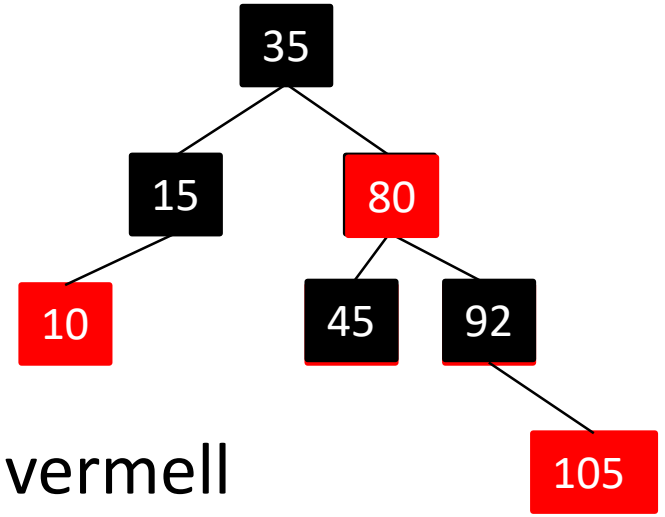
Sessió 20 Arbres

LP 2019-20

Red Black Tree. Inserció: ROTACIÓ

Tornem a l'aplicació web

- Insereix el node 9



Què ha passat? L'oncle no és vermell

Red Black Tree. Inserció: ROTACIÓ

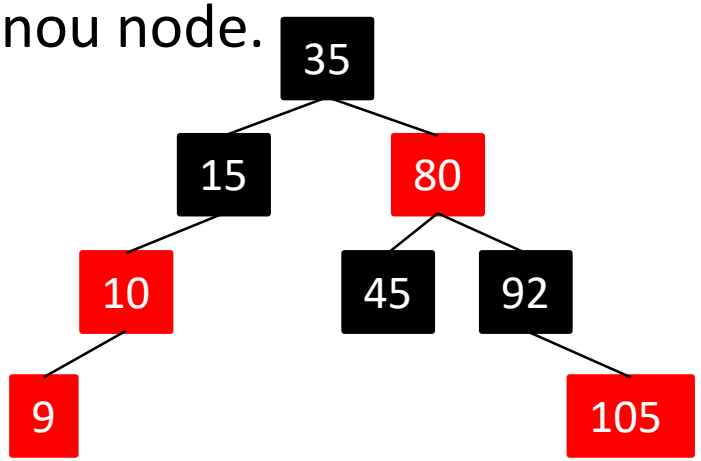
Inserció: 9

1. Utilitzem la cerca binària per saber a on va el nou node.
2. Un node nou ***(x)*** és **RED**.
3. Si ***(x)*** és l'arrel el posem a **BLACK**
4. Sino
    1. si el pare és **BLACK** ja està
    2. Sino (el pare és **RED)**
        1. Si l'oncle és **RED** (avi ha de ser **BLACK**)
        2. Si oncle és **BLACK**

NULL es considera com a **BLACK**

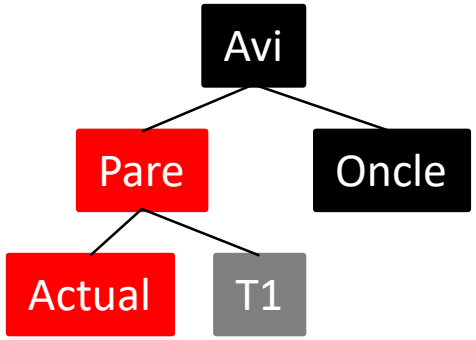
i. **Cas Esquerre-Esquerre**
ii. **Cas Esquerre-Dret**
iii. **Cas Dret-Dret**
iv. **Cas Dret-Esquerre**

mètode `rota`

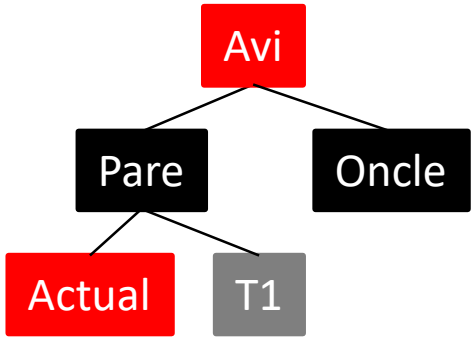


Red Black Tree. Rotacions

1. CAS: Esquerre esquerre: Node actual és fill esquerre i el seu pare també

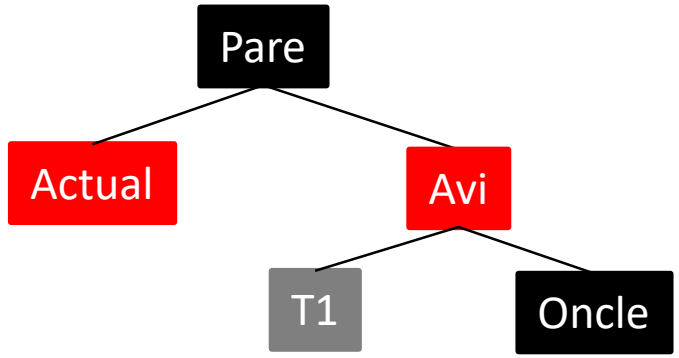


Problema: dos vermells seguits



Canvi Color Pare →Avi

Solució: Ja no hi ha 2 vermells seguits.
Problema: Ara el nombre de nodes negres no és igual per tots els camins.



