Árboles RN

Estructuras de Datos

Andrea Rueda

Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas

¿Cómo garantizar árboles “bonitos”?

- Balanceando:
  - Evitar “listas”.
  - Evitar ramas cortas.
- Garantizar búsqueda / inserción / eliminación en O(log n).
- Árboles AVL y RN.

Árboles RN
(Rojo-Negro)
(*Red-Black Trees*)

Árboles RN

- Árboles Rojo-Negro (*Red-Black trees*):

  Inicialmente conocido como Árbol-B binario simétrico.

  Guibas, Leo J., and Robert Sedgewick. "A dichromatic framework for balanced trees." 19th Annual Symposium on Foundations of Computer Science. IEEE, 1978.

Árboles RN

- Árboles Rojo-Negro (*Red-Black trees*):

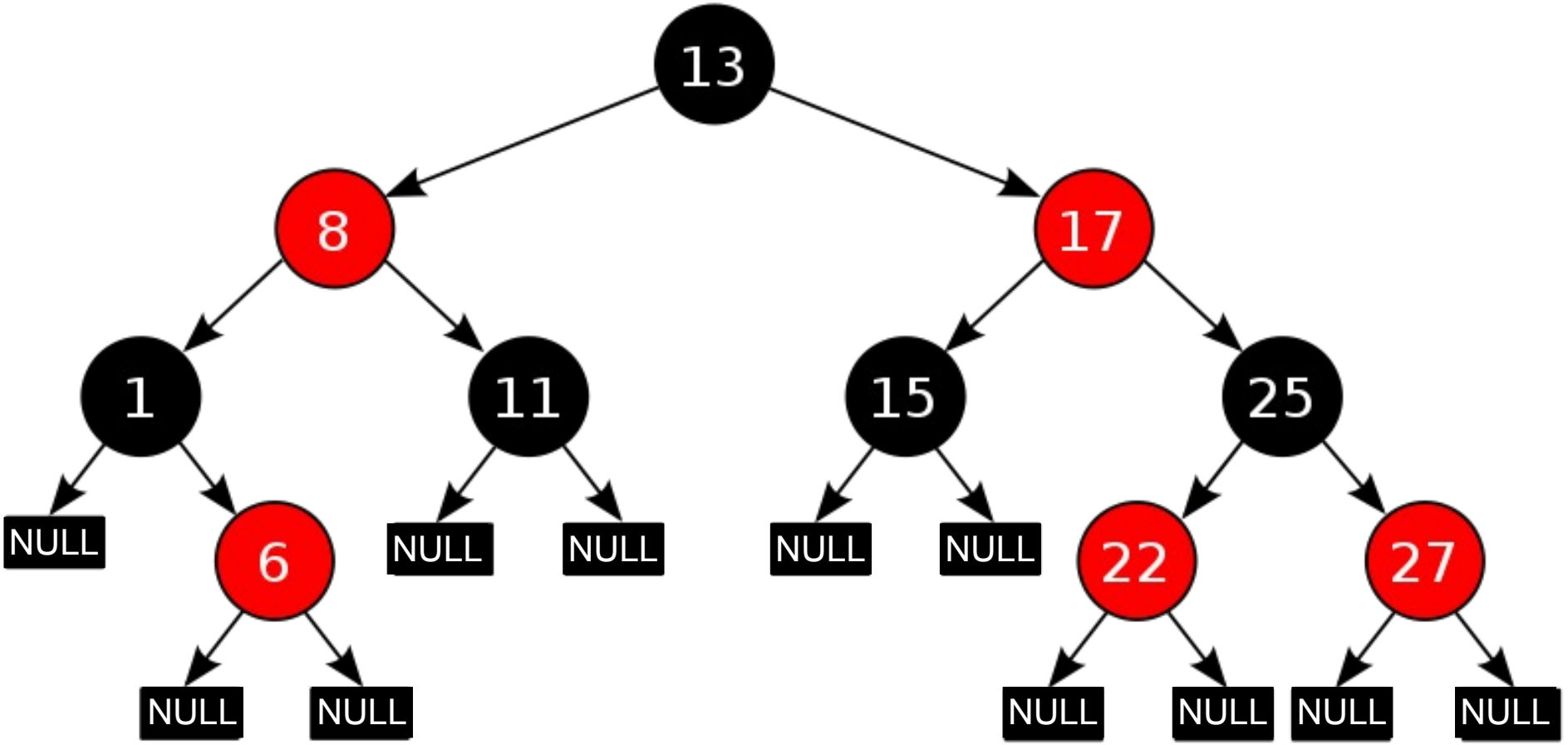
  Como los árboles AVL, garantiza que las operaciones de búsqueda, inserción y eliminación en un árbol binario ordenado toman en el peor caso O(log n).

  Requiere que el nodo del árbol incluya una propiedad adicional de color (**rojo** o **negro**).

Árboles RN

- Propiedades a garantizar:
  1. Cada nodo es **rojo** o **negro**.
  2. La raíz y las hojas (nulos) son siempre **negras**.
  3. Si un nodo es **rojo**, entonces su padre es **negro**.
  4. Cada nodo **rojo** debe tener dos hijos **negros**.
  5. Todas las rutas desde un nodo x hacia un descendiente hoja tienen el mismo número de nodos **negros** (altura-negra(x)).

Árboles RN


13
8
17
1
11
15
25
NULL
6
NULL
NULL
NULL
NULL
22
27
NULL
NULL
NULL
NULL
NULL
NULL

Árboles RN

- Inserción:

  Se realiza como en un árbol binario ordenado.

  ¿Cuál debe ser el color del nuevo nodo?

  - **Negro**

Árboles RN

- Inserción:

  Se realiza como en un árbol binario ordenado.

  ¿Cuál debe ser el color del nuevo nodo?

  - **Negro**, causa una diferencia en el número de nodos negros en una ruta, incumpliendo la propiedad 5, y es una situación más difícil de corregir.

Árboles RN

- Inserción:

  Se realiza como en un árbol binario ordenado.

  ¿Cuál debe ser el color del nuevo nodo?

  - Entonces siempre lo pintaremos **rojo**.

Árboles RN

- Inserción:

  Se realiza como en un árbol binario ordenado.

  ¿Cuál debe ser el color del nuevo nodo?

  - Entonces siempre lo pintaremos **rojo**.

  ¿Y si el padre del nodo es **rojo**?

