

Trazas de reescritura — Delete 10 y Delete 8

Dos términos que comparten la construcción del BST con tres Inserts y se diferencian solo en el elemento que se borra. Spec: `BSTree-hide.nereus`.

Términos a evaluar:

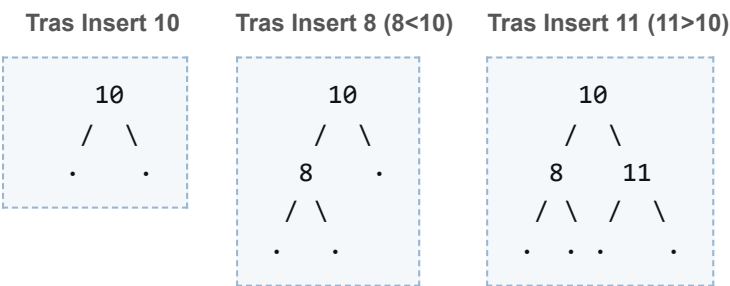
```
A) BSTree_Delete(BSTree_Insert(BSTree_Insert(BSTree_Insert(BTree_Empty() as BSTree,
10), 8), 11), 10)
B) BSTree_Delete(BSTree_Insert(BSTree_Insert(BSTree_Insert(BTree_Empty() as BSTree,
10), 8), 11), 8)
```

La parte interior (los tres Inserts) construye el mismo BST en ambos casos. La diferencia está en el segundo argumento del Delete: A borra la **raíz 10** (caso complejo con dos hijos no vacíos), B borra la **hoja 8** (caso simple: recurrir y eliminar nodo sin hijos).

0. Convenciones

- `BTree_Empty()` as `BSTree` en las vistas de árbol.
- El redex se resalta en **amarillo**.
- Los pasos comunes se numeran **A1–A5**; los del Delete 10 se numeran **B1–B4**; los del Delete 8 se numeran **C1–C3**.

Parte A — Construcción del BST (común a ambos términos)



Paso A1 — caso base de BSTree_Insert ($i = 10$)

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

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

(Uso k como variable simbólica para el segundo argumento del Delete: vale 10 en el caso A, 8 en el caso B.)

Paso A2 — Insert 8 con $i < j$ ($8 < 10$)

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

Bindings: $b1 = b2 = \text{BTree_Empty}() \text{ as BSTree}, j = 10, i = 8.$

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

Paso A3 — caso base de BSTree_Insert ($i = 8$)

```
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),  
  k)
```

Paso A4 — Insert 11 con $i > j$ ($11 > 10$)

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

Bindings: $b1 = \text{BTree_Create}(\cdot, \cdot, 8)$ as BSTree, $b2 = \text{BTree_Empty}()$ 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,
  k)
```

Paso A5 — caso base de BSTree_Insert ($i = 11$)

```
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,
  k)
```

Llamemos T a este árbol completamente expandido. Desde aquí cada caso de Delete avanza por su rama propia.

Parte B — Borrar 10 (raíz con dos hijos no vacíos)

Sustituyendo $k = 10$ en τ :

```
BSTree_Delete(T, 10)
donde T = BTree_Create(
  BTree_Create( $\cdot$ ,  $\cdot$ , 8) as BSTree,
  BTree_Create( $\cdot$ ,  $\cdot$ , 11) as BSTree,
  10) as BSTree
```

Paso B1 — BSTree_Delete caso F (j = i, ambos hijos no vacíos)

```
(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 as BSTree, BSTree_Min(b2 as BSTree)),
    BSTree_Min(b2 as BSTree)
  ) as BSTree;
```

b1 = BTree_Create(·, ·, 8) as BSTree → no vacío ✓
b2 = BTree_Create(·, ·, 11) as BSTree → no vacío ✓
i = 10, j = 10 → j = i ✓

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

Paso B2 — calcular BSTree_Min(b2)

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

Sobre BTree_Create(·, ·, 11) as BSTree: hijo izquierdo vacío ✓
Resultado: BSTree_Min(b2) = 11

Paso B3 — calcular BSTree_Delete(b2, 11) (caso C: j = i, ambos hijos vacíos)

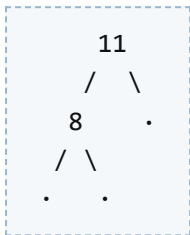
```
(j = i and BTree_IsEmpty(b1) and BTree_IsEmpty(b2)) implies
BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
  BTree_Empty() as BSTree;
```

Sobre BSTree_Delete(BTree_Create(·, ·, 11) as BSTree, 11):
ambos hijos vacíos, i = j = 11 ✓
Resultado: BTree_Empty() as BSTree

Paso B4 — sustituir y obtener la forma normal

