
```

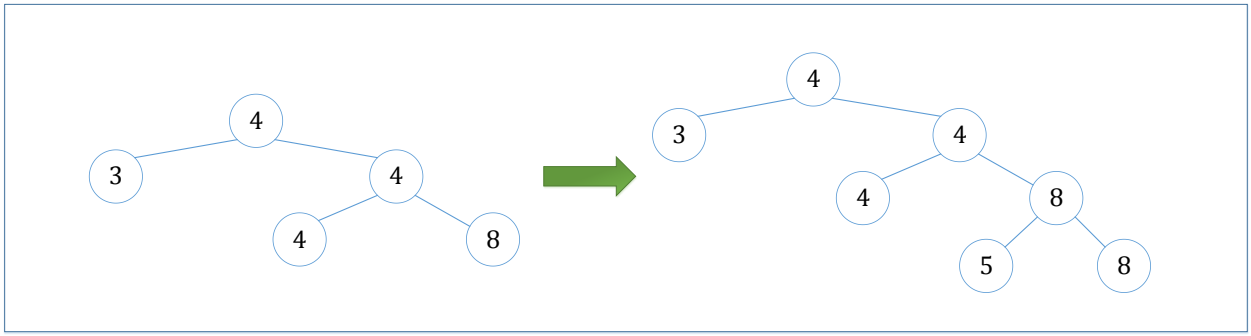
A continuación se muestra como queda el árbol binario de búsqueda extendido después de cada inserción.



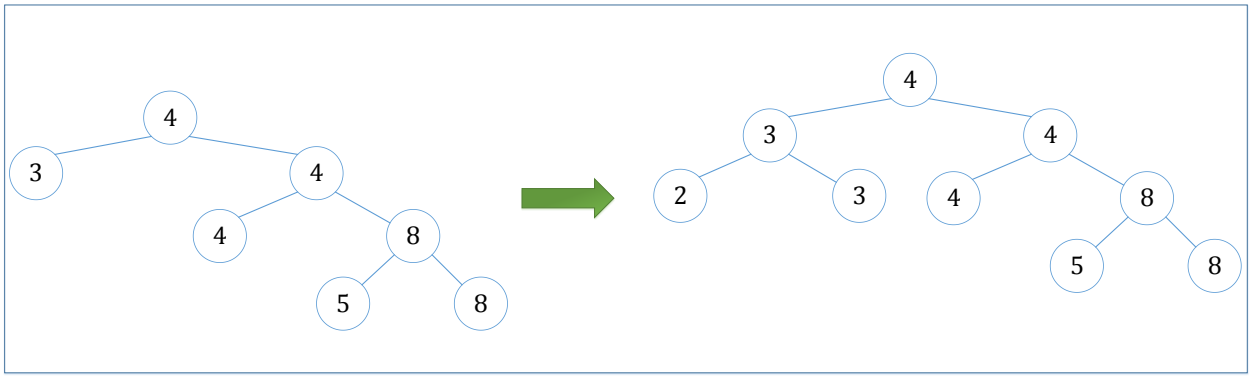
Después de Insertar “3” (el nodo 4 se tiene que repetir para que siga siendo hoja)



Después de Insertar "8" (el nodo 4 se vuelve a repetir para que siga siendo hoja)



Después de Insertar "5" (el nodo 8 se tiene que repetir para que siga siendo hoja)



Después de Insertar "2" (el nodo 3 se tiene que repetir para que siga siendo hoja)

Nótese que después de cada inserción el árbol mantiene su misma estructura, es decir, no se ha reacomodado a ningún nodo.

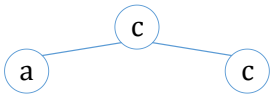
Si en este árbol aplicamos el método `FrecuenciaPorValor` se obtendría la salida:

```
foreach (var f in t.FrecuenciaPorValor())
    Console.WriteLine("{0}: {1}", f.Valor, f.Cantidad);

// 2: 1
// 3: 2
// 4: 3
// 5: 1
// 8: 2
```

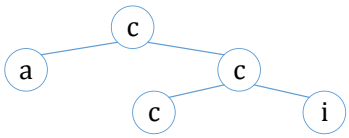
Igualmente se puede construir el árbol del ejemplo 2:

```
ABBE<string> t = new ABBE<string>();

t.Inserta("c");
foreach (var f in t.FrecuenciaPorValor())
    Console.WriteLine("{0}: {1}", f.Valor, f.Cantidad);

// c: 1
```

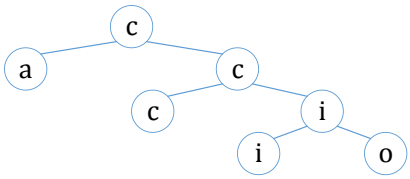

```
t.Inserta("a");
foreach (var f in t.FrecuenciaPorValor())
    Console.WriteLine("{0}: {1}", f.Valor, f.Cantidad);

// a: 1
// c: 2
```



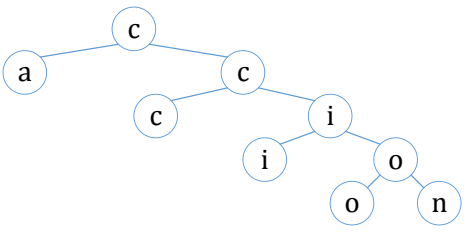

```
t.Inserta("i");
foreach (var f in t.FrecuenciaPorValor())
    Console.WriteLine("{0}: {1}", f.Valor, f.Cantidad);

// a: 1
// c: 3
// i: 1
```




```
t.Inserta("o");
foreach (var f in t.FrecuenciaPorValor())
    Console.WriteLine("{0}: {1}", f.Valor, f.Cantidad);

// a: 1
// c: 3
// i: 2
// o: 1
```




```
t.Inserta("n");
foreach (var f in t.FrecuenciaPorValor())
```



```
    Console.WriteLine("{0}: {1}", f.Valor, f.Cantidad);

// a: 1
// c: 3
// i: 2
// o: 2
// n: 1
```