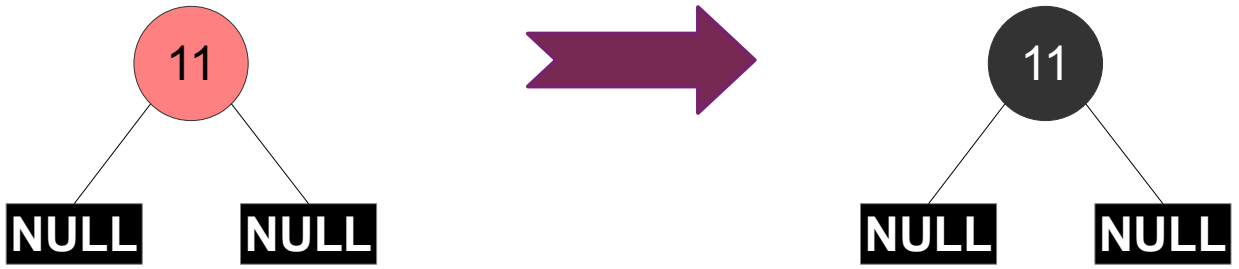
  - Se incumple la propiedad 3, pero se puede arreglar con rotaciones o cambiando el color de los ancestros.

Árboles RN

- Inserción:

  <u>Caso 1:</u> el nodo insertado es la raíz del árbol (primera inserción en un árbol vacío).

  - en este caso, el nodo se repinta a **negro**, para cumplir la propiedad 2.

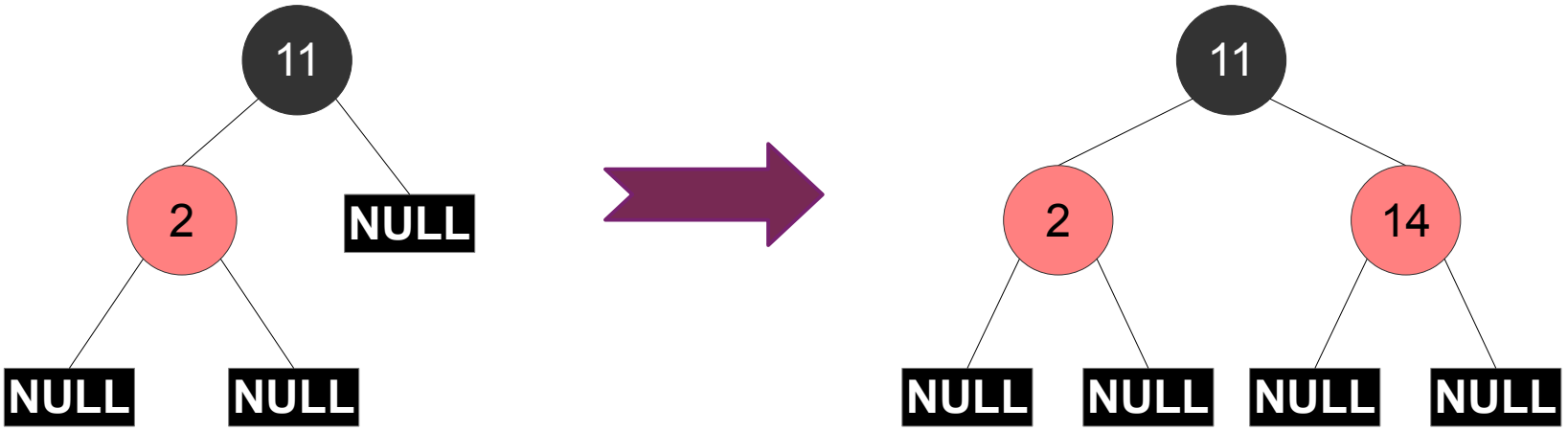


Árboles RN

- Inserción:

  <u>Caso 2:</u> el padre del nodo insertado es **negro**.

  - en este caso, el árbol es válido sin ninguna modificación.



Árboles RN

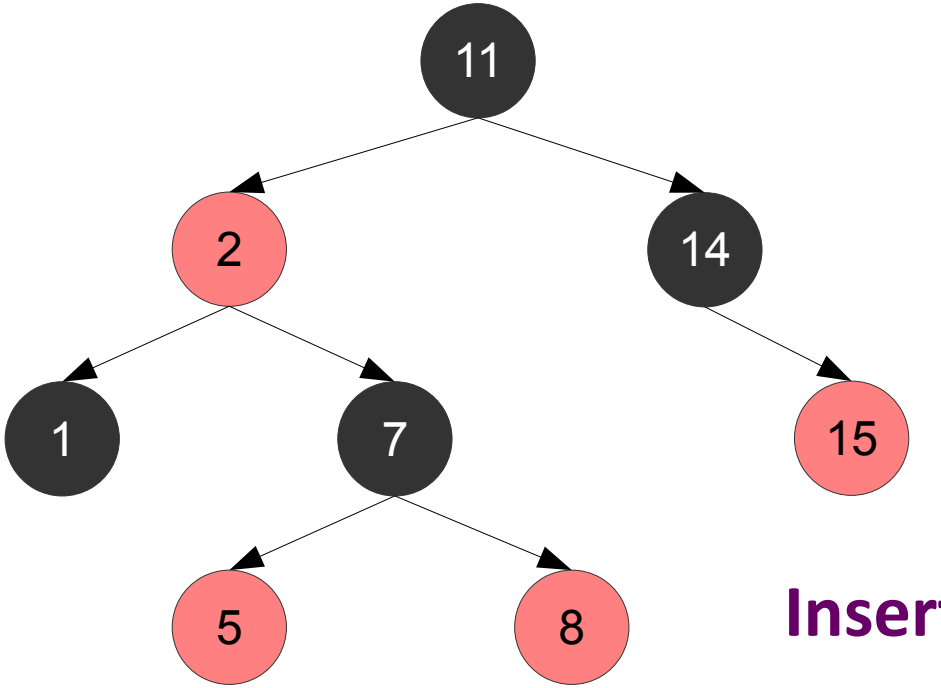
- Inserción:

  Caso 3: el padre del nodo insertado es **rojo**, se analizan varias situaciones:

  - **Situación 1:** Que el tío del nodo (hermano del padre) sea **rojo**.