```
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:



La raíz 10 fue reemplazada por el mínimo del subárbol derecho (11), y ese 11 quedó borrado de la derecha. La invariante BST se preserva: $8 < 11$.

Parte C — Borrar 8 (hijo izquierdo, hoja sin descendencia)

Sustituyendo $k = 8$ en T:

```
BSTree_Delete(T, 8)  
donde T = BTree_Create(  
    BTree_Create(., ., 8) as BSTree,  
    BTree_Create(., ., 11) as BSTree,  
    10) as BSTree
```

Paso C1 — BSTree_Delete caso A ($j < i: 8 < 10$), recurrir en el subárbol izquierdo

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

```
b1 = BTree_Create(., ., 8) as BSTree  
b2 = BTree_Create(., ., 11) as BSTree  
i = 10, j = 8  $\rightarrow$  j < i  $\checkmark$ 
```

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

Paso C2 — calcular el BSTree_Delete interno (caso C: j = i, ambos hijos vacíos)

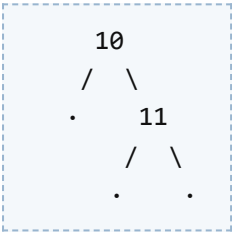
```
(j = i and BTree_IsEmpty(b1) and BTree_IsEmpty(b2)) implies
BSTree_Delete(BTree_Create(b1, b2, i) as BSTree, j) =
  BTree_Empty() as BSTree;
```

Sobre BSTree_Delete(BTree_Create(., ., 8) as BSTree, 8):
ambos hijos vacíos ✓, i = j = 8 ✓
Resultado: BTree_Empty() as BSTree

Paso C3 — sustituir y obtener la forma normal

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

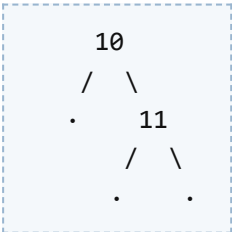
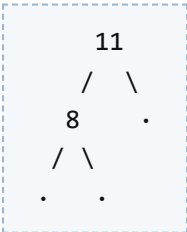
Vista de árbol:



8 era una hoja: bastó con reemplazarlo por BTree_Empty() as BSTree bajo la raíz 10. La invariante BST se preserva trivialmente.

Comparación de los dos casos

Forma normal de A — Delete 10 Forma normal de B — Delete 8



	Caso A — Delete 10	Caso B — Delete 8
Tipo de nodo	raíz con dos hijos no vacíos	hoja
Axioma terminal	caso F → necesita BSTree_Min + sub-Delete	caso C directamente
Pasos del Delete	4 (B1–B4)	3 (C1–C3)

Cambia la raíz	Sí (10 → 11)	No (queda 10)
Operaciones auxiliares	BSTree_Min, BSTree_Delete recursivo	solo BSTree_Delete recursivo

Observación: ambos términos reducen a una forma normal construida exclusivamente con los constructores básicos públicos de la jerarquía (BTree_Empty, BSTree_Insert) más BTree_Create en su rol interno. La cláusula HIDE BTree_Create garantiza que un cliente solo pueda producir términos equivalentes a éstos a través de la API pública (composición de Inserts y Delete), nunca construyendo árboles arbitrarios que pudieran violar la invariante BST.

Trazas generadas para la spec BSTree-hide.nereus. Caso A: 5 pasos comunes + 4 propios = 9 pasos. Caso B: 5 pasos comunes + 3 propios = 8 pasos.