

Especificación BSTree (Nereus)

Árbol binario de búsqueda como subtipo algebraico de BTree. Versión con axioma base de BSTree_Insert corregido para permitir reescritura iterativa.

1. Especificación

```

CLASS BTree [elem]
BASIC CONSTRUCTORS BTree_Empty, BTree_Create
EFFECTIVE
TYPE BTree
OPERATIONS
  BTree_Empty: -> BTree;
  BTree_Create: BTree * BTree * elem -> BTree;
  BTree_IsEmpty: BTree -> Boolean;
  BTree_Root: BTree(t) -> elem      pre: not BTree_IsEmpty(t);
  BTree_Left: BTree(t) -> BTree     pre: not BTree_IsEmpty(t);
  BTree_Right: BTree(t) -> BTree     pre: not BTree_IsEmpty(t);
  belongs: elem * BTree -> Boolean;

```

```

AXIOMS t1, t2: BTree; e, e1: elem;

```

```

  BTree_IsEmpty(BTree_Empty())          = True;
  BTree_IsEmpty(BTree_Create(t1, t2, e)) = False;
  BTree_Root (BTree_Create(t1, t2, e))   = e;
  BTree_Left (BTree_Create(t1, t2, e))   = t1;
  BTree_Right (BTree_Create(t1, t2, e))  = t2;
  belongs(e, BTree_Empty())              = False;
  belongs(e, BTree_Create(t1, t2, e1)) =
    (e = e1) or (belongs(e, t1) or belongs(e, t2));

```

```

END-CLASS

```

```

CLASS BSTree [Element : WithOrder]
IS-SUBTYPE-OF BTree[Element]
BASIC CONSTRUCTORS BSTree_Insert
EFFECTIVE
OPERATIONS
  BSTree_Insert: BSTree(t) * Element(e) -> BSTree  pre: not belongs(e, t);
  BSTree_Delete: BSTree(t) * Element(e) -> BSTree  pre: not BTree_IsEmpty(t);
  BSTree_Min: BSTree(t) -> Element pre: not BTree_IsEmpty(t);

```

```

AXIOMS b1, b2: BSTree; i, j: Element;

```

```

  BSTree_Insert(BTree_Empty() as BSTree, i) =
    BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, i) as BSTree;

```

```

  (i < j) implies
    BSTree_Insert(BTree_Create(b1, b2, j) as BSTree, i) =
      BTree_Create(BSTree_Insert(b1, i), b2, j) as BSTree;

```

```

  (i > j) implies
    BSTree_Insert(BTree_Create(b1, b2, j) as BSTree, i) =
      BTree_Create(b1, BSTree_Insert(b2, i), j) as BSTree;

```

```

  BTree_IsEmpty(b1) implies

```

```

    BSTree_Min(BTree_Create(b1, b2, i) as BSTree) = i;

not BTree_IsEmpty(b1) implies
    BSTree_Min(BTree_Create(b1, b2, i) as BSTree) = BSTree_Min(b1);

(j < i) implies
    BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
        BTree_Create(BSTree_Delete(b1, j), b2, i) as BSTree;

(j > i) implies
    BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
        BTree_Create(b1, BSTree_Delete(b2, j), i) as BSTree;

(j = i and BTree_IsEmpty(b1) and BTree_IsEmpty(b2)) implies
    BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) = BTree_Empty() as BSTree;

(j = i and BTree_IsEmpty(b1) and not BTree_IsEmpty(b2)) implies
    BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) = b2;

(j = i and not BTree_IsEmpty(b1) and BTree_IsEmpty(b2)) implies
    BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) = b1;

(j = i and not BTree_IsEmpty(b1) and not BTree_IsEmpty(b2)) implies
    BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
        BTree_Create(b1,
            BSTree_Delete(b2, BSTree_Min(b2)),
            BSTree_Min(b2)) as BSTree;

```

END-CLASS

Por qué se corrigió el axioma base de `BSTree_Insert`: los hijos vacíos en el RHS deben llevar as `BSTree` explícito. Sin el casteo, al reescribir iterativamente, el axioma recursivo `BSTree_Insert(BTree_Create(b1, b2, j) as BSTree, i)` no puede ligar `b1, b2` (declarados `BSTree`) a un `BTree_Empty()` "pelado". Con el casteo, cada subárbol producido por el caso base ya es `BSTree` y la reescritura continúa.

2. Traza de reescritura

Término a evaluar:

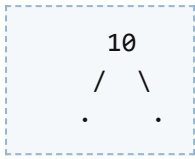
```

BSTree_Delete(
    BSTree_Insert(
        BSTree_Insert(
            BSTree_Insert(BTree_Empty() as BSTree, 10),
            8),
        11),
    10)

```

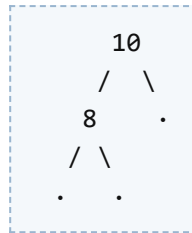
Estado del árbol que vamos construyendo (semánticamente):

Después de Insert 10:



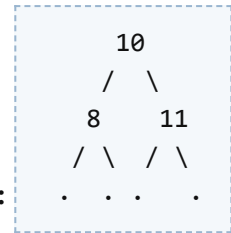
Después de Insert 8 (8<10,

izquierda):