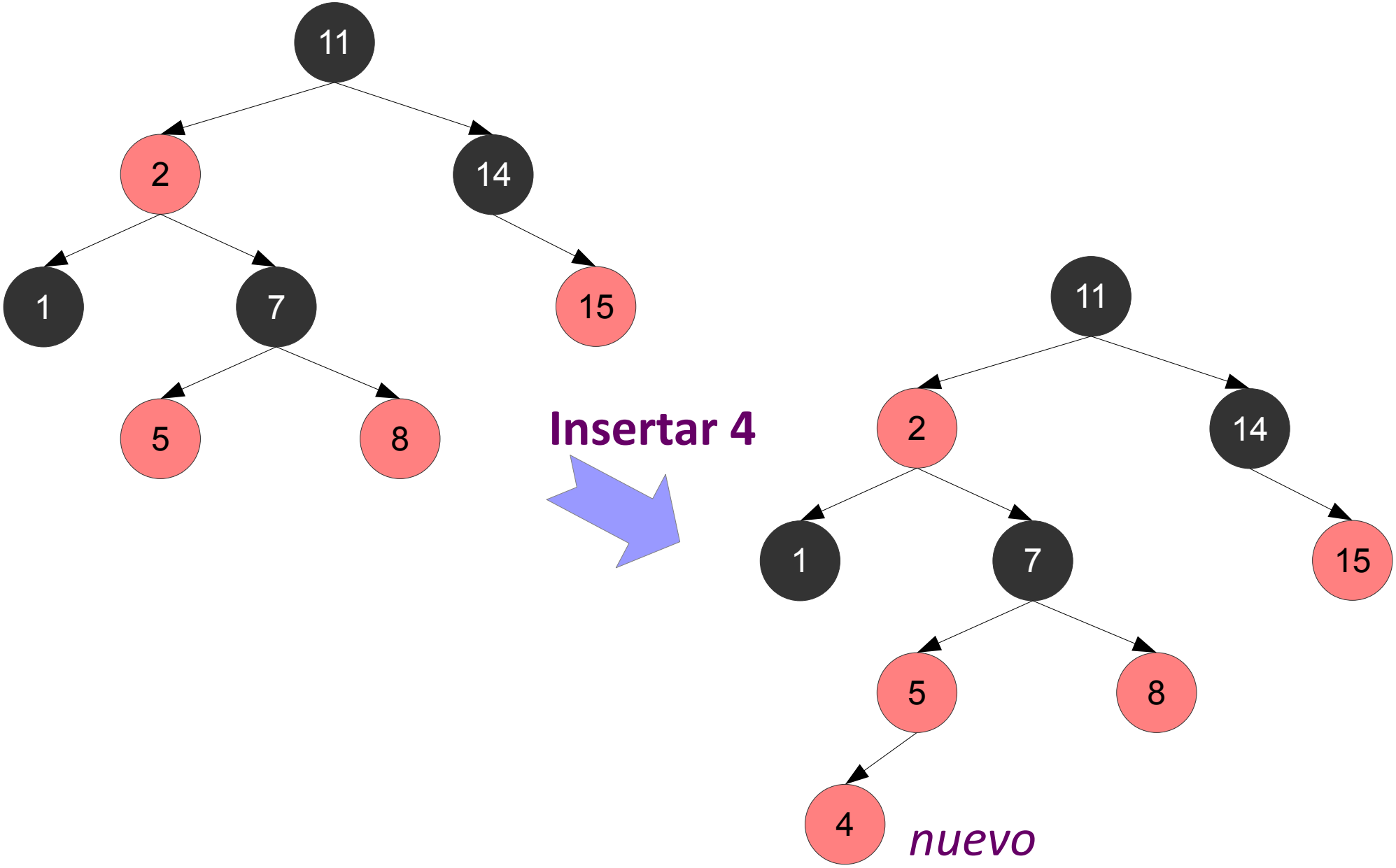
    En ese caso, se cambia el color del padre, del tío y del abuelo.