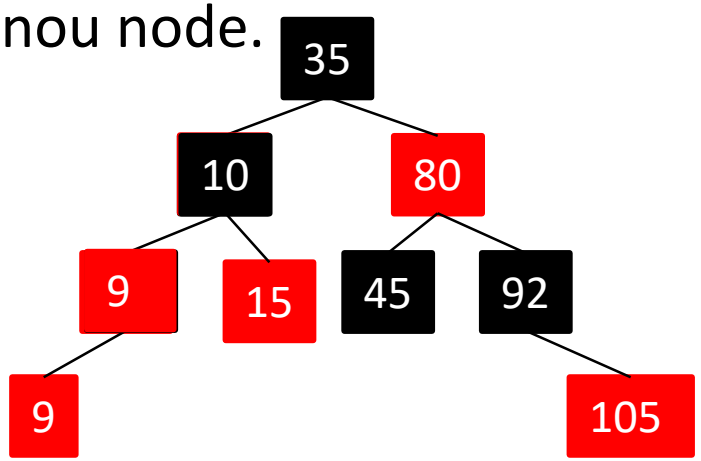
Rotació a dreta Avi

Solució: Ara ja tenim el mateix nombre de nodes negres seguits que abans.
Nota: L'arbre continua estant ordenat.
Pare<Avi
Pare<T1
Avi>T1

Red Black Tree. Inserció: ROTACIÓ

Inserció: 9

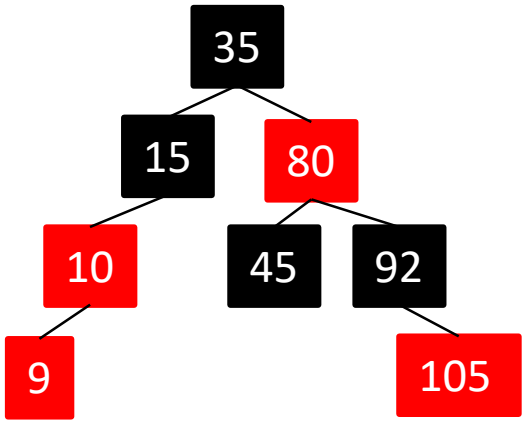
1. Utilitzem la cerca binària per saber a on va el nou node.
2. Un node nou ***(x)*** és **RED**.
3. Si ***(x)*** és l'arrel el posem a **BLACK**
4. Sino
    1. si el pare és **BLACK** ja està
    2. Sino (el pare és **RED)**
        1. Si l'oncle és **RED** (avi ha de ser **BLACK**)
        2. Si oncle és **BLACK**
            i. **Cas Esquerre-Esquerre**
                a. **Canvi color pare i avi**
                b. **Rotació dreta**



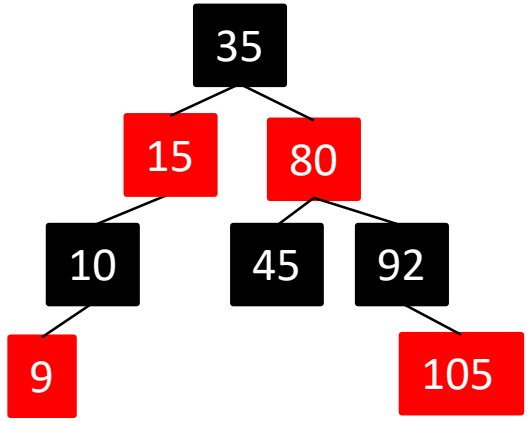
NULL es considera com a **BLACK**

Red Black Tree. Inserció: ROTACIÓ

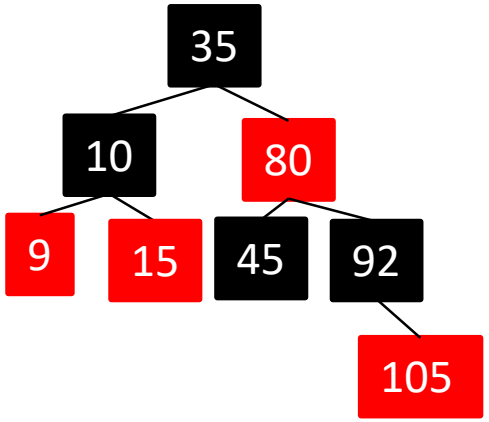
Inserció: 9



1. Cerca binària.
2. Inserim vermell



3. Pare vermell, oncle negre o null
4. Rotació cas esquerre-esquerre
   4-1. Canvi color pare-avi



4.2 Rotació dreta avi

Exercici

Modifica mètode

```
template<class T> void TreeRB<T>::arreglaREDRED(TreeRB<T>*
nouNode)
```

per tal que si l'oncle és null o és negre cridi al mètode rota()