Después de Insert 11 (11>10,

derecha):



(• = BTree_Empty() as BSTree)

Paso 1 — caso base de Insert sobre el más interno

Aplicar: $BSTree_Insert(BTree_Empty() \text{ as } BSTree, i) = BTree_Create(BTree_Empty() \text{ as } BSTree, BTree_Empty() \text{ as } BSTree, i) \text{ as } BSTree$ con $i = 10$

```
BSTree_Delete(  
  BSTree_Insert(  
    BSTree_Insert(  
      BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 10) as  
      BSTree,  
      8),  
      11),  
      10)
```

Paso 2 — Insert 8 con $i < j$ (8<10), aplicar rama izquierda

Aplicar: $(i < j) \text{ implies } BSTree_Insert(BTree_Create(b1, b2, j) \text{ as } BSTree, i) = BTree_Create(BSTree_Insert(b1, i), b2, j) \text{ as } BSTree$ con $b1 = b2 = BTree_Empty() \text{ as } BSTree, j = 10, i = 8$

```
BSTree_Delete(  
  BSTree_Insert(  
    BTree_Create(  
      BSTree_Insert(BTree_Empty() as BSTree, 8),  
      BTree_Empty() as BSTree,  
      10) as BSTree,  
      11),  
      10)
```

Paso 3 — reapiro caso base sobre el Insert interior

```
BSTree_Delete(  
  BSTree_Insert(  
    BTree_Create(  
      BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 8) as BSTree,  
      BTree_Empty() as BSTree,  
      10) as BSTree,
```

```
11),  
10)
```

Paso 4 — Insert 11 con $i > j$ ($11 > 10$), aplicar rama derecha

Aplicar: $(i > j)$ implies $BSTree_Insert(BTree_Create(b1, b2, j) \text{ as } BSTree, i) = BTree_Create(b1, BSTree_Insert(b2, i), j) \text{ as } BSTree$
con $b1 = BTree_Create(\cdot, \cdot, 8) \text{ as } BSTree, b2 = BTree_Empty() \text{ as } BSTree, j = 10, i = 11$

```
BSTree_Delete(  
  BTree_Create(  
    BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 8) as BSTree,  
    BSTree_Insert(BTree_Empty() as BSTree, 11),  
    10) as BSTree,  
  10)
```

Paso 5 — caso base de Insert sobre el subárbol derecho

```
BSTree_Delete(  
  BTree_Create(  
    BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 8) as BSTree,  
    BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 11) as BSTree,  
    10) as BSTree,  
  10)
```

Paso 6 — Delete con $j=i$ y ambos hijos no vacíos

Sea $b1 = BTree_Create(\cdot, \cdot, 8) \text{ as } BSTree$ (no vacío) y $b2 = BTree_Create(\cdot, \cdot, 11) \text{ as } BSTree$ (no vacío); $i = j = 10$.

Aplicar: $(j = i \text{ and not } BTree_IsEmpty(b1) \text{ and not } BTree_IsEmpty(b2))$ implies
 $BSTree_Delete(BTree_Create(b1, b2, i) \text{ as } BSTree, j) = BTree_Create(b1, BSTree_Delete(b2, BTree_Min(b2)), BTree_Min(b2)) \text{ as } BSTree$

```
BTree_Create(  
  BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 8) as BSTree,  
  BSTree_Delete(b2, BTree_Min(b2)),  
  BTree_Min(b2)  
) as BSTree
```

Paso 7 — calcular $BSTree_Min(b2)$

Aplicar: $BTree_IsEmpty(b1)$ implies $BSTree_Min(BTree_Create(b1, b2, i) \text{ as } BSTree) = i$
Sobre $BTree_Create(BTree_Empty() \text{ as } BSTree, BTree_Empty() \text{ as } BSTree, 11) \text{ as } BSTree$: el hijo izquierdo es vacío, entonces $BSTree_Min = 11$.

Paso 8 — calcular $BSTree_Delete(b2, 11)$

Sobre $BTree_Create(\cdot, \cdot, 11) \text{ as } BSTree$ con $j = i = 11$ y ambos hijos vacíos:

Aplicar: $(j = i \text{ and } BTree_IsEmpty(b1) \text{ and } BTree_IsEmpty(b2))$ implies $BSTree_Delete(...) =$

BTree_Empty() as *BSTree*

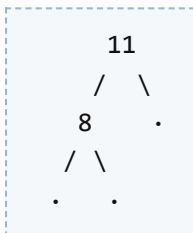
Paso 9 — sustituir resultados parciales

```
BTree_Create(  
  BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 8) as BSTree,  
  BTree_Empty() as BSTree,  
  11  
) as BSTree
```

Resultado final

```
BTree_Create(  
  BTree_Create(BTree_Empty() as BSTree, BTree_Empty() as BSTree, 8) as BSTree,  
  BTree_Empty() as BSTree,  
  11  
) as BSTree
```

Vista de árbol:



Interpretación: al borrar la raíz 10 con dos hijos no vacíos, el axioma reemplaza la raíz por el mínimo del subárbol derecho (11) y borra ese mínimo del subárbol derecho. Como el subárbol derecho era el nodo hoja 11, queda vacío. El resultado es un árbol con raíz 11 y único hijo izquierdo 8.