Árboles RN

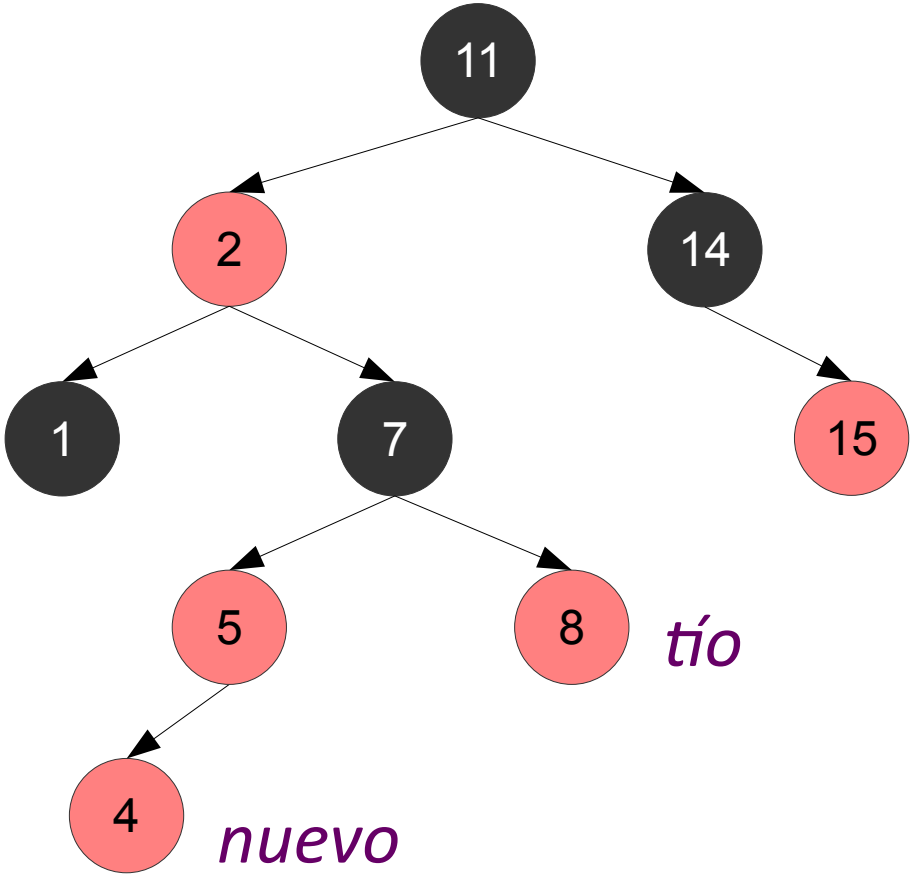


Insertar 4

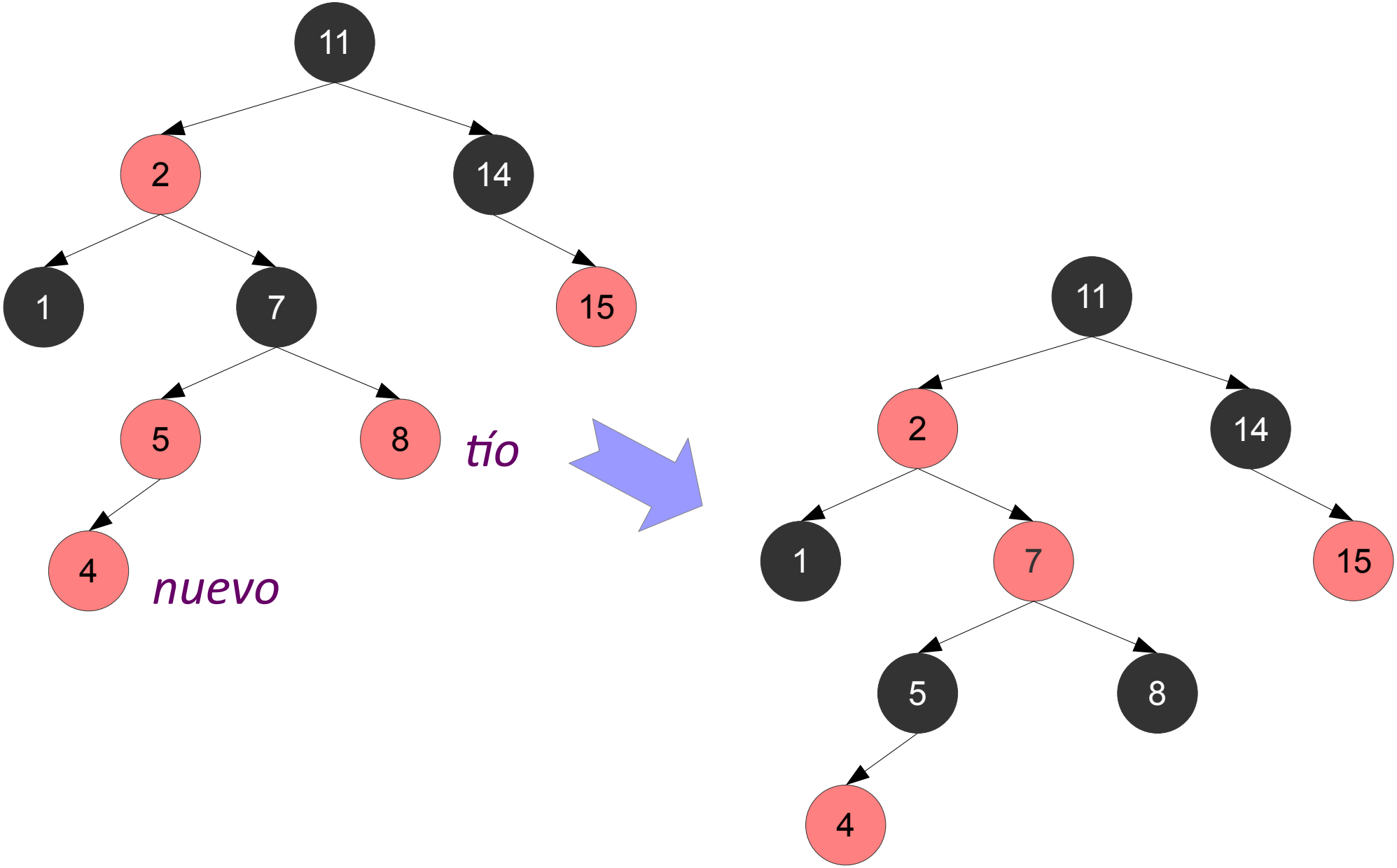
Árboles RN



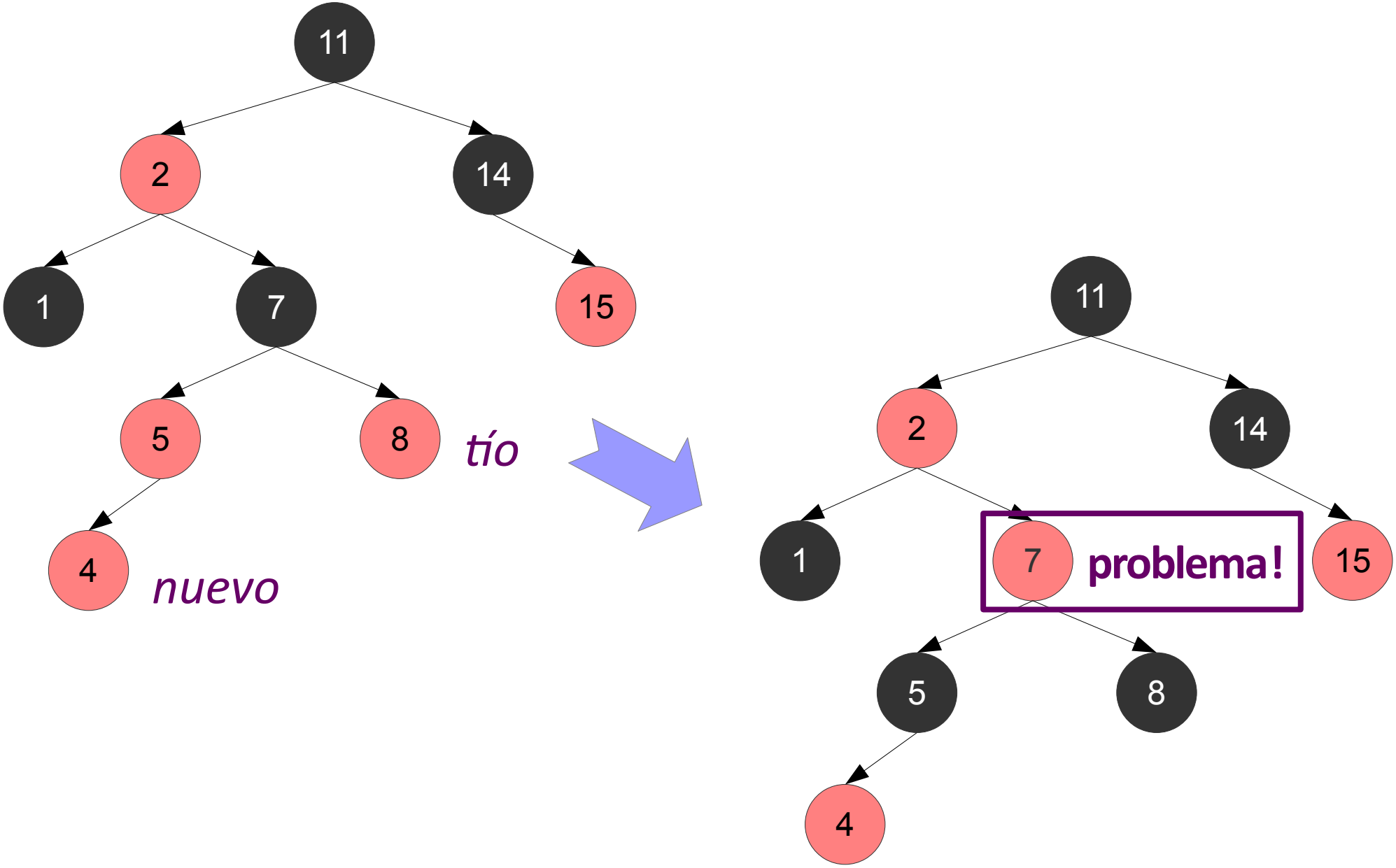
Árboles RN



Árboles RN



Árboles RN



Árboles RN

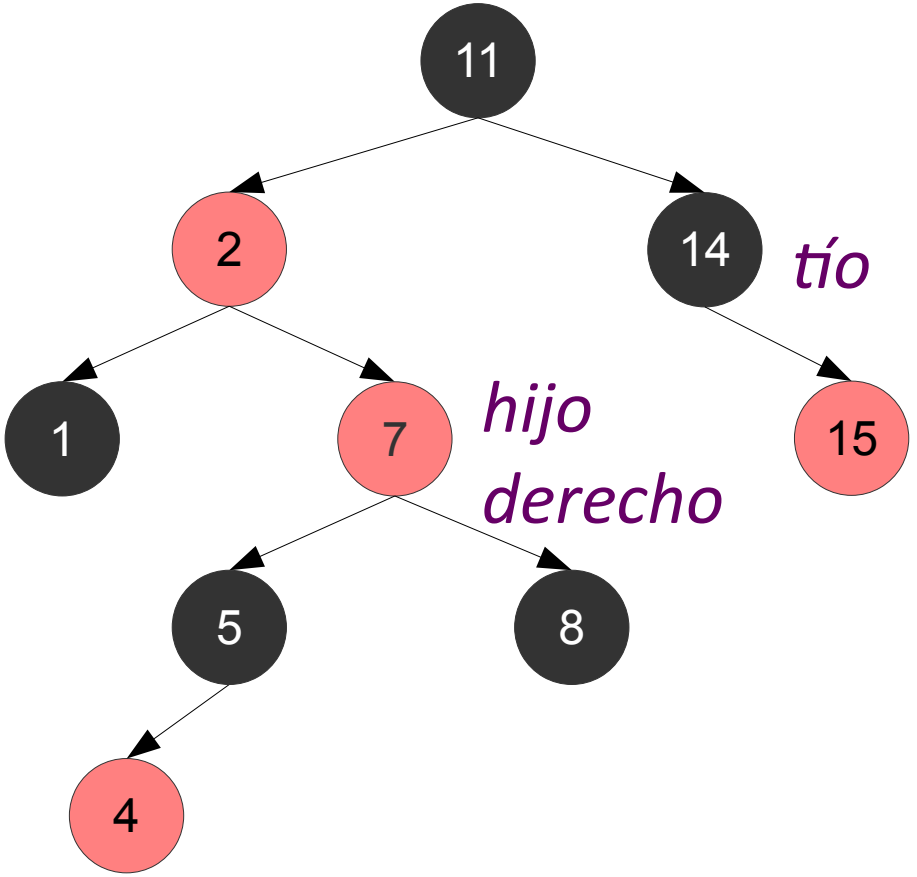
- Inserción