Recordem Solució

```
template<class T> void TreeRB<T>::arreglaREDRED(TreeRB<T>* nouNode)
{       if (nouNode->m_father == NULL) //Es arrel
        {       nouNode->m_color = BLACK;     }
        else    //Mirem pare, avi i oncle
        {       TreeRB<int>* pPare = nouNode->m_father;
                TreeRB<int>* pAvi = nouNode->m_father->m_father;
                TreeRB<int>* pOncle = nouNode->oncle();
                if (pPare->m_color != BLACK)
                {//Si el pare es BLACK ja hem acabat
                        if (pOncle!=NULL) //Oncle NULL és com NEGRE
                                if (pOncle->m_color == RED)
                                {//Recoloregem
                                        pPare->m_color = BLACK;
                                        pOncle->m_color = BLACK;
                                        pAvi->m_color = RED;
                                        arreglarREDRED(pAvi);
                                }
                }
        }
}
```

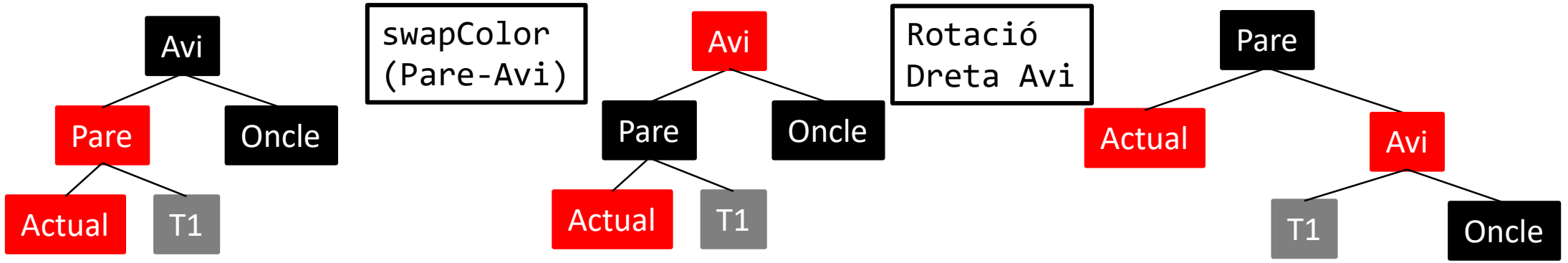
Solució

```
template<class T> void TreeRB<T>::arreglaREDRED(TreeRB<T>* nouNode)
{  if (nouNode->m_father == NULL) //Es arrel
   {nouNode->m_color = BLACK;}
   else
   {//Mirem pare, avi i oncle
        TreeRB<int>* pPare = nouNode->m_father;
        TreeRB<int>* pAvi = nouNode->m_father->m_father;
        TreeRB<int>* pOncle = nouNode->oncle();
        if (pPare->m_color != BLACK)
        {//Si el pare es BLACK ja hem acabat
            if (pOncle!=NULL)
                if (pOncle->m_color == RED) //Recoloregem
                {  pPare->m_color = BLACK;
                   pOncle->m_color = BLACK;
                   pAvi->m_color = RED;
                   arreglaREDRED(pAvi);
                }
                else{ rota(nouNode,pPare,pAvi);}
            else{ rota(nouNode, pPare, pAvi);}
        }
    }
}
```

Exercici

Implementa mètode rota pel cas esquerre-esquerre:

```
template<class T>
void TreeRB<T>::rota(TreeRB<T>* nouNode,
                     TreeRB<T>* pPare, TreeRB<T>* pAvi)
```



Per fer la rotació necessària pel cas esquerre-esquerre

Suposeu que teniu definits els mètodes:

```
template<class T>
void TreeRB<T>::rotaDreta(TreeRB<T>* pNodAct)

template<class T>
void TreeRB<T>::swapColor(TreeRB<T>* pNod1, TreeRB<T>* pNod2)
```

Solució

```
template<class T> void TreeRB<T>::rota(TreeRB<T>* nouNode, TreeRB<T>* pPare,
                                       TreeRB<T>* pAvi)
{ if (pPare->esFillEsq())
  { if (nouNode->esFillEsq())
    {//Esquerre-esquerre
       swapColor(pPare,pAvi);
    }
    else
    {//Esquerre-dret
              //Afegiriem codi per Esquerre-dret
    }
    //Per esquerre-esquerre i esquerre-dret
    rotaDreta(pAvi);
  }
  else
  {//Per rotacions Dret-Dret i Dret-Esquerre
        //Afegiriem codi
  }
}
```

Exercici

Implementa:

```
template<class T>
void TreeRB<T>::swapColor(TreeRB<T>* pNod1, TreeRB<T>* pNod2)
```

Solució

```
template<class T>
void TreeRB<T>::swapColor(TreeRB<T>* pNod1, TreeRB<T>* pNod2)
{
    COLOR c = pNod1->m_color;
    pNod1->m_color = pNod2->m_color;
    pNod2->m_color = c;
}
```

Exercici

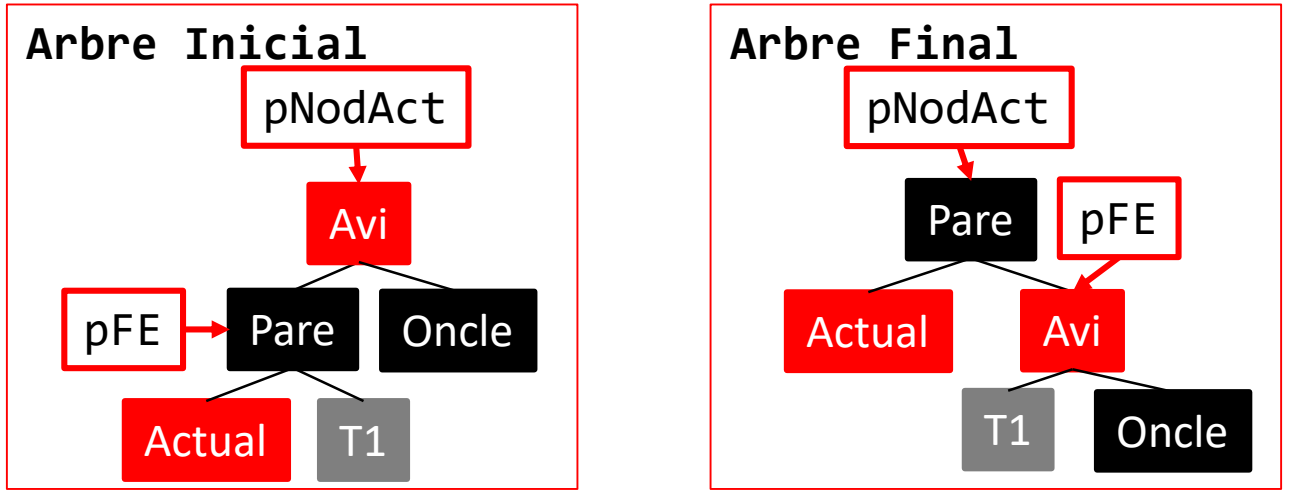
Implementa:

```
template<class T>
void TreeRB<T>::rotaDreta(TreeRB<T>* pNodAct)
```

Problema: Al fer rotacions podem estar canviant el node arrel. Però la zona de memòria a on està el node arrel ha de ser sempre el node arrel sinó perdem part de l'arbre.

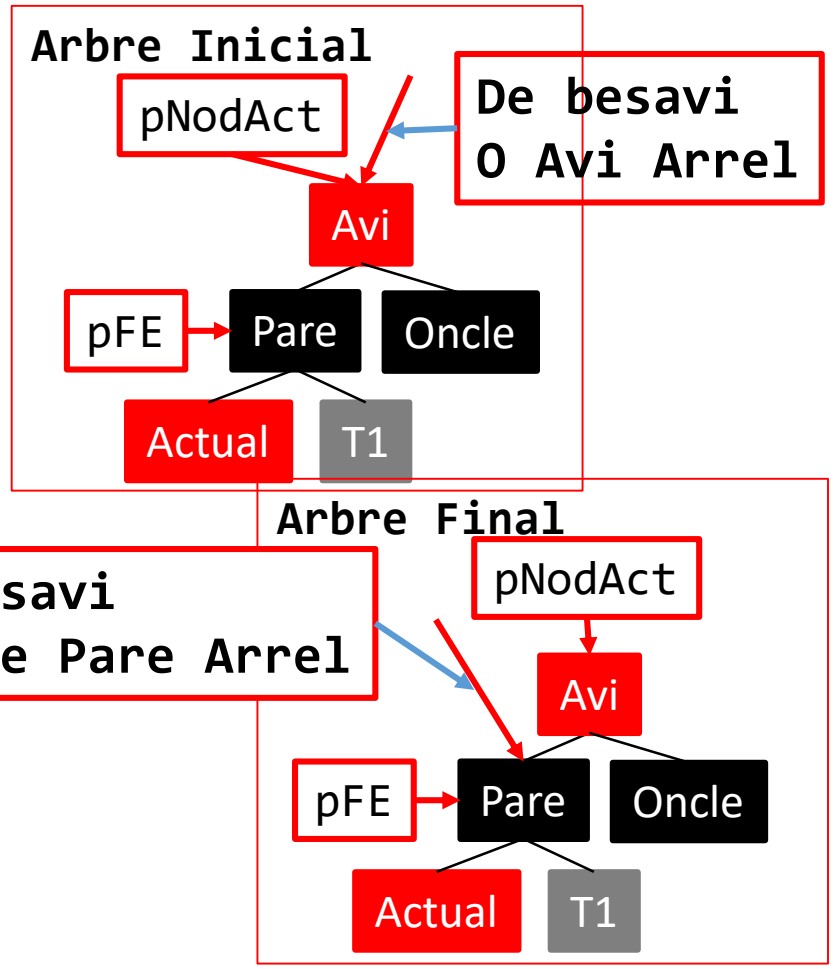
```
Tree<int> tOrd();
```

Solució: En cas que l'avi sigui arrel farem l'actualització de punters mitjançant el mètode `swapRootContents( pNodAct, pFE );`