  Caso 3: el padre del nodo insertado es **rojo**, se analizan varias situaciones:

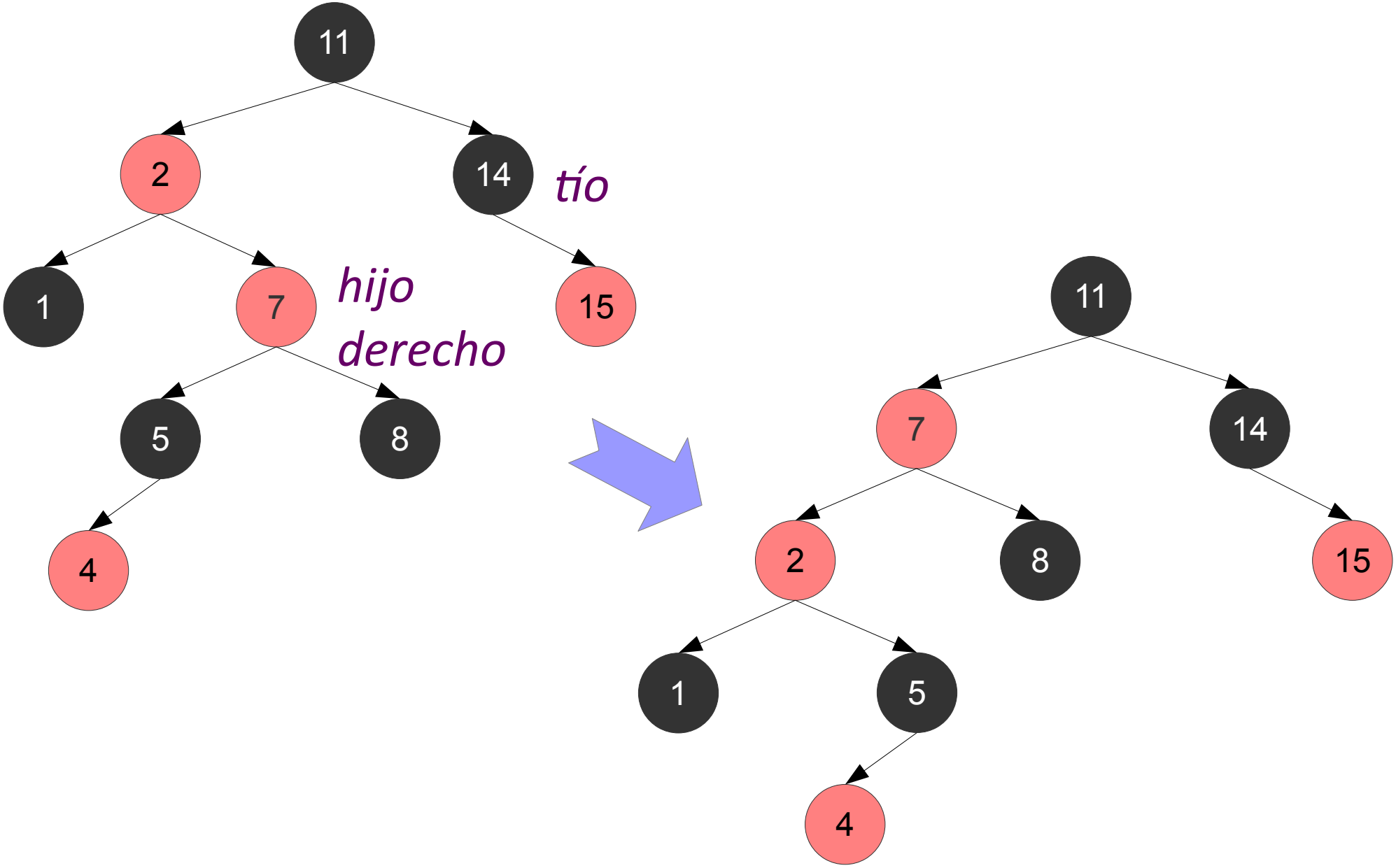
  - **Situación 2:** Que el tío del nodo (hermano del padre) sea **negro** y el nodo sea el hijo derecho del padre.

    En ese caso, se aplica una rotación a izquierda sobre el nodo.

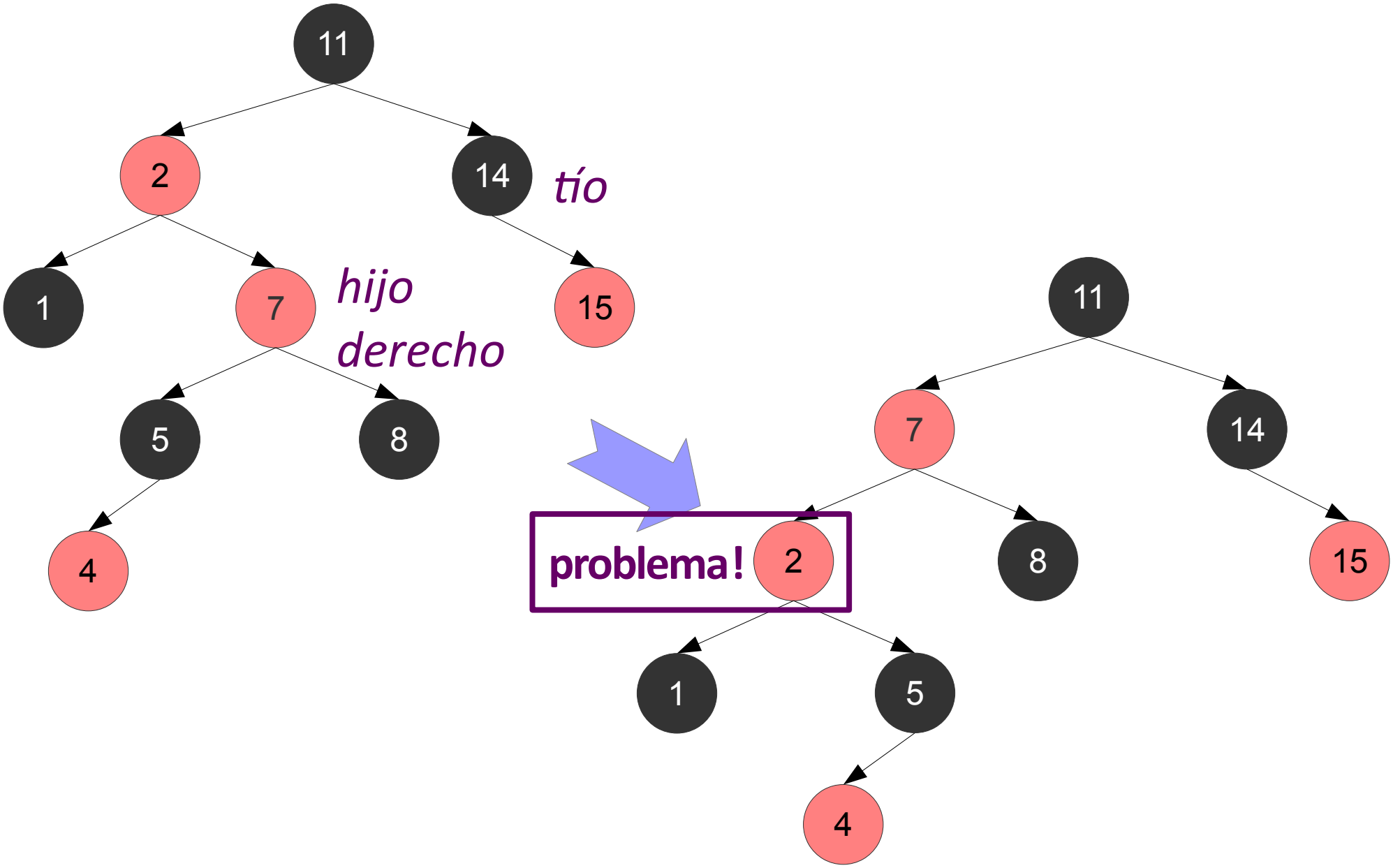
Árboles RN



Árboles RN



Árboles RN



Árboles RN

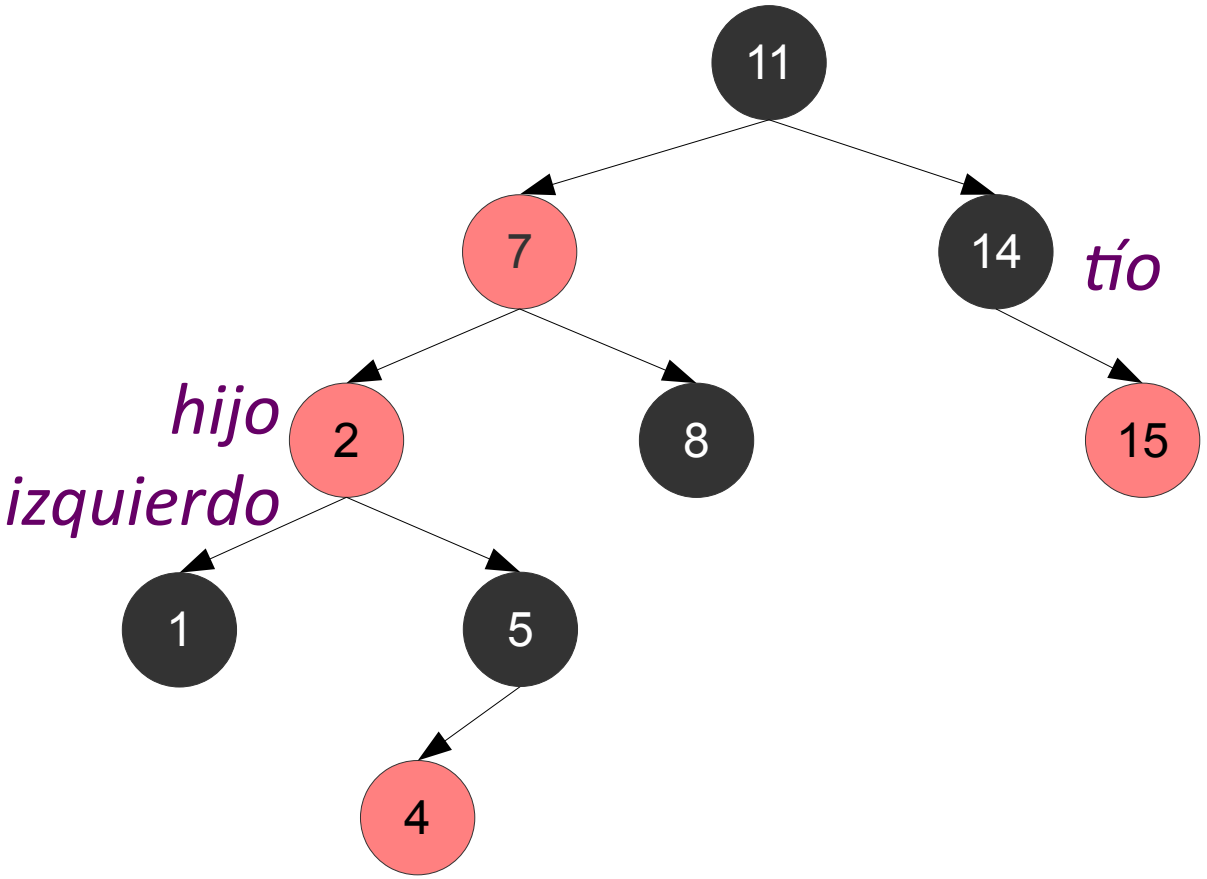
- Inserción

  Caso 3: el padre del nodo insertado es **rojo**, se analizan varias situaciones:

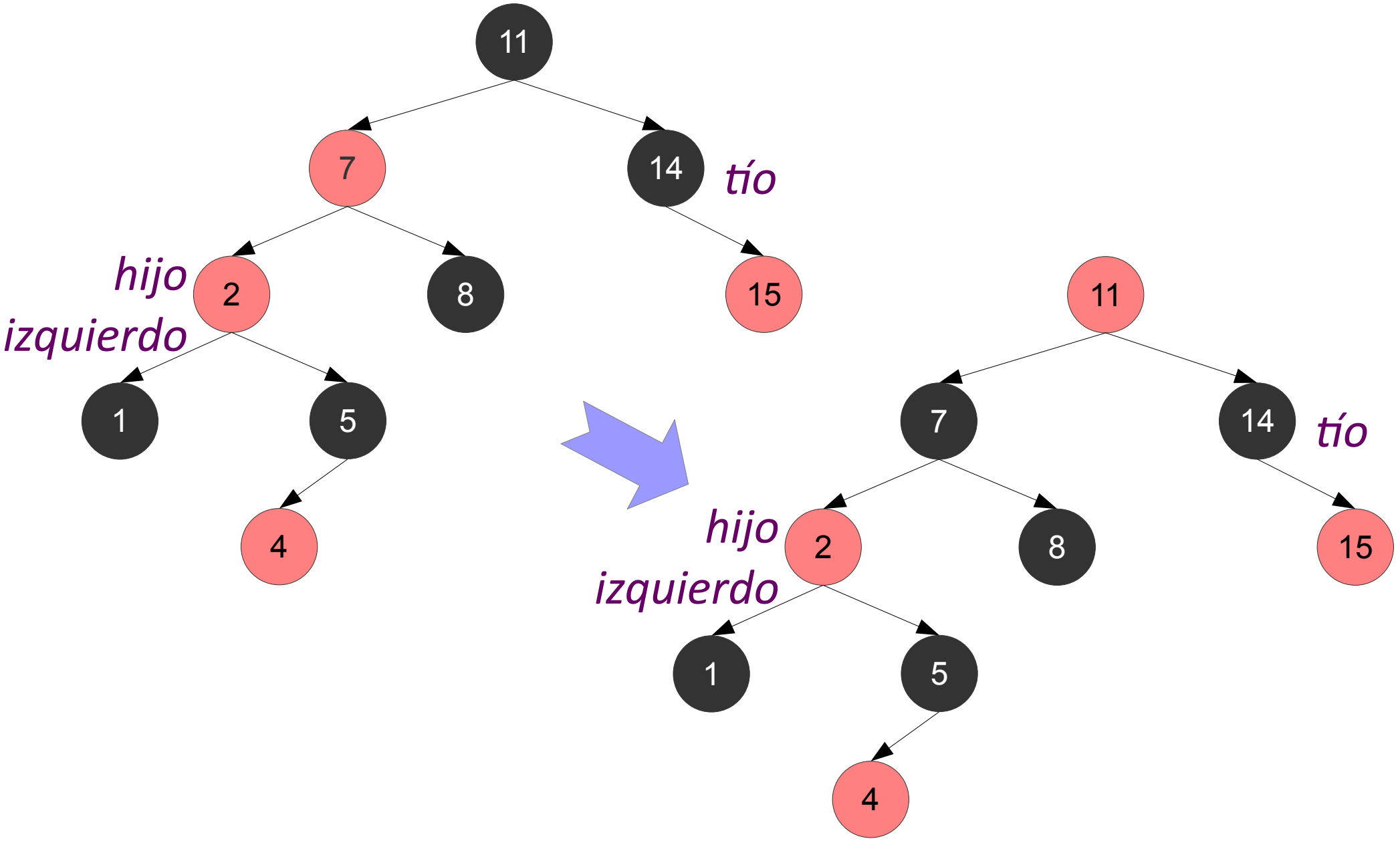
  - **Situación 3:** Que el tío del nodo (hermano del padre) sea **negro** y el nodo sea el hijo izquierdo del padre.

    En ese caso, se cambia el color del padre y del abuelo, y luego se aplica una rotación a derecha sobre el abuelo.