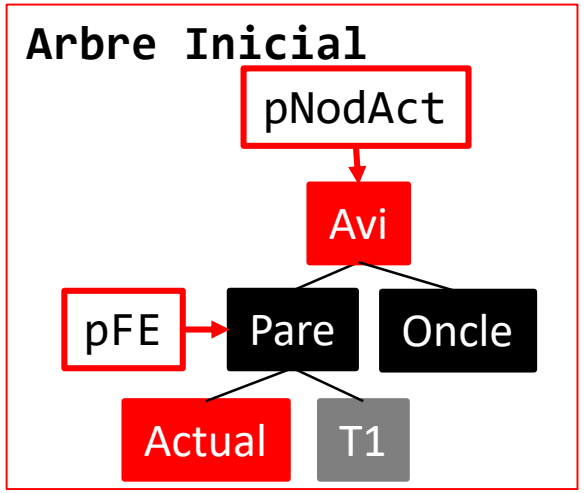
Solució

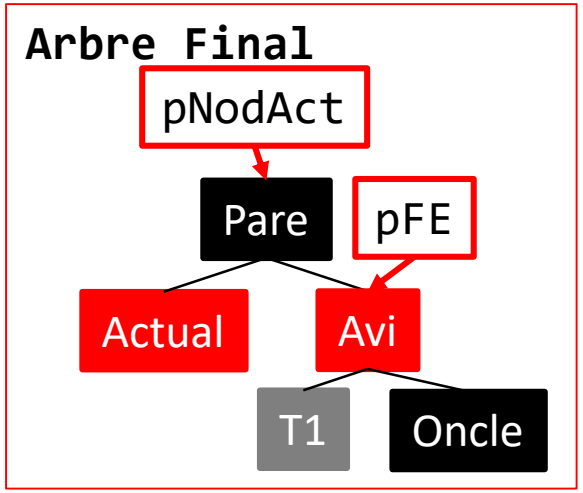
```
template<class T> void TreeRB<T>::rotaDreta(TreeRB<T>* pNodAct)
{//pNodAct passa a ser fill dret del seu fill esquerre actual
 //fill esquerre(FE) de pNodAct passara a ser pare
 //fill dret de FE (si !=NULL) passara a ser el fill esquerre de l'avi (pNodAct)
 //ATENCIO: avi pot ser arrel ➔ no movem node de lloc (intercanvi pNodAct i pFE)
    //Guardem apuntador a Node de Fill Esquerre
   TreeRB<T>* pFE = pNodAct->m_left;
   //Besavi (per no perdre el node d'on penja el subarbre)
   TreeRB<T>* pBesavi = pNodAct->m_father;
   if( pBesavi ){ // Si l'avi no és arrel
          if( pNodAct->esFillDret()){
              pBesavi->m_right = pFE;
          }else{
              pBesavi->m_left = pFE;
          }
          pFE->m_father = pBesavi;
   }else{
           this->swapRootContents( pNodAct, pFE );
   }
//...
}
```



Solució

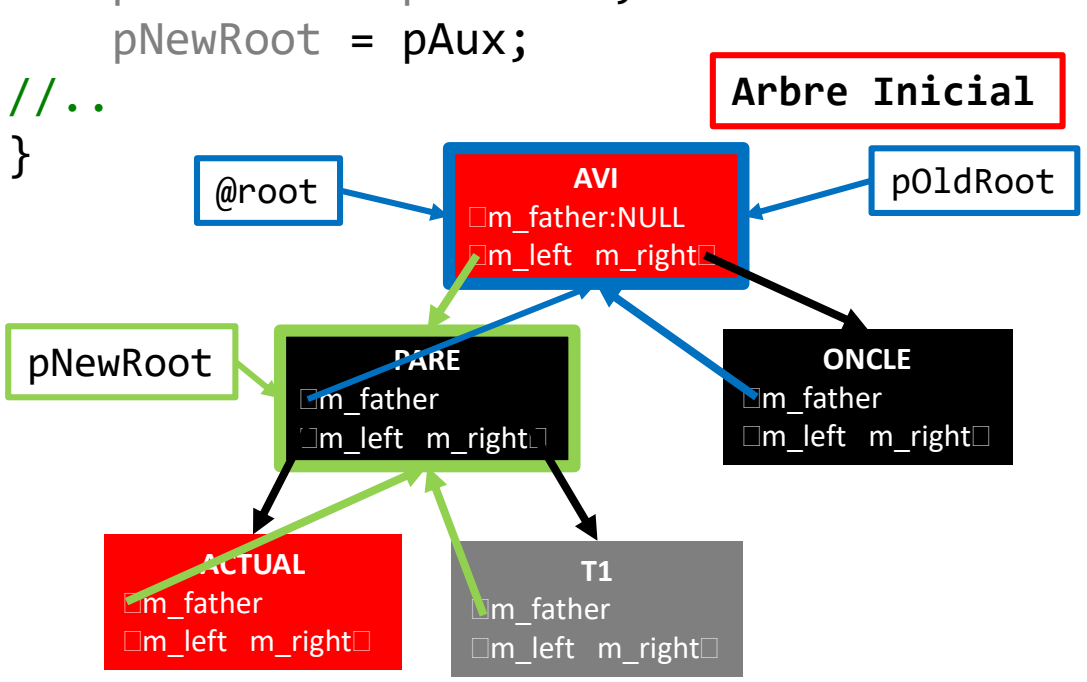
```
template<class T> void TreeRB<T>::rotaDreta(TreeRB<T>* pNodAct)
{//pNodAct pasa a ser fill dret del seu fill esquerre actual
 //fill esquerre(FE) de pNodAct passara a ser pare
 //fill dret de FE (si !=NULL) passara a ser el fill esquerre de l'avi (pNodAct)
 //ATENCIO: avi pot ser arrel ➔ no movem node de lloc (intercanvi pNodAct i pFE)
    //…
 //Fem que el fill esquerre de l'avi apunti a T1
     pNodAct->m_left = pFE->m_right;
     if( pFE->m_right){ // Si hi ha T1
         pFE->m_right->m_father = pNodAct;
     }
     //Fem que l'avi (pNodAct) apunti al pare (pFE)
     pNodAct->m_father = pFE;
     pFE->m_right = pNodAct;
}
```

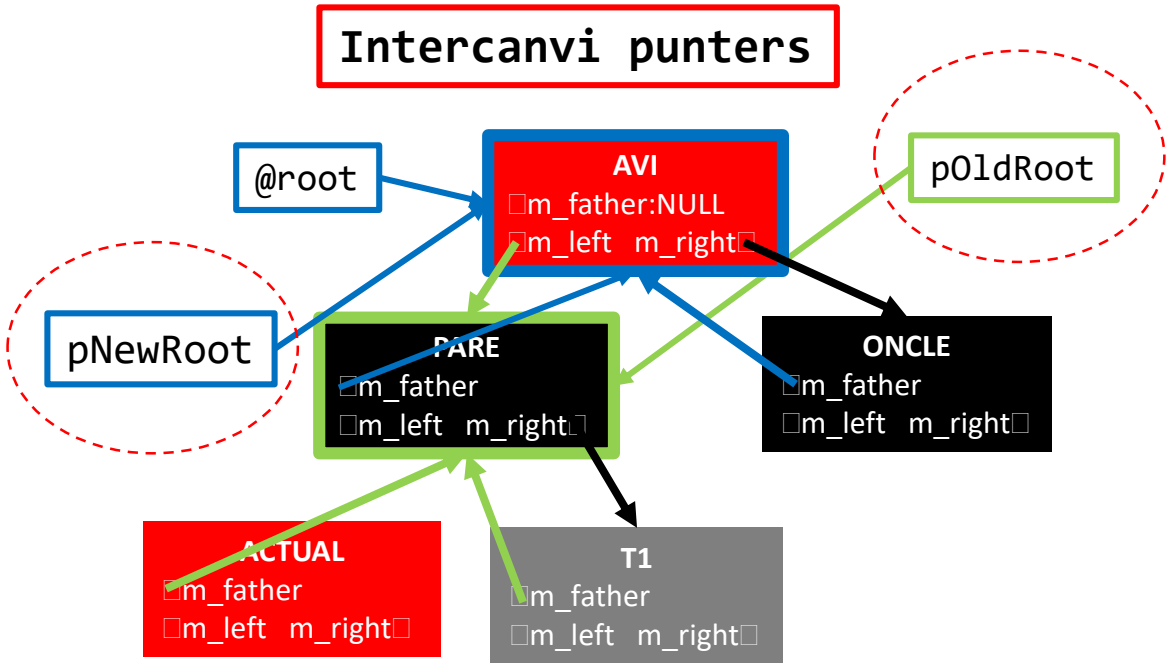




Solució

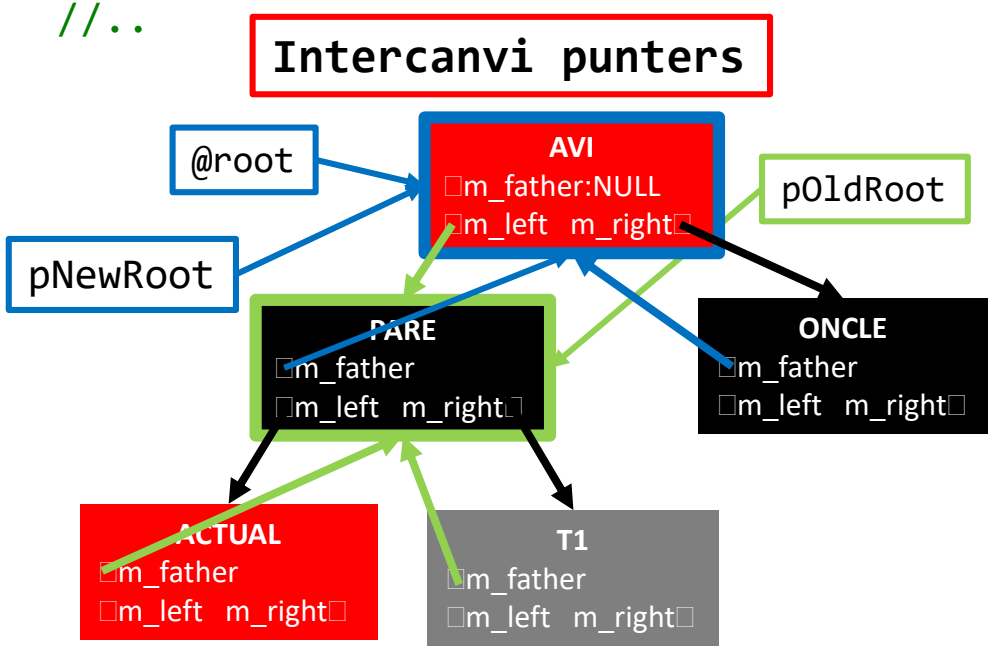
```
template<class T>
void TreeRB<T>::swapRootContents( TreeRB<T>* &pOldRoot, TreeRB<T>* &pNewRoot){
    TreeRB<T> aux;
    TreeRB<T> *pAux;
    //Intercanviem els punters
    pAux = pOldRoot;
    pOldRoot = pNewRoot;
    pNewRoot = pAux;
//..
}
```

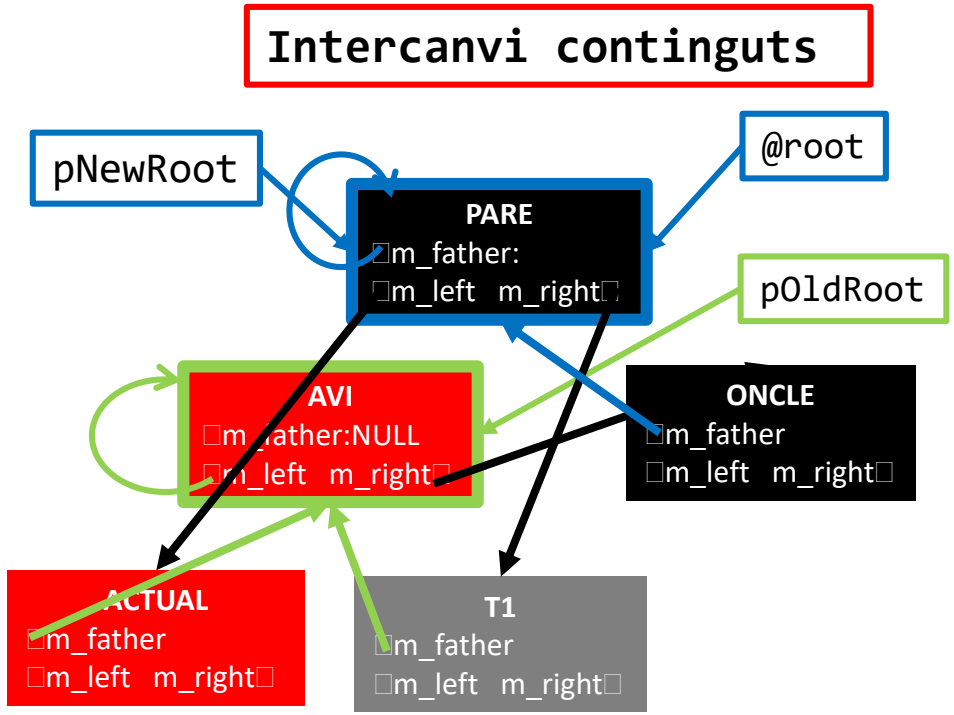




Solució

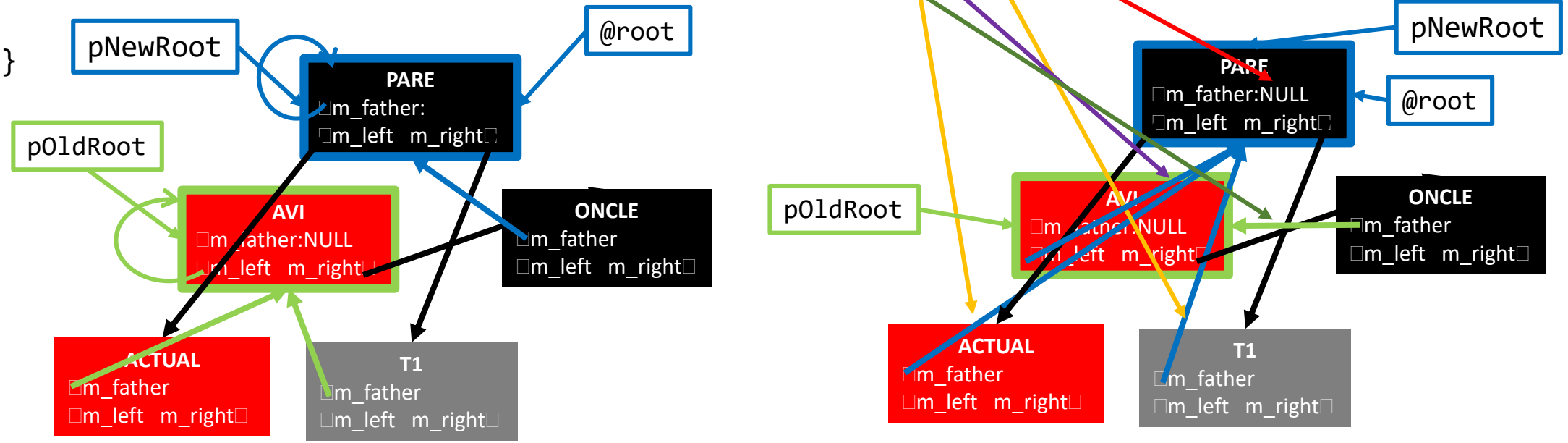
```
template<class T>
void TreeRB<T>::swapRootContents( TreeRB<T>* &pOldRoot, TreeRB<T>* &pNewRoot){
    TreeRB<T> aux;
    TreeRB<T> *pAux;
    //Intercanviem els punters
    // ...
    //Intercanviem els continguts dels punters (nodes)
    aux = *pOldRoot;
    *pOldRoot = *pNewRoot;
    *pNewRoot = aux;
    //..
}
```





Solució

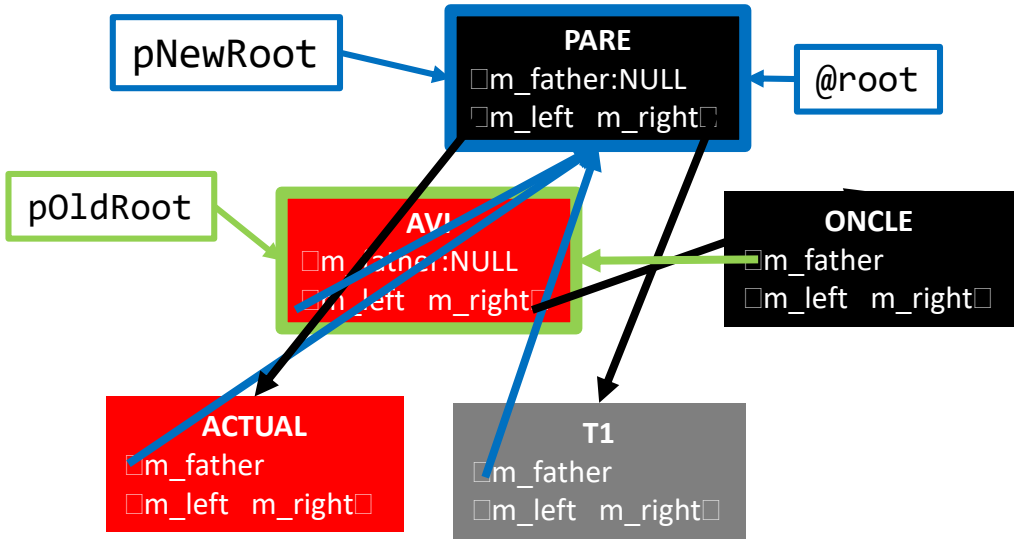
```
template<class T>
void TreeRB<T>::swapRootContents( TreeRB<T>* &pOldRoot, TreeRB<T>* &pNewRoot){
//...Actualitzem punters m_father dels fills, i de Pare. i m_left avi
    pNewRoot->m_left->m_father = pNewRoot;
    if (pNewRoot->m_right)
        pNewRoot->m_right->m_father = pNewRoot;
    pNewRoot->m_father = NULL;
    pOldRoot->m_left = pNewRoot;
    if (pOldRoot->m_right)
        pOldRoot->m_right->m_father = pOldRoot;
}
```



Solució

```
template<class T>
void TreeRB<T>::swapRootContents( TreeRB<T>* &pOldRoot, TreeRB<T>* &pNewRoot){
//..


}
```



Recolocant nodes al dibuix
Arbre final quedaria així