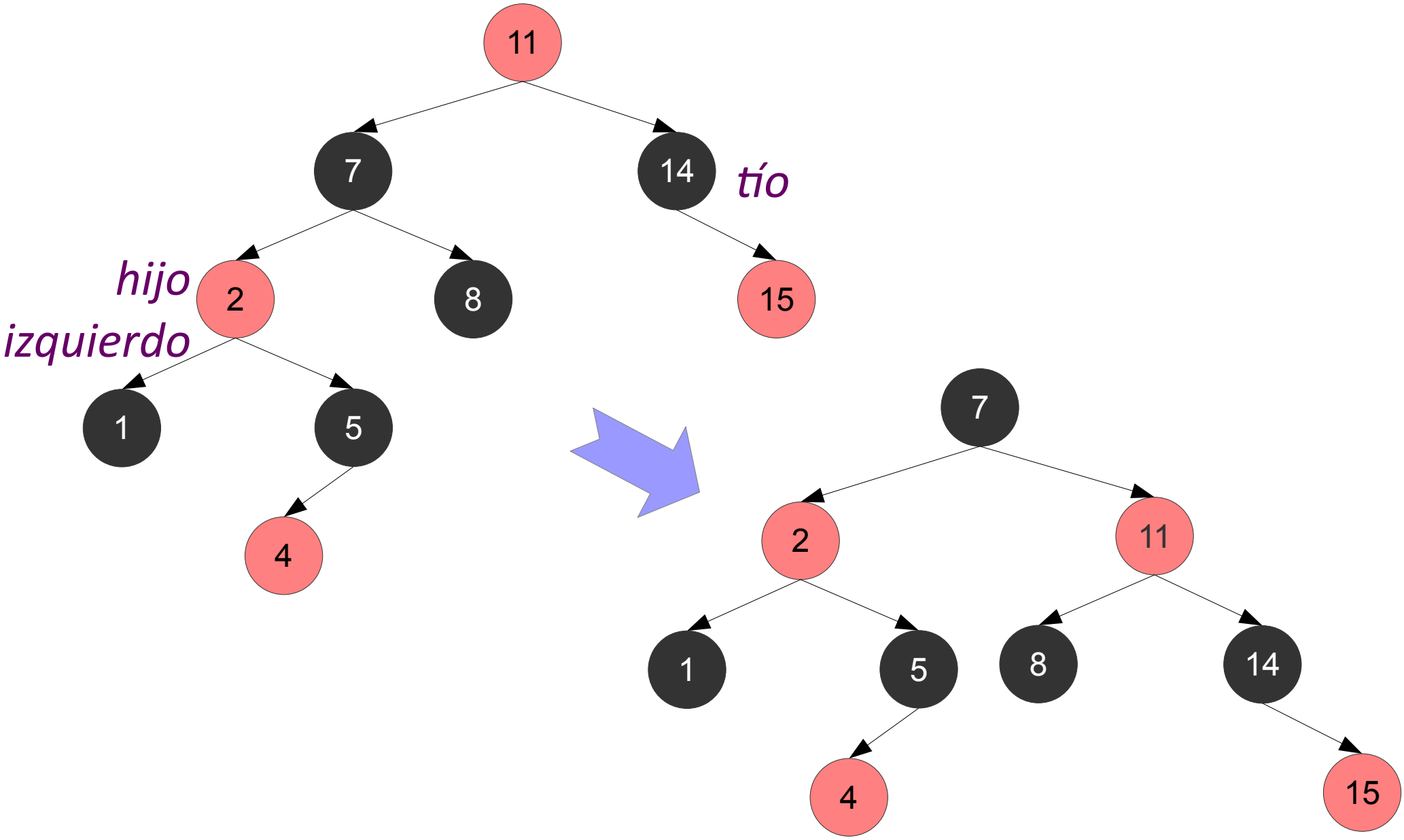
Árboles RN



Árboles RN


11
7
14
tío
hijo
izquierdo
2
8
15
1
5
4
11
7
14
tío
hijo
izquierdo
2
8
15
1
5
4

Árboles RN



Árboles RN

- Eliminación:

  Se consideran los mismos casos de eliminación que en un árbol binario ordenado:

  - Eliminar un nodo hoja.
  - Eliminar un nodo con un solo hijo (derecho o izquierdo).
  - Eliminar un nodo con dos hijos.

Árboles RN

- Eliminación:

  ... pero además hay que tener en cuenta el color del nodo a eliminar ...

  Si se elimina un nodo **rojo**, usualmente las propiedades se siguen manteniendo.

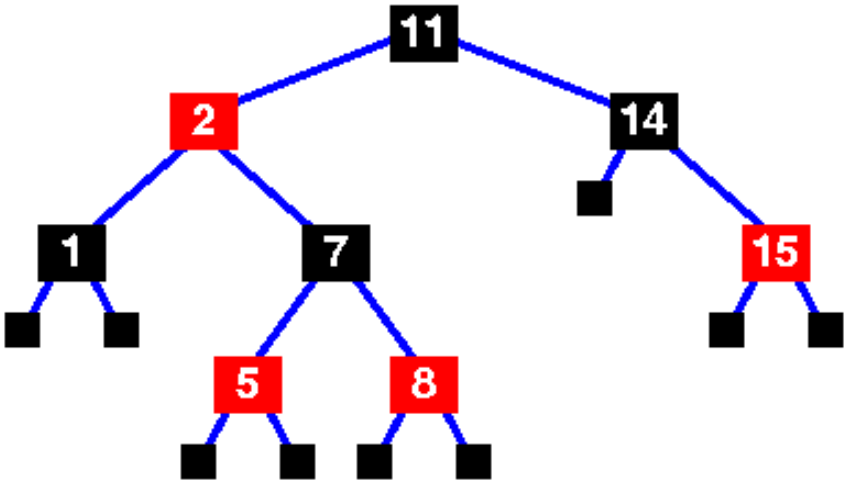
  Si se elimina un nodo **negro**, es necesario identificar las opciones de cambio de color y rotación para garantizar las propiedades del árbol.

Árboles RN

- Applet de demostración

  http://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Árbol RN en la STL



- `#include <set>`
  - `std::set< T >`
  - `std::multiset< T >`
- `#include <map>`
  - `std::map< K, T >`
  - `std::multimap< K, T >`

STL: `std::set< T >`

- Se “ve” como una lista ordenada de elementos únicos.
- T debe ser un tipo de “ordenamiento estrictamente débil” (*strict weak ordering*).
  - i.e. Debe existir el operador “<” para T.

```
std::set< T >::insert( const T& v );
std::set< T >::erase( iterator pos );
std::set< T >::[r]begin( );
std::set< T >::[r]end( );
```

STL: `std::map< K, T >`

- Se “ve” como un vector dinámico de índices de elementos únicos.
- K debe ser un tipo de “ordenamiento estrictamente débil” (*strict weak ordering*).
  - i.e. Debe existir el operador “<” para K.

```
std::map< int, float > v;
        ↔ float v[ MAX ];

std::map< K, T >::iterator it;

it->first; // K

it->second; // T
```

```
struct ltstr
{
  bool operator()(const char* s1, const char* s2) const
  {
    return strcmp(s1, s2) < 0;
  }
};

int main()
{
  map<const char*, int, ltstr> months;

  months["january"] = 31;
  months["february"] = 28;
  months["march"] = 31;
  months["april"] = 30;
  months["may"] = 31;
  months["june"] = 30;
  months["july"] = 31;
  months["august"] = 31;
  months["september"] = 30;
  months["october"] = 31;
  months["november"] = 30;
  months["december"] = 31;

  cout << "june -> " << months["june"] << endl;
  map<const char*, int, ltstr>::iterator cur  = months.find("june");
  map<const char*, int, ltstr>::iterator prev = cur;
  map<const char*, int, ltstr>::iterator next = cur;
  ++next;
  --prev;
  cout << "Previous (in alphabetical order) is " << (*prev).first << endl;
  cout << "Next (in alphabetical order) is " << (*next).first << endl;
}
```

Árboles

- Implementaciones de:

  - Árbol general (ArbolGeneral), nodo general (NodoGeneral).

  - Árbol binario ordenado (ArbolBinarioOrd), nodo binario (NodoBinario).

  - Árbol AVL (ArbolAVL), nodo AVL (NodoAVL).

  - Árbol RN (implementación STL).

Referencias

- L. Joyanes Aguilar, I. Zahonero. Algoritmos y estructuras de datos: una perspectiva en C. McGraw-Hill, 2004.
- www.cs.duke.edu/~reif/courses/alglectures/skiena.lectures/lecture10.pdf
- www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch11.html
- www.stolerman.net/studies/cs521/red_black_trees.pdf
- http://lcm.csa.iisc.ernet.in/dsa/node114.html
- en.wikipedia.org/wiki/Red-black